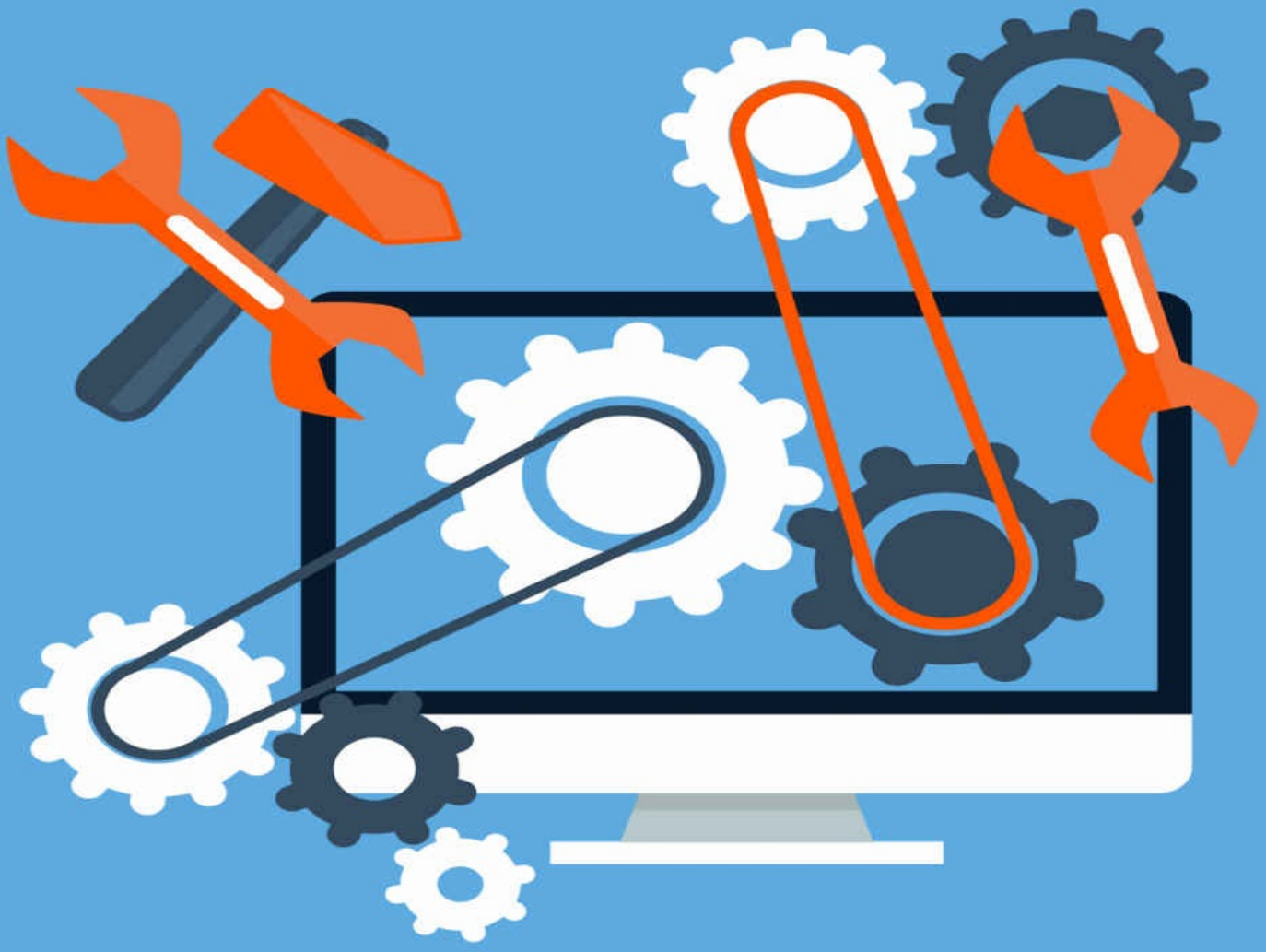
JAVA
FOR BEGINNERS

A Simple Start To Java

(Written By A Software Engineer)



SCOTT SANDERSON

Table of Contents

- Java - Basic Syntax
  - First Java Program:
  - Basic Syntax
  - Java Keywords:
  - Comments in Java
  - Using Blank Lines:
  - Inheritance:
  - Interfaces:
- Objects and Classes
- Basic Data Types
- Variable Types
- Operators in Java
  - The Arithmetic Operators
  - The Relational Operators
  - The Bitwise Operators
  - The Logical Operators
  - The Assignment Operators
  - Misc Operators
  - Conditional Operator (?):
  - instanceof Operator:
  - Precedence of Java Operators
- Loops in Java
  - The while Loop:
- Decision Making
- Strings in Java
  - String Methods
- Arrays
- Regular Expressions
  - Regular Expression Syntax
  - Methods of the Matcher Class
  - Index Methods:
  - PatternSyntaxException Class Methods:
- Methods
- File Handling
  - Byte Streams
  - FileOutputStream:
- Exception Handling
  - Throws Keyword
  - Finally Keyword
  - Creating An Exception
  - Common Exceptions
- Interfaces and Packages
- Java Applets

Java For Beginners

A Simple Start To Java Programming (Written By A Software Engineer)

Scott Sanderson

Table of Contents

Java - Basic Syntax

- First Java Program:
- Basic Syntax
- Java Keywords:
- Comments in Java
- Using Blank Lines:
- Inheritance:
- Interfaces:

Objects and Classes

Basic Data Types

Variable Types

Operators in Java

- The Arithmetic Operators
- The Relational Operators
- The Bitwise Operators
- The Logical Operators
- The Assignment Operators
- Misc Operators
- Conditional Operator (?):
- instanceof Operator:
- Precedence of Java Operators

Loops in Java

- The while Loop:

Decision Making

Strings in Java

- String Methods

Arrays

Regular Expressions

- Regular Expression Syntax
- Methods of the Matcher Class
- Index Methods:
- PatternSyntaxException Class Methods:

Methods

File Handling

Byte Streams

FileOutputStream:

Exception Handling

Throws Keyword

Finally Keyword

Creating An Exception

Common Exceptions

Interfaces and Packages

Java Applets

Other Scott Sanderson Books:

SPECIAL BONUS

Copyright 2016 by Globalized Healing, LLC - All rights reserved.

Click here to receive incredible ebooks absolutely free!

Introduction

Java, the programming language, was introduced by Sun Microsystems. This work was initiated by James Gosling and the final version of Java was released in the year 1995. However, initially Java was released as a component of the core Sun Microsystem platform for Java called J2SE or Java 1.0. The latest release of Java or J2SE is Java Standard Version 6.

The rising popularity of Java, as a programming platform and language has led to the development of several tools and configurations, which are made keeping Java in mind. For instance, the J2ME and J2EE are two such configurations. The latest versions of Java are called Java SE and Java EE or Java ME instead of J2SE, J2EE and J2ME. The biggest advantage of using the Java platform is the fact that it allows you to run your code at any machine. So, you just need to write your code once and expect it to run everywhere.

As far as the features of Java are concerned, they are as follows:

- Object Oriented

  In Java, everything is an object. Java can be effectively stretched out and extended to unimaginable dimensions since it is focused around the Object model.

- Independent of the platform

  Dissimilar to numerous other programming dialects including C and C++, when Java is aggregated, it is not converted into a form, which is particular to any machine. Instead, it is converted into a machine-independent byte code. This byte code is conveyed over the web and deciphered by Virtual Machines or JVM on whichever stage it is generally run.

- Simple

  Java is intended to be not difficult to learn. In the event that you comprehend the essential idea of OOP, Java would not be difficult to ace.

- Secure

With Java's security framework, it empowers to create frameworks, which are free of viruses and tampering. Public-key encryption is used as the core authentication strategy.

- Independent of Machine Architecture

  Java compiler produces an object file format, which is independent of the architecture of the machine. The assembled code can be executed on numerous processors, with the single requirement that they must all have Java runtime framework.

- Portability

  The fact that Java code is machine and platform independent makes it extremely compact. Compiler in Java is composed in ANSI C with a clean conveyability limit, which is a POSIX subset.

- Robustness

  Java tries to kill circumstances, which can lead to potential system failures, by stressing chiefly on runtime checking and compile time checking.

- Support for Multithreaded Applications

  With Java's multithreaded feature, it is conceivable to compose programs that can do numerous assignments at the same time. This configuration gimmick permits designers to build easily running intelligent applications.

- Interpreted Code

  Java byte code is interpreted on the fly to local machine. The advancement methodology is more quick and expository since the interfacing is an incremental and lightweight process.

- High Performance

  With the utilization of Just-In-Time compilers, Java enhances the performance of the system.

- Distributed

  Java is intended for the conveyed environment of the web.

- Dynamic

  Java is thought to be more dynamic than C or C++ since it is intended to adjust to

an advancing environment. Java projects can convey broad measure of run-time data that can be utilized to check for accesses and respond to the same on run-time.

History of Java

James Gosling started working on the Java programming language in June 1991 for utilization in one of his numerous set-top box ventures. The programming language, at first, was called Oak. This name was kept after an oak tree that remained outside Gosling's office. This name was changed to the name Green and later renamed as Java, from a list of words, randomly picked from the dictionary.

Sun discharged the first open usage as Java 1.0 in 1995. It guaranteed Write Once, Run Anywhere (WORA), giving no-expense run-times on prominent stages. On 13 November 2006, Sun discharged much of Java as free and open source under the terms of the GNU General Public License (GPL). On 8 May 2007, Sun completed the procedure, making the greater part of Java's center code free and open-source, beside a little parcel of code to which Sun did not hold the copyright.

Pre-requisites

In order to run and experiment with the examples given in this book, you shall require a Pentium 200-Mhz machine with at least 64 MB of RAM. You additionally will require the accompanying programming platforms:

- Microsoft Notepad or Any Word Processor
- Java JDK 5
- Linux 7.1 or Windows XP or higher Operating Systems

Java - Basic Syntax

A basic Java program can be broken down into several constructs and elements. Typically, it can be characterized as a collection of objects, which communicate with each other by calling each other's routines. The basic definitions of objects and classes are given below:

- Class

  A class can be described as a blueprint that portrays the practices/expresses all the behaviors and states of its objects.

- Object

  Objects are characterized by two components namely, methods and attributes or variables. For instance, if you consider the example of a puppy, then it has the following attributes or states: name, color and breed. In addition, it also has the following behaviours, which include woofing, wagging and consuming. Any object is nothing but an instance of a class.

- Instance Variables

  Each object has its set of variables. An object's state is made by the qualities alloted to these variables during program execution.

- Methods

  A method is the definition of a method. Moreover, a class can contain numerous methods. It is in these methods that behaviours like where the rationales are composed, information is controlled and all the activities are executed.

First Java Program:

In order to start with basic basic Java programming, let us look at the standard Hello World program.

```
public class MyFirstJavaProgram {

    public static void main(String []args) {

    System.out.println("Say Hello World To The World!");

    }

}
```

As you can see, the program uses a single line of code in the main() function, which prints

the statement ‘Hello World!’. However, before that can be done, let us look at the steps that you must follow in your quest to execute the file.

- Open any text editor and paste this code in that file.
- Save the file with a .java extension. For example, you can save the file as Sample.java.
- The next step is to to open the command prompt of the system and relocate its reference to the directory in which the file is saved. For instance, if you have saved the file in C:\, then you must take the prompt to the same directory.
- In order to compile the code, you must type the following:

  javac Sample.java

- If there are no errors, you will automatically be taken to the next line. You can now execute the code using the following command:

  java Sample.java

- You should be able to see the following output on the screen.

  Say Hello World To The World!

Basic Syntax

About Java programs, it is paramount to remember the accompanying points.

- Class Names –

  For all class names, the first letter ought to be in Upper Case.

  On the off chance that few words are utilized to structure a name of the class, every internal word’s first letter ought to be in Upper Case. For example, a standard class name is as follows:

  class Sampleclass

- Case Sensitivity - Java is case sensitive, which implies that the identifier Hi and hi would have distinctive importance in Java.
- Method Names - All system names ought to begin with a Lower Case letter. In the event that few words are utilized to structure the name of the method, then every internal word’s first letter ought to be in Upper Case. An example of this convention is follows:

  public void mysamplemethod ()

- Filename –

  The name of the system record ought to precisely match the class name. At the point when you are saving the file, you ought to save it utilizing the class name. Remember Java is case touchy and affix ".java" to the end of the name. If the document name and the class name don't match your system won't assemble. Consider the example of a class name Sample. In this case, you must name the file as sample.java.

- public static void main(string args[])

  Java system handling begins from the main() function, which is a required piece of each Java program.

Java Identifiers

All Java components require names. Names utilized for classes, variables and strategies are called identifiers. In Java, there are a few focuses to recall about identifiers. They are as per the following standard:

- All identifiers ought to start with a letter (beginning to end or a to z), underscore (_) or special character ($).
- After the first character, identifiers can have any mix of characters.
- You cannot use a keyword as an identifier.
- Most significantly, identifiers are case sensitive. So, Sample is not same as sample.
- Examples of identifiers include $salary, age, __1_value and _value.
- Examples of illicit identifiers include –compensation and 123abc.

Java Modifiers

Like is the case with any programming language, it is conceivable to alter classes and systems by utilizing modifiers. There are two classifications of modifiers:

- Access Modifiers: public, default, protected and private
- Non-access Modifiers: strictfp, final and abstract

We will be researching more insights about modifiers in the following chapters.

Java Variables

Several types of variables are supported by Java. These types of variables include:

- Instance Variables (Non-static variables)
- Class Variables (Static Variables)

- Local Variables

Java Arrays

Arrays are contiguous memory locations that store different variables of the same sort. On the other hand, an array itself is an article on the memory. We will research how to proclaim, develop and instate these in the chapters to follow.

Java Enums

Enums were introduced as part of the Java package in java 5.0. Enums limit a variable to have one of just a couple of predefined qualities. The qualities in this identified list are called enums. With the utilization of enums it is conceivable to diminish the quantity of bugs in your code. Case in point, in the event that we consider an application for a cafe, it would be conceivable to limit the mug size to extra large, large, medium and small. This would verify that it would not permit anybody to request any size other than the sizes mentioned in the menu or application listing.

Please note that enums can be pronounced as their own or inside a class. However, routines, variables, constructors can be created inside the body of enums as well.

Java Keywords:

Keywords or reserved words in Java are shown in the table below. As a rule, these words cannot be used as names for variables or constants.

- assert
- abstract
- break
- boolean
- case
- byte
- char
- catch
- const
- class
- default
- continue
- double

- do
- enum
- else
- final
- extends
- float
- finally
- goto
- for
- implements
- if
- instanceof
- import
- int
- long
- interface
- new
- native
- private
- package
- protected
- return
- public
- static
- short
- super
- strictfp
- synchronized
- switch
- throw
- this
- transient
- throws
- while
- try

- volatile
- void

Comments in Java

Just as in the case of C++ and C, Java supports two types of comments namely, single line comments and multi-line comments. The syntax for these types of comments are as follows:

Single line comment:

//<comment>

Multiple line comment:

/*<comment>*/

All characters that exist in the comments region are simply ignored by the Java compiler.

Using Blank Lines:

Any line that is only composed of whitespace characters or comments is considered a blank line. These lines are just ignored by the compiler and are not included in the executable.

Inheritance:

Java supports inheritance. In other words, it is possible to derive one class from another class in this programming language. For instance, if you need to create a new class, which is an extension of an existing class, then you can simply derive this new class from an existing class. This allows the new class to access the elements already implemented in the existing class. In this case, the new class is called the derived class and the existing class is referred to as the super class.

Interfaces:

As mentioned previously, Java is all about interaction between objects. The manner in which different objects communicate with each other is defined in what is called an ‘interface.’ Moreover, interfaces are also an important aspect of the inheritance feature of java. As part of an interface, the methods that can be used by a derived or sub-class are declared. However, all the methods declared as usable for the subclass must be implemented in the subclass.

Objects and Classes

Java is an Object-Oriented programming language. As an issue that has the Object Oriented peculiarity, Java underpins the accompanying essential ideas:

- Inheritance
- Polymorphism
- Abstraction
- Encapsulation
- Objects
- Message Parsing
- Classes
- Method
- Instance

In this part, we will investigate the concepts of Classes and Objects.

- Class - A class can be described as an blueprint that declares and defines the attributes and methods that its objects will implement and use.
- Object - Objects are simple real world entities that possess a state and its characteritic behaviour.

For example, if you consider a real world entity, a labrador dog, then this dog is an object. However, it belong to the class of dogs. Therefore, the associated class is Dog.

Objects in Java

Let us now look profoundly into what are objects. In the event that we consider this present reality, we can discover numerous entities around us, Cars, Humans, Dogs and several other. N fact, any real world entity can be modelled as an object. The one common thing between all these entities is the fact that they contain states and behaviours. On the off chance that we consider a dog, then its state is - breed, name and color. However, its behaviour includes eating habits and other characteristics like running and barking.

Classes in Java

A class is a blue print from which individual objects are made. A specimen of a class is given underneath:

```
open class Dogs {
```

String breed;

String shade;

int age;

void eating (){ }

void barking (){ }

}

A class can contain any of the accompanying variable sorts.

- Local variables

  Variables that are declared and used inside routines, constructors or pieces of code are called local variables. The variable will be proclaimed and instated inside the method or scope and the variable will be destroyed when the execution of a method terminates.

- Instance variables

  Instance variables are variables inside a class yet outside any system. These variables are instantiated when the class is stacked. These variables can be gotten to from inside any technique, constructor or squares of that specific class.

- Class variables

  Class variables will be variables, which are declared within a class, outside any system, with the static word before them.

A class can have any number of routines to get to the estimation of different sorts of methods. In the above illustration, eating() and barking() are the used methods. Underneath specified are a percentage of the vital subjects that need to be examined when researching classes of the Java Language.

Constructors

At the point when talking about classes, a standout amongst the most vital sub theme would be constructors. Each class has a constructor. In the event that we don’t unequivocally compose a constructor for a class, the Java compiler manufactures a default constructor for that class. Each time an object is made, no less than one constructor will be summoned.

The fundamental principle of constructors is that they ought to have the same name as the

class. A class can have more than one constructor and depending on the parameters given and return type expected, the matching constructor is called. A sample implementation for this type of a method is given below:

```
public class Puppies{

public Puppies(){

}

public Puppies(string puppyname){

}
```

The above class has two constructors. One of the constructors requires no parameters. However, the other constructor requires a string equivalent to the name of the puppy. Java additionally upholds Singleton Classes where you would have the capacity to make one and only object of a class.

Making Objects

As specified previously, a class gives the outlines to object creation. So, fundamentally an object is made from a class. In Java, the new essential word is utilized to make new objects.

There are three steps involved in the creation of any object. These steps are illustrated below:

- Declaration: A variable assertion with a variable name and object type.
- Instantiation: The “new” word is utilized to make the object of an already declared class.
- Initialization: The “new” word is trailed by a call to a constructor. This call instantiates the class and creates an object of the same, as a result.

Sample implementation is given below for better understanding of the concept.

```
public class Puppies{

public Puppies(string name){

        System.out.println(“Passed Name of the puppy is:” + name );

}

public static void main(string []args){
```

```
    Puppies samplepuppy = new Puppies( “jimmy” );
}
```

On the off chance that we compile and run the above project, then it would deliver the accompanying result:

Passed Name of the puppy is: jimmy

Getting to Instance Variables and Methods:

Variables and methods are gotten to by means of made objects of classes. To get to a variable, the qualified way ought to be the following:

The following statement creates an object.

Newobject = new Constructormethod();

The following statements can be used to access the variable and method associated with the object.

Newobject.variablesname;

Newobject.methodsname();

A sample implementation of this concept is given below:

```
public class Dog{
int dogAge;
public dogAge(String dogname){
    System.out.println(“Dog Name Passed is :” + dogname );
}
public void initAge( int dogage ){
    dogAge = dogage;
}
public int getDogAge( ){
    System.out.println(“Dog’s present age is:” + dogAge );
    return dogAge;
}
```

```
public static void main(String []args){
        Dog myDog = new Dog( “jimmy” );
        myDog.initAge( 5 );
        myDog.getDogAge( );
        System.out.println(“Variable dogAge Value is:” + myDog.dogAge );
}
}
```

Upon compilation and execution of the following code, you shall be able to see the following result.

Variable dogAge Value is: 5

Declaration Guidelines for Source Files

As the last piece of this area how about we now investigate the source file declaration standards. These tenets are key when declaring classes, importing declarations and packages in a source file.

- There can be stand out public class for every source record.
- A source document can have numerous non public classes.
- The public class name ought to be the name of the source document. The name of the source file must be affixed by the string .java. For instance, the class name is public class Employeerecord{}, then the source document ought to be saved as Employeerecord.java.
- If the class is declared inside a package, then the package articulation ought to be the first proclamation in the source record.
- If import articulations are available, then they must be composed between the package proclamation and the class revelation. On the off chance that there are no package proclamations, then the import articulation ought to be the first line in the source file.
- Import and package articulations will intimate to all the classes show in the source record. It is impractical to announce diverse import and/or package explanations to distinctive classes in the source file.

Classes have a few access levels. Moreover, there are diverse sorts of classes, which include final classes, in addition to several others. Separated from the aforementioned

sorts of classes, Java likewise has some uncommon classes called Inner classes and Anonymous classes.

Java Packages

Basically, it is a method for classifying the classes and interfaces. At the point when creating applications in Java, many classes and interfaces will be composed. In such a scenario, ordering these classes is an unquestionable requirement and makes life much less demanding. In Java, if a completely qualified name, which incorporates the class and package name, is given, then the compiler can without much of a stretch find the source code or classes. Import declarations is a method for giving the correct area for the compiler to find that specific class.

Case in point, in order to load all the classes accessible in java_installation/java/io, you must use the following statement:

import java.io.*;

A sample implementation for this concept is given below:

The following code uses two classes Employeerecord and Employeerecordtest. The first step is to open the text editor you intend to use at your system and copy and paste the code shown below into the text editor application. Keep in mind that the public class in the code is Employeerecord. Therefore, the name of the file should be Employeerecord.java. This class uses the variables, methods and constructor as shown below:

```
import java.io.*;

public class Employeerecord {

int empage;

String empname;

double empcompensation;

public Employee(string empname){

this.empname = empname;

}

public void employeeage(int employeeage){

empage =  employeeage;

}
```

```
public void empcompensation(double empcompensation){
empcompensation = empcompensation;
}
public void printemp(){
System.out.println(“empname:”+ empname );
System.out.println(“empage:” + empage );
System.out.println(“empcompensation:” + empcompensation);
}
```

As specified awhile ago in this exercise, handling begins from the main function. Accordingly, with this goal, we should create a main function for this Employeerecord class. Given beneath is the Employeerecordtest class, which makes two instances of the class Employeerecord and conjures the techniques for each one item to allot values for every variable. You can save this file as Employeerecordtest.java.

```
import java.io.*;
public class Employeerecordtest{
public static void main(String args[]){
Employeerecord employee1 = new Employeerecord(“Jack Wright”);
Employeerecord employee2 = new Employeerecord(“Mary John”);
employee1.employeeage(32);
employee1.empcompensation(5000);
employee2.employeeage(25);
employee2.empcompensation(2000);
employee1.printemp();
employee2.printemp();
}
}
```

Upon compilation and execution, you must get the following output:

```
empname: Jack Wright
```

empage: 32

empcompensation: 5000

empname: Mary John

empage: 25

empcompensation: 2000

Basic Data Types

Variables are only saved memory areas to store values. This implies that when you make a variable, you save some space in memory. In light of the data type of a variable, the working framework distributes memory and chooses what can be put in the held memory. Consequently, by appointing diverse data types to variables, you can store whole numbers, decimals, or characters in these variables.

There are two data types accessible in Java:

- Reference/Object Data Types
- Primitive Data Types

Primitive Data Types

There are eight primitive information types, which are supported by Java. Primitive data types are predefined by the dialect and named by a catchphrase. This section discusses these data types in detail.

byte:

- byte information sort is a 8-bit marked two's supplement whole number.
- Maximum worth is 2^7 -1, which is equal to 127. This value is also included in the range of these values.
- Minimum worth is -2^7, which is equal to -128.
- Default value stored in a variable of this type is 0.
- byte information sort is utilized to spare space in vast exhibits, principally set up of numbers, since a byte is four times littler than an int.
- Example:

  byte x = 200, byte y = -20

short:

- short information sort is a 16-bit marked two's supplement number.
- Maximum value is 2^15 -1, which is equal to 32,767. This number is also included in the range.
- Minimum value is -2^15, which is equal to -32,768.
- short information sort can likewise be utilized to spare memory as byte information sort. A short is 2 times littler than an int
- The default value for this data type is 0.

- Example:

  short x = 425164, short y = -76686

int:

- int information sort is a 32-bit marked two's supplement number.
- Maximum value for this data type is 2^31 -1, which is equal to 2,147,483,647. This number is also included in the range for this data type.
- Minimum value for this data type is -2^31, which is equal to - 2,147,483,648.
- int is for the most part utilized as the default information sort for its indispensable qualities unless there is a worry about memory.
- The default value for this data type is 0.
- Example:

  int x = 826378, int y = -64782

long:

- long information sort is a 64-bit marked two's supplement whole number.
- Maximum value for this data type is 2^63 -1, which is equal to 9,223,372,036,854,775,807.
- Minimum value for this data type is -2^63, which is equal to -9,223,372,036,854,775,808.
- This sort is utilized when a more extensive memory range than int is required.
- The default value for those data type is 0l.
- Example:

  long x = 174636l, int y = -536452l

float:

- float is a data type, which is know for its solitary exactness, 32-bit IEEE 754 gliding point.
- float is for the most part used to spare memory in vast exhibits of coasting point numbers.
- The default value for this data type is 0.0f.
- float information sort is never utilized for exact values, for example, money.
- Example:

  float x = 254.3f

double:

- double information sort is a float with twofold exactness 64-bit IEEE 754 drifting point.
- This information sort is for the most part utilized as the default information sort for decimal qualities.
- double information sort ought to never be utilized for exact values, for example, money.
- The default value for this data type is 0.0d.
- Example:

```
double x = 321.4
```

boolean:

- boolean information sort speaks to one bit of data.
- Any boolean variable can assume one of the two values: true or false.
- This information sort is utilized for basic banners that track genuine/false conditions.
- The default value for this data type is false.
- Example:

```
boolean check = true;
```

char:

- char information sort is a solitary 16-bit Unicode character.
- Maximum value for a variable of this type is “\uffff” (or 65,535 comprehensive).
- Minimum value for a variable of this type is “\u0000” (or 0).
- char information sort is utilized to store any character.
- example:

```
char text =‘a’
```

Reference Data Types

- Reference variables are made utilizing characterized constructors of the classes. They are utilized to get to objects. These variables are proclaimed to be of a particular data type that can’t be changed. A few examples of such data types are Employee and Dog.
- Class objects, and different kind of variables go under reference data type.
- Default estimation of any reference variable is invalid.

- A reference variable can be utilized to allude to any object of the announced sort.
- Example: myanimal = new Animals(“rabbit”);

Java Literals

A literal in Java is a source code representation of a settled worth. They are spoken to specifically in the code without any calculation. Literals can be appointed to any primitive sort variable. Case in point:

byte x = 86;

char x = “a”

int, byte, short and long can be communicated in hexadecimal(base 16), decimal(base 10) or octal(base 8) number frameworks too. Prefix 0 is utilized to show octal while prefix 0x demonstrates hexadecimal when utilizing these number frameworks for literals. For example,

int numd = 134;

int numo = 0243;

int numx = 0x95;

String literals in Java are determined like they are in most different programming languages by encasing a grouping of characters between a couple of twofold quotes. Illustrations of string literals are:

“Hi Everyone” “two\nlines” “"these characters are inside quotes"”

String sorts of literals can contain any Unicode characters. For instance:

String news = “\u0001”

You can also use escape sequences with Java. Here is a list of escape sequences that you can use.

Double quote - "

Carriage return (0x0d) - \r

Newline (0x0a) - \n

Single quote - '

Backspace (0x08) - \b

Formfeed (0x0c) - \f

Tab - \t

Space (0x20) - \s

Octal character (ddd) - \ddd

Backslash - \

Hexadecimal UNICODE character (xxxx) - \uxxxx

Variable Types

A variable gives us named capacity that our code can control. Every variable in Java has a particular sort, which decides the size and format of the variable's memory; the scope of values that can be put away inside that memory; and the set of operations that can be connected to the variable. You must make an explicit declaration of all variables before they can be utilized. Variables can be declared in the following manner:

Data type <variable name>;

Here data type is one of Java's datatypes. On the other hand, a variable is the name or the identifier associated with the variable. To pronounce more than one variable of the pointed out type, you can utilize a comma-divided rundown. Here are a few examples of declarations:

The following declaration declares three integer variables.

int x, y, z;

In a similar manner, variables of other data types may also be declared.

Java supports three types of variables. These types are as follows:

- Class/static variables
- Instance variables
- Local variables

Local Variables

- Local variables are announced in systems, constructors, or scopes.
- Local variables are made when the constructor or method is entered and the variable will be decimated once it retreats the system, constructor or scope.
- Access modifiers can't be utilized for neighborhood variables.
- Local variables are noticeable just inside the announced method, constructor or scope.
- Local variables are executed at stack level.
- There is no default value for these variables. So, local variables ought to be declared and a beginning value ought to be relegated before the first utilization.

Sample Implementation:

Here, age is a neighborhood variable. This is characterized inside pupage() strategy and its

degree is constrained to this system just.

```
public class myTest{

open void newfunc(){

int myvar = 1;

myvar = myvar + 10;

System.out.println(“The value of myvar is: ” + myvar);

}

public static void main(string args[]){

mytest = new myTest ();

mytest.myfunc();

}
```

The output of the execution of this code is:

The value of myvar is: 11

Instance Variables

- The declaration of an instance variable is made inside the class. However, it is made outside the system, constructor or any scope.
- Instance variables are made when an object is made with the utilization of the keyword “new” and obliterated when the item is destroyed.
- When a space is dispensed for an item in the memory, an opening for each one variable value is made.
- Instance variables can be pronounced in class level before or after utilization.
- Instance variables hold values that must be referenced by more than one method, constructor or piece, or key parts of an object’s express that must be available all through the class.
- Access modifiers can be given for sample variables.
- Instance variables have default values. For numbers, the default quality is 0. However, for Booleans, it is false and for object references, it is invalid. Qualities can be relegated amid the statement or inside the constructor.
- The case variables are unmistakable for all methods, constructors and scope in the class. Regularly, it is prescribed to make these variables private (access level). However perceivability for subclasses can be given with the utilization of access

modifiers for these variables.

- Instance variables can be gotten to by calling the variable name inside the class. The following statement can be used for this purpose: Objectreference.variablename.

Sample Implementation:

```
import java.io.*;
public class Employeerecord{
public String empname;
private double empcompensation;
public Employee (String name){
empname = name;
}
public void initsalary(double empsalary){
empcompensation = empsalary;
}
public void printemployee(){
System.out.println(“Employee name  : ” + empname );
System.out.println(“Employee salary :” + empcompensation);
}
public static void main(string args[]){
Employeerecord employee1 = new Employeerecord(“Mary”);
employee1.initsalary(7500);
employee1.printemployee();
}
```

The compilation and execution would deliver the accompanying result:

```
Employee name  : Mary
Employee compensation :7500.0
```

Class/Static Variables

- Class variables otherwise called static variables are declared with the static keyword in a class, yet outside a constructor, method or scope.
- There would just be one duplicate of each class variable for every class, paying little mind to what number of objects are made from it.
- Static variables are seldom utilized other than being pronounced as constants. Constants are variables that are announced as private/public, static and final. Consistent variables never show signs of change from their introductory quality.
- Static variables are put away in static memory. It is uncommon to utilize static variables other than announced final and utilized as either private or public constants.
- Static variables are made when the system begins and annihilated when the execution stops.
- Visibility is like instance variables. In any case, most static variables are announced public since they must be accessible for clients of the class.
- Default values for these variables are also same as instance variables. For numbers, the default value id typically 0. However, the same value for Booleans is false and for object reference is invalid. Values can be doled out amid the assertion or inside the constructor. Furthermore, values can be appointed in unique static initializer brackets.
- Static variables can be gotten to by calling with the class name . Classname.variablename.
- When announcing class variables as public static final, variables names (constants) must all be in upper case. Moreover, the static variables are not public and the naming convention is the same as local and instance variables.

Sample Implementation:

```
import java.io.*;

public class Employeerecord{

private static double empcompensation;

public static final String empdept = “HR “;

public static void main(string args[]){

empcomp = 7500;
```

System.out.println(empdept+” Compensation: “+empcompensation);

}

The compilation and execution of this code shall create the accompanying result:

HR Compensation: 7500

Modifier Types

Modifiers are catchphrases that you add to definitions to change their implications. The Java programming language has a wide and mixed bag of modifiers, including the accompanying:

- Non-Access Modifiers
- Java Access Modifiers

In order to utilize a modifier, you incorporate its catchphrase in the meaning of a class, variable or method. The modifier goes before whatever is left of the announcement.

Access Control Modifiers:

Java gives various access modifiers to set access levels for classes, variables, routines and constructors. The four right to gain access are:

- Private: visible to the class.
- Default: visible to the bundle. No modifiers are required.
- Secured: visible to all subclasses and package.
- Public: visible to the world.

Non Access Modifiers:

Java gives various non-access modifiers to attain numerous other usefulness.

- Static:

  The static modifier for making class variables and methods.

- Final

  The final modifier for concluding the executions of classes, variables and methods.

- Abstract

  This modifier is used for for creating abstract methods and classes.

- Volatile and Synchronized

These modifiers are typically used for threads.

Operators in Java

The operator set in Java is extremely rich. Broadly, the operators available in Java are divided in the following categories.

- Relational Operators
- Arithmetic Operators
- Logical Operators
- Bitwise Operators
- Misc Operators
- Assignment Operators

The Arithmetic Operators

Operations in Java are used in essentially the same manner as in algebra. They are used with variables for performing arithmetic operations. Here is a list of arithmetic operators available in Java.

Operation	Operator	Description
Addition	+	Adds the values of two variables
Subtraction	-	Subtracts the values of two variables
Multiplication	*	Multiplies the values of two variables
Division	/	Divides the values of two variables
Modulus	%	The resultant value is the the remainder of division
Increment	++	Increases the value by 1
Decrement	—	Decreases the value by 1

The Relational Operators

Java also supports several relational operators. The list of relational operators that are supported by Java are given below.

Operation	Operator	Description
Equal To	==	Compares the values of two variables for equality
Not Equal To	!=	Compares the values of two variables for inequality
Greater Than	>	Checks if one value is greater than the other value
Lesser Than	<	Checks if one value is lesser than the other value
Greater Than Or Equal To	>=	Checks if one value is greater than or equal to the other value
Lesser Than Or Equal To	<=	Checks if one value is lesser than or equal to the other value

The Bitwise Operators

The bitwise operators available in Java can be easily applied to a number of data types. These data types include byte, short, long, int and char. Typically, any bitwise operator performs the concerned operation bit-wise. For instance, if you consider the example of an integer x, which has the value 60. Therefore, the binary equivalent of x is 00111100. Consider another variable y, with the value 13 or 00001101. If we perform the bitwise operation & on these two numbers, then you will get the following result:

x&y = 0000 1100

The table shown below shows a list of bitwise operators that are available in Java.

Operation	Operator	Description
BINARY AND	&	Performs the AND operation

BINARY OR	|	Performs the OR operation
BINARY XOR	^	Performs the XOR operation
ONE'S COMPLEMENT	~	Performs the complementation operation on a unary variable
BINARY LEFT SHIFT	<<	Performs the left shifting of bits
BINARY RIGHT SHIFT	>>	Performs the right shifting of bits

In addition to the above mentioned, Java also supports right shift zero fill operator (>>>), which fills the shifted bits on the right with zero.

The Logical Operators

Logical operators are an integral part of any operator set. The logical operators supported by Java are listed in the table below.

Operation	Operator	Description
Logical AND	&&	Returns True if both the conditions mentioned are true
Logical OR	||	Returns True if one or both the conditions mentioned are true
Logical NOT	!	Returns True if the condition mentioned is False

The Assignment Operators

There are following assignment operators supported by Java language:

Operation	Operator	Description
Simple assignment operator	=	Assigns a value on the right to the variable in the left
Add - assignment operator	+=	Adds the value on the right to the value of the variable on the left and assigns the resultant to the variable on the left
Subtract - assignment operator	-=	Subtracts the value on the right to the value of the variable on the left and assigns the resultant to the variable on the left
Multiply - assignment operator	*=	Multiplies the value on the right to the value of the variable on the left and assigns the resultant to the variable on the left
Divide - assignment operator	/=	Divides the value on the right to the value of the variable on the left and assigns the resultant to the variable on the left
Modulus - assignment operator	%=	It takes the modulus of the LHS and RHS and assigns the resultant to the variable on the left
Left shift - assignment operator	<<=	It takes the left shift of the LHS and RHS and assigns the resultant to the variable on the left

Right shift - assignment operator	>>=	It takes the right shift of the LHS and RHS and assigns the resultant to the variable on the left
Bitwise - assignment operator	&=	It takes the bitwise AND of the LHS and RHS and assigns the resultant to the variable on the left
bitwise exclusive OR - assignment operator	^=	It takes the bitwise XOR of the LHS and RHS and assigns the resultant to the variable on the left
bitwise inclusive OR - assignment operator	|=	It takes the bitwise OR of the LHS and RHS and assigns the resultant to the variable on the left

Misc Operators

In addition to the above mentioned, there are several other operators, which are supported by Java.

Conditional Operator (? :):

The conditional operator is a ternary operator that contains three operands. Essentially, this operator is used for the evaluation of boolean expressions. The operator tests the first operand or condition and if the condition is true, then the second value is assigned to the variable. However, if the condition is false, the third operand is assigned to the variable. The syntax of this operator is as follows:

variable a = (<condition>) ? valueiftrue : valueiffalse

Sample implementation:

public class myTest {

public static void main(String args[]){

```
int x, y;

x = 5;

y = (x == 5) ? 15: 40;

System.out.println( “y = ” +  y );

y = (x == 34) ? 60: 95;

System.out.println( “x = ” + y );

}

}
```

The compilation and execution of this code shall give the following result:

```
y = 15

y = 95
```

instanceof Operator:

Only object reference variables can be used with this operator. The objective of this operator is to check is an object is an instance of an exiting class. The syntax of this operator is as follows:

(<object reference variable>) instanceof(<interface/class>)

Sample implementation of this operator and its purpose of use is given below:

```
public class myTest {

public static void main(String args[]){

int x = 4;

boolean resultant = x instanceof int;

System.out.println( resultant );

}

}
```

The output of this code shall be true. This operator can also be used in comparison. A sample implementation of this is given below:

```
class Animal {}
```

```
public class Monkey extends Animal {

public static void main(String args[]){

Animal newa = new Monkey();

boolean resultant =  newa instanceof Monkey;

System.out.println( resultant );

}

}
```

The output for this code will also be true.

Precedence of Java Operators

More often than not, operators are used in combinations in expressions. However, you must have also realized that it becomes difficult to predict the order in which operations will take place during execution. The operator precedence table for Java shall help you predict operator operations in an expression deduction. For instance, if you are performing addition and multiplication in the same expression, then multiplication takes place prior to addition. The following table illustrates the order and hierarchy of operators in Java. The associativity for all the operators is left to right. However, the unary, assignment and conditional operator follows right to left associativity.

Operator	**Category**
() [] . (dot operator)	Postfix
++ - - ! ~	Unary
* / %	Multiplicative
+ -	Additive
>> >>> <<	Shift
> >= < <=	Relational

== !=	Equality
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
&&	Logical AND
||	Logical OR
?:	Conditional
= += -= *= /= %= >>= <<= &= ^= |=	Assignment
,	Comma

Loops in Java

Looping is a common programming situation that you can expect to encounter rather regularly. Loop can simply be described as a situation in which you may need to execute the same block of code over and over. Java supports three looping constructs, which are as follows:

- for Loop
- do…while Loop
- while Loop

In addition to this, the foreach looping construct also exists. However, this construct will be explained in the chapter on arrays.

The while Loop:

A while loop is a control structure that permits you to rehash an errand a specific number of times. The syntax for this construct is as follows:

while(boolean_expression) {

/Statements

}

At the point when executing, if the boolean_expression result is genuine, then the activities inside the circle will be executed. This will proceed till the time the result for the condition is genuine. Here, key purpose of the while loop is that the circle may not ever run. At the point when the interpretation is tried and the result is false, the body of the loop will be skipped and the first proclamation after the while circle will be executed.

Sample:

public class myTest {

public static void main(string args[]) {

int i=5;

while(i<10) {

System.out.print(” i = ” + i);

i++;

```
System.out.print(“\n”);
}
}
```

This would deliver the accompanying result:

```
x = 5
x = 6
x = 7
x = 8
x = 9
x = 5
```

The do…while Loop

A do…while loop is similar to the while looping construct aside from that a do…while circle is ensured to execute no less than one time. The syntax for this looping construct is as follows:

```
do {
/Statements
}while(<booleanexpression>);
```

Perceive that the Boolean declaration shows up toward the end of the circle, so the code execute once before the Boolean is tried. In the event that the Boolean declaration is genuine, the stream of control bounced go down to do, and the code execute once more. This methodology rehashes until the Boolean articulation is false.

Sample implementation:

```
public class myTest {
public static void main(string args[]){
int i = 1;
do{
System.out.print(“i = ” + i );
i++;  System.out.print(“\n”);
```

}while(i<1);

}

This would create the accompanying result:

i = 1

The for Loop

A for circle is a reiteration control structure that permits you to effectively compose a loop that needs to execute a particular number of times. A for looping construct is helpful when you know how often an errand is to be rehashed. The syntax for the looping construct is as follows:

The punctuation of a for circle is:

for(initialization; Boolean_expression; redesign)

{

/Statements

}

Here is the stream of control in a for circle:

- The introduction step is executed in the first place, and just once. This step permits you to pronounce and introduce any loop control variables. You are not needed to put an announcement here, the length of a semicolon shows up.
- Next, the Boolean outflow is assessed. In the event that it is genuine, the assemblage of the loop is executed. In the event that it is false, the assortment of the loop does not execute and stream of control hops to the following articulation past the for circle.
- After the group of the for circle executes, the stream of control bounced down to the overhaul explanation. This announcement permits you to overhaul any circle control variables. This announcement can be left clear, the length of a semicolon shows up after the Boolean declaration.
- The Boolean outflow is currently assessed once more. On the off chance that it is genuine, the loop executes and the scope rehashes itself. After the Boolean declaration is false, the for loop ends.

Sample Implementation

```
public class myTest {

public static void main(string args[]) {

for(int i = 0; i < 5; i = i+1) {

System.out.print(“i = ” + i );

System.out.print(“\n”);

}

}
```

This would deliver the accompanying result:

```
i = 0

i = 1

i = 2

i = 3

i = 4
```

Extended Version of for Loop in Java

As of Java 5, the upgraded for loop was presented. This is basically utilized for Arrays. The syntax for this loop is as follows:

```
for(declaration : statement)

{

//Statements

}
```

- Declaration: The recently declared variable, which is of a sort perfect with the components of the show you are getting to. The variable will be accessible inside the for piece and its esteem would be the same as the current array component.
- Expression: This assesses to the exhibit you have to loop through. The interpretation can be an array variable or function call that returns an array.

Sample Implementation:

```
public class myTest {

public static void main(string args[]){
```

```
int [] mynumber = {0, 5, 10, 15, 20};

for(int i : mynumber ){

System.out.print( i );

System.out.print(” ,”);

}

}
```

This would deliver the accompanying result:

0, 5, 10, 15, 20

The break Keyword

The break keyword is utilized to stop the whole loop execution. The break word must be utilized inside any loop or a switch construct. The break keyword will stop the execution of the deepest circle and begin executing the following line of code after the ending curly bracket. The syntax for using this keyword is as follows:

```
break;
```

Sample Implementation:

```
public class myTest {

public static void main(string args[]) {

int [] mynumbers = {0, 5, 10, 15, 20};

for(int i : mynumbers ) {

if( i == 15 ) {

break;

}

System.out.print( i );

System.out.print(“\n”);

}

}
```

This would deliver the accompanying result:

0

5

10

The Continue Keyword

The proceed with decisive word can be utilized as a part of any of the loop control structures. It causes the loop to quickly bounce to the following emphasis of the loop.

- In a for circle, the continue keyword reasons stream of control to quickly bounce to the overhaul articulation.
- In a while or do/while loop, stream of control instantly hops to the Boolean interpretation.

The syntax of using this keyword is as follows:

continue;

Sample Implementation:

```
public class myTest {

public static void main(String args[]) {

int [] mynumbers = {0, 5, 10, 15, 20};

for(int i : mynumbers ) {

if( i == 15 ) {

continue;

}

System.out.print( i );

System.out.print("\n");

}

}

}
```

The expected output of the code is:

0

5

10

20

Decision Making

There are two sorts of decision making constructs in Java. They are:

- if constructs
- switch constructs

The if Statement:

An if constructs comprises of a Boolean outflow emulated by one or more proclamations. The syntax for using this construct is as follows:

if(<condition>) {

//Statements if the condition is true

}

In the event that the Boolean construct assesses to true, then the scope of code inside the if proclamation will be executed. If not the first set of code after the end of the if construct (after the end wavy prop) will be executed.

Sample Implementation:

public class myTest {

public static void main(string args[]){

int i = 0;

if(i < 1){

System.out.print(“The if construct is executing!”);

}

}

This would create the accompanying result:

The if construct is executing!

The if…else Statement

An if proclamation can be trailed by a non-compulsory else explanation, which executes when the Boolean outflow is false. The syntax for this construct is as follows:

if(<condition>){

```
//Executes if condition is true
}
else{
//Executes if condition is false
}
```

Sample Implementation:

```
public class myTest {
public static void main(string args[]){
int i = 0;
if( i > 1 ){
System.out.print(“The if construct is executing!”);
}
else{
System.out.print(“The else construct is executing!”);
}
}
```

This would create the accompanying result:

The else construct is executing!

The if…else if Statement

An if proclamation can be trailed by a non-compulsory else if…else explanation, which is exceptionally helpful to test different conditions utilizing single if…else if articulation.

At the point when utilizing if , else if , else proclamations there are few focuses to remember.

- An if can have zero or one else’s and it must come after any else if’s.
- An if can have zero to numerous else if’s and they must precede the else.
- If one of the if conditions yield a true, the other else ifs and else are ignored.

The syntax for using this decision making construct Is as follows:

```
if(condition_1){
//Execute if condition_1 is true
}
else if(condition_2){
//Execute if condition_2 is true
}
else if(condition_3){
//Execute if condition_3 is true
}
else
{
//Execute if all conditions are false
}
```

Sample Implementation:

```
public class myTest {
public static void main(string args[]){
int i = 0;
if( i > 1 ){
System.out.print(“The first if construct is executing!”);
}
else if(i == 0){
System.out.print(“The second if construct is executing!”);
}
else{
System.out.print(“The else construct is executing!”);
}
}
```

This would create the accompanying result:

The second if construct is executing!

Nested if…else Statement

It is legitimate to home if-else constructs, which implies you can utilize one if or else if proclamation inside an alternate if or else if explanation. The syntax for using this construct Is as follows:

```
if(condition_1){
//Execute if condition_1 is true
        if(condition_2){
        //Execute if condition_2 is true
        }
}
else if(condition_3){
//Execute if condition_3 is true
}
else
{
//Execute if all conditions are false
}
```

Sample Implementation:

```
public class myTest {
public static void main(string args[]){
int i = 1;
if( i >= 1 ){
System.out.println(“The if construct is executing!”);
        if(i == 1){
        System.out.println(“The nested if construct is executing!”);
```

```
        }
}
else{
System.out.print("The else construct is executing!");
}
}
```

This would create the accompanying result:

The if construct is executing!

The nested if construct is executing!

The switch Statement

A switch construct permits a variable to be tried for equity against a rundown of values. Each one value is known as a case, and the variable being exchanged on is checked for each one case. The syntax for using this decision making construct is as follows:

```
switch(<condition>){
case value1:
//Statements
break;
case value2 :
//Statements
break;
default:
//Optional
}
```

The accompanying runs apply to a switch construct:

- The variable utilized as a part of a switch explanation must be a short, byte, char or int.
- You can have any number of case explanations inside a switch. Each one case is trailed by the value to be contrasted with and a colon.

- The value for a case must be the same type as the variable in the switch and it must be a steady or an exacting value.
- When the variable being exchanged on is equivalent to a case, the announcements after that case will execute until a break is arrived at.
- When a break is arrived at, the switch ends, and the stream of control bounces to the following line after the switch.
- Not each case needs to contain a break. In the event that no break shows up, the stream of control will fall through to consequent cases until a break is arrived at.
- A switch articulation can have a discretionary default case, which must show up toward the end of the switch. The default case can be utilized for performing an undertaking when none of the cases is true. No break is required in the default case. However, as per the convention, the use of the same is recommended.

Sample Implementation:

```
public class myTest {

public static void main(string args[]){

char mygrade = ‘A’;

switch(mygrade)

{

case “A” :

System.out.println(“Excellent Performance!”);

break;

case “B” :

System.out.println(“Good Performance!”);

break;

default :

System.out.println(“Failed”);

}
```

Aggregate and run above code utilizing different inputs to grade. This would create the accompanying result for the present value of mygrade:

Excellent Performance!

Strings in Java

Strings, which are generally utilized as a part of Java, for writing computer programs, are a grouping of characters. In the Java programming language, strings are like everything else, objects. The Java platform provides the String class to make and control strings.

Instantiating Strings

The most appropriate approach to make a string is to use the following statement:

String mystring = “Hi world!”;

At whatever point it experiences a string exacting in your code, the compiler makes a String object with its value for this situation, “Hi world!’.

Similarly as with other objects, you can make Strings by utilizing a constructor and a new keyword. The String class has eleven constructors that permit you to give the starting estimation of the string utilizing diverse sources, for example, a cluster of characters.

public class myStringdemo{

public static void main(string args[]){

char[] myarray = { ‘h’, ‘i’, ‘.’};

String mystring = new String(myarray);

System.out.println(mystring);

}

This would deliver the accompanying result:

hi.

Note: The String class is changeless, so that once it is made a String object, its type can’t be changed. In the event that there is a need to make a great deal of alterations to Strings of characters, then you ought to utilize String Buffer & String Builder Classes.

Determining String Length

Routines used to get data about an object are known as accessor methods. One accessor technique that you can use with strings is the length() function, which furnishes a proportional payback of characters contained in the string item. This function can be utilized in the following manner:

```
public class mystring {

public static void main(string args[]) {

String newstr = “I am hungry!”;

int strlen = newstr.length();

System.out.println( “length = ” + strlen );  }
```

This would create the accompanying result:

```
length = 12
```

How to Concatenate Strings

The String class incorporates a function for connecting two strings:

```
mystring1.concat(mystring2);
```

This returns another string that is mystring1 with mystring2 added to it toward the end. You can likewise utilize the concat() system with string literals, as in:

```
“My name is “.concat(“mary”);
```

Strings are all the more usually concatenated with the + administrator, as in:

```
“Hi,” + ” world” + “!”
```

which brings about:

```
“Hi, world!”
```

Sample Implementation:

```
public class MyString {

public static void main(string args[]) {

String mystr = “Sorry”;

System.out.println(“I ” + “am ” + mystr);

}
```

This would deliver the accompanying result:

```
I am Sorry
```

Format Strings

You have format() and printf() functions to print the output with designed numbers. The

function format() of the String class returns a String object as against a Printstream object. This function creates a formatted string that can be reused. This function can be used in the following manner:

String fstr;

fstr = String.format(“Float variable value ” + “%f, and Integer value ” + “variable is %d, and the contents of the string is ” + ” %s”, fVar, iVar, sVar);

System.out.println(fstr);

String Methods

This section contains a list of methods that are available as part of the String class.

int compareTo(Object obj) – This function compares the specified string with the object concerned.

char charAt(int chindex) – This function returns the char present at the index value ‘index.’

int compareToIgnoreCase(String mystr) – This function performs the lexographic comparison of the two strings. However, the case differences are ignored by this function.

int compareTo(String aString) – This function performs the lexographic comparison between the strings.

boolean contentEquals(StringBuffer strb) – This function checks if the string is same as the sequence of characters present in the StringBuffer. It returns true on success and false on failure.

String concat(String strnext) – This function appends the string with another string at the end.

static String copyValueOf(char[] mydata, int xoffset, int xcount) – This function returns a stringf, which is indicative of the character sequence in the original string.

static String copyValueOf(char[] newdata) – This function copies the string of characters into a character buffer in the form of a sequence of characters.

boolean equals(Object aObject) – This function compares the object with the string concerned.

boolean endsWith(String newsuffix) – This function appends the string with the specified suffix.

byte getBytes() – Using this function, the string can be encoded into bytes format, which are stored in a resultant array.

boolean equalsIgnoreCase(String aString) – This function makes a comparison of the two strings without taking the case of characters into consideration.

void getChars(int srcBegin, int sourceEnd, char[] dst, int destinationBegin) – This function copies characters from the specified beginning character to the end character into an array.

byte[] getBytes(String charsetnm) - Using this function, the string can be encoded into bytes format using named char set, which are stored in a resultant array.

int indexOf(int charx) – This function returns the index of first character that is same as the character specified in the function call.

int hashCode() – A hash code is returned by this string.

int indexOf(String newstr) - This function returns the index of the first occurrence of a substring in a string.

int indexOf(int charx, int fromIndexloc) – This function returns the index of first character that is same as the character specified in the function call. The search starts from the specified index.

String intern() – A canonical representation of a string object given in the function call is returned.

int indexOf(String newstr, int fromIndexloc) - This function returns the index of the first occurrence of a substring in a string. The search starts from the specified index.

int lastIndexOf(int charx, int fromIndexloc) - This function makes a search for a character from the specified index and returns the index where the last occurrence is found.

int lastIndexOf(int charx) - This function makes a search for a character backwards and returns the index where the last occurrence or first occurrence in a backward search is found.

int lastIndexOf(String newstr, int fromIndexloc) – This function makes a search for a sub-string from the specified index and returns the index where the last occurrence is found.

int lastIndexOf(String newstr) - This function makes a search for a sub-string backwards and returns the index where the last occurrence or first occurrence in a backward search is found.

boolean matches(String aregex) - – This function checks for equality between a string region and a regular expression.

int length() – This function calculates and returns the string length.

boolean regionMatches(int totaloffset, String otherstr, int otheroffset, int strlen) – This function checks for equality between string regions.

boolean regionMatches(boolean ignorecharcase, int totaloffset, String otherstr, int otheroffset, int strlen) – This function checks for equality between string regions.

String replace(char oldCharx, char newCharx) - This function looks for a substring that matches the regular expression and then replaces all the occurrences with the specified string. The function returns the resultant string, which is obtained after making all the replacements.

String replaceFirst(String newregex, String newreplacement) – This function looks for a substring that matches the regular expression and then replaces the first occurrence with the specified string.

String replaceAll(String newregex, String xreplacement) - This function looks for a substring that matches the regular expression and then replaces all the occurrences with the specified string.

String[] split(String newregex, int xlimit) – This function performs splitting of the string according to the regular expression with it and the given limit.

String[] split(String newregex) - This function performs splitting of the string according to the regular expression with it.

boolean startsWith(String newprefix, int totaloffset) – This function checks if the given string has the prefix at the specified index.

boolean startsWith(String newprefix) – This function checks if the given string begins with the prefix sent with the function call.

String substring(int beginIndexloc) - This function returns a string, which is substring of the specified string. The substring is determined by the beginning index to the end of the string.

CharSequence subSequence(int beginIndexloc, int endIndexloc) - This function returns a character sequence, which is sub-character sequence of the specified character sequence. The substring is determined by the beginning and ending indexes.

char[] toCharArray() – This function performs the conversion of a string to a character array.

String substring(int beginIndexloc, int endIndexloc) – This function returns a string, which is substring of the specified string. The substring is determined by the beginning and ending index.

String toLowerCase(Locale localenew) - This function converts all the characters in the specified string to lower case using given locale rules.

String toLowerCase() - This function converts all the characters in the specified string to lower case using default locale rules.

String toUpperCase() - This function converts all the characters in the specified string to upper case using default locale rules.

String toString() – This function returns the string itself.

String toUpperCase(Locale localenew) – This function converts all the characters in the specified string to upper case using locale rules.

static String valueOf(primitive data type x) – A string representation is returned by this function.

String trim() – Omits the whitespace that trails and leads a string.

Arrays

Java supports an information structure, which is similar to a cluster. This information structure is called an array. It is capable of storing an altered size successive accumulation of components of the same data type. An array is utilized to store an accumulation of information, yet it is frequently more valuable to think about it as an exhibit for storing variables of the same sort.

As opposed to making declarations of individual variables, for example, num0, num1 and num99, you can declare one array variable. For example, an array of four elements is declared as arrayname[4]. This chapter discusses all the facets of array declaration, access and manipulation.

How To Declare array Variables

To utilize an array as a part of a system, you must declare a variable to reference the array. Besides this, you must determine the sort of array the variable can reference. Here is the syntax for declaring a variable of the type array:

datatype[] myarray;

Sample Implementation:

The accompanying code bits are illustrations of this concept:

double[] myarray;

Making Arrays

You can make an exhibit by utilizing the new operator with the accompanying statement:

myarray = new datatype[sizeofarray];

The above declaration does two things:

- It makes an exhibit with the help of the new operator in the following manner:

  new datatype[arraysize];

- It relegates the reference of the recently made array to the variable myarray.

Proclaiming a array variable, making an exhibit, and doling out the reference of the show to the variable can be consolidated in one declaration, as appeared:

datatype[] myarray = new datatype[sizeofarray];

On the other hand, you can also make clusters in the following manner:

datatype[] myarray = {val0, val1, …, valk};

The components of the array are gotten to through the record. Array lists are 0-based; that is, they begin from 0 to go up to myarray.length-1.

Sample Implementation:

The declaration shown below declares an array, myarray, makes a cluster of 10 components of double type and doles out its reference to myarray:

double[] myarray = new double[10];

Handling Arrays

At the point when handling components of an array, we frequently utilize either for or foreach in light of the fact that the majority of the components in an array are of the same sort and the extent of the exhibit is known.

Example:

```
public class Mytestarray {

public static void main(string[] args) {

double[] myarray = {0.5, 1.2, 2.2, 3.4, 4.7};

for (int k = 0; k < myarray.length; k++) {

System.out.println(myarray[k] + ” “);

}

double aggregate = 0;

for (int k = 0; k < myarray.length; k++) {

aggregate += myarray[k];

}

System.out.println(“Aggregate value = ” + aggregate);

double maxval = myarray[0];

for (int k = 1; k < mylist.length; k++) {

if (myarray[i] > maxval)

maxval = myarray[k];
```

}

System.out.println(“Max Value is ” + maxval);

}

This would create the accompanying result:

0.5 1.2 2.2 3.4 4.7

Aggregate = 12.0

Max Value is 4.7

The foreach Loops

JDK 1.5 presented another for construct, which is known as foreach loop or extended for loop. This construct empowers you to cross the complete array successively without utilizing an extra variable.

Sample Implementation:

public class Mytestarray {

public static void main(string[] args) {

double[] myarray = {0.5, 1.2, 2.2, 3.4, 4.7};

for (double i: myarray) {

System.out.println(i);

}

}

This would deliver the accompanying result:

0.5 1.2 2.2 3.4 4.7

Passing Arrays to Methods:

Generally, just as you can pass primitive values to methods or functions, you can likewise pass arrays to systems. Case in point, the accompanying method shows the components in an int array:

public static void printarr(int[] arr) {

for (int k = 0; k < arr.length; k++) {

System.out.print(arr[k] + ” “);

}

You can summon it by passing an array. Case in point, the accompanying declaration conjures the printarr function to show the elements of the array.

printarr(new int[]{0, 3, 5, 3, 1});

The compilation and execution of this code yields the following result:

0 3 5 3 1

How Can A Method Return An Array

A system might likewise give back an array. Case in point, the method demonstrated underneath returns an array that is the inversion of an alternate array:

public static int[] revarr(int[] myarr) {

int[] resultarr = new int[myarr.length];

for (int k = 0, i = resultarr.length - 1; k <= myarr.length/2; k++, i- -) {

resultarr[j] = myarr[k];

}

return resultarr;

}

The Arrays Class

The java.util.arrays class contains different functions for sorting and seeking values from array, looking at arrays, and filling components into arrays. These functions are available for all primitive data types.

- public static boolean equals(long[] a, long[] a2) - returns true if the two indicated arrays are equivalent to each other. Two arrays are viewed as equivalent if both of them contain the same number of components, and all relating sets of components in the two arrays are equivalent. This returns true if the two shows are equivalent. Same function could be utilized by all other primitive data types.
- public static int binarysearch(object[] an, Object key) - looks the pointed out array of Object for the defined value utilizing the double calculation. The array must be sorted before making this call. This returns list of the keys, in the event that it is contained in the list; generally, (-(insertion point + 1).
- public static void sort(Object[] a) – This function can be used to sort a given array

in the ascending order. It can likewise be used for any data type.

- public static void fill(int[] an, int val) - appoints the detailed int value to every component of the pointed out array of ints. Same function could be utilized for arrays of other data types as well.

Regular Expressions

Java includes the java.util.regex package to match with regular expressions. Java's normal outflows are fundamentally the same to the Perl programming language and simple to learn. A consistent outflow is an exceptional succession of characters that helps you match or discover different strings or sets of strings, utilizing a specific syntax held as a part of an example. They can be utilized to find, alter, or control content and information. The java.util.regex package essentially comprises of the accompanying three classes:

- Pattern Class: A Pattern article is an arranged representation of a consistent declaration. The Pattern class does not have any public constructors. To make an example, you should first conjure one of its public static methods, which will then give back a Pattern object. These functions acknowledge a normal statement as the first contention.
- Matcher Class: A Matcher article is the motor that translates the example and performs match operations against an information string. Like the Pattern class, Matcher has no public constructors. You get a Matcher object by conjuring the matcher method on a Pattern object.
- Patternsyntaxexception: A Patternsyntaxexception object is an unchecked exemption that shows a sentence structure mistake in a consistent statement design.

Catching Groups

Catching groups are an approach to treat various characters as an issue unit. They are made by putting the characters to be assembled inside a set of enclosures. Case in point, the normal declaration (canine) makes a solitary gathering containing the letters "d", "o", and "g". Catching gatherings are numbered by numbering their opening enclosures from left to right. In the representation ((A)(b(c))), for instance, there are four such gatherings:

- (a)
- (c)
- (b(c))
- ((a)(b(c)))

To discover what number of gatherings are available in the declaration, call the groupcount strategy on a matcher object. The groupcount technique gives back an int demonstrating the quantity of catching gatherings show in the matcher's example. There is likewise an uncommon gathering, gathering 0, which dependably speaks to the whole

outflow. This gathering is excluded in the aggregate reported by groupcount.

Sample Implementation:

This sample code emulates how to discover from the given alphanumeric string a digit string:

```
import java.util.regex.matcher;

import java.util.regex.pattern;

public class Myregexmatches {

public static void primary( String args[] ){

String line = “Request for Qt3000! “;

String example = “(.*)(\d+)(.*)”;

Pattern myr = Pattern.compile(pattern);

Matcher mym = myr.matcher(line);

if (mym.find( )) {

System.out.println(“Value = ” + mym.group(0) );

System.out.println(“Value = ” + mym.group(1) );

System.out.println(“Value = ” + mym.group(2) );

}

else {

System.out.print(“No match found!”);

}

}
```

Regular Expression Syntax

Given below is a list of regular expression syntax for your reference.

Matches	**Subexpression**
Matches line beginning	^
Matches line end	$

Matches single characters except for the newline character	.
Matches single character in braces	[…]
Matches single character, which are not in braces	[^…]
String beginning	\A
String end	\z
String end except for the final line terminating character	\Z
Matches 0 or more instances of expression	re*
Matches 1 or more instances of the expression	re+
Matches 0 or 1 instances of expression.	re?
Matches exactly n of instances of expression.	re{ n}
Matches n or more instances of the specified expression.	re{ n,}
Matches minimum n and maximum m instances of the expression.	re{ n, m}
Matches one of these: a or b.	a| b
Groups regular expressions. The matching text is remembered.	(re)
Groups regular expressions. The text is not remembered.	(?: re)
Matches independent pattern. No backtracking is supported.	(?> re)

Matches characters in a word.	\w
Matches characters, which are non-word.	\W
Matches whitespace. These characters are equivalent to [\t\n\r\f].	\s
Matches space, which is non-whitespace.	\S
Matches digits. These are typically equal to 0 to 9.	\d
Matches non-digits.	\D
Matches string beginning.	\A
Matches string end just before the newline character appears.	\Z
Matches string end.	\z
Matches the point where the last matching condition was found.	\G
Group number n back-reference	\n
Matches boundaries of the word when used without brackets. However, backspace is matched when it is used inside brackets.	\b
Matches boundaries, which are non-word	\B
Matches carriage returns, newlines, and tabs	\n, \t, etc.
Escape all the characters until a \E is found	\Q
Ends any quotes that begin with \Q	\E

Methods of the Matcher Class

Index Methods:

The following table gives a list of methods hat show correctly where the match was found in the info string:

- public int start(int bunch)

  Furnishes a proportional payback record of the subsequent caught by the given group amid the past match operation.

- public int begin()

  Furnishes a proportional payback record of the past match.

- public int end(int bunch)

  Furnishes a proportional payback after the last character of the subsequent caught by the given group amid the past match operation.

- public int end()

  Furnishes a proportional payback after the last character matched.

Study Methods:

Study methods survey the info string and return a Boolean demonstrating whether the example is found:

- public boolean find()

  Endeavors to discover the following subsequence of the info arrangement that matches the example.

- public boolean lookingat()

  Endeavors to match the info arrangement, beginning toward the start of the district, against the example.

- public boolean matches()

  Endeavors to match the whole district against the example.

- public boolean find(int begin)

  Resets this matcher and after that endeavors to discover the following subsequence of the information grouping that matches the example, beginning at the detailed list.

Substitution Methods:

Substitution methods are valuable methods for supplanting content in a data string:

- public static String quotereplacement(string mystr)

  Gives back an exacting substitution String for the tagged String. This method creates a String that will function as an issue substitution s in the appendreplacement system for the Matcher class.

- public Stringbuffer appendtail(stringbuffer strbuff)

  Actualizes a terminal annex and-supplant step.

- public Matcher appendreplacement(stringbuffer strbuff, String strsubstitution)

  Actualizes a non-terminal annex and-supplant step.

- public String replacefirst(string strsubstitution)

  Replaces the first subsequence of the data succession that matches the example with the given substitution string.

- public String replaceall(string strsubstitution)

  Replaces each subsequence of the data succession that matches the example with the given substitution string.

The begin and end Methods:

Taking after is the sample that tallies the quantity of times the statement “felines” shows up in the data string:

```
import java.util.regex.pattern;

import java.util.regex.matcher;

public class Regexmatches {

private static last String INPUT =  “feline cattie feline”;

private static last String REGEX = “\bcat\b”;

public static void principle( String args[] ){

Pattern myp = Pattern.compile(regex);

Matcher mym = myp.matcher(input);

int checkval = 0;

while(mym.find()) {
```

```
count++;
System.out.println(“match number “+count);
System.out.println(“start(): “+mym.start());
System.out.println(“end(): “+mym.end());
}
}
```

You can see that this sample uses word limits to guarantee that the letters “c” “a” “t” are not only a substring in a more extended word. It likewise provides for some helpful data about where in the information string the match has happened. The begin technique gives back where its due record of the subsequence caught by the given group amid the past match operation, and end furnishes a proportional payback of the last character matched, in addition to one.

The matches and lookingat Methods:

The matches and lookingat methods both endeavor to match an information succession against an example. The distinction, then again, is that matches requires the whole enter grouping to be matched, while lookingat does not. Both techniques dependably begin toward the start of the data string.

The replacefirst and replaceall Methods:

The replacefirst and replaceall routines supplant content that matches a given standard representation. As their names show, replacefirst replaces the first event, and replaceall replaces all events.

The appendreplacement and appendtail Methods:

The Matcher class additionally gives appendreplacement and appendtail routines to content substitution.

PatternSyntaxException Class Methods:

A PatternSyntaxException is an exception, which is unchecked. This exception indicates a syntactical error in the pattern of the regular expression. The PatternSyntaxException class offers the following methods to the developer for use.

- public int getIndex()

  This function returns the index of error.

- public String getDescription()

  This function returns the description of error.

- public String getMessage()

  This function returns the description and index of error.

- public String getPattern()

  This function returns the error-causing regular expression pattern.

Methods

A Java method is an accumulation of explanations that are gathered together to perform an operation. When you call the System.out.println function, for instance, the framework executes a few articulations so as to show a message on the output screen. Presently, you will figure out how to make your own routines with or without return qualities, call a method with or without parameters, over-loaded methods utilizing the same names, and apply method deliberation in the system plan.

How To Create Methods

Considering the accompanying sample to clarify the structure of a method:

```
public static int functionname(int x, int y) {

//Statements

}
```

Here, the method uses the following elements:

- Modifier: public static
- Data type of the return value: int
- Method name: functionname
- Formal Parameters: x, y

Such constructs are otherwise called Functions or Procedures. However, there is a distinctive quality of these two:

- Functions: They return an explicit value.
- Procedures: They don’t give back any quality.

Function definition comprises of a system header and body. The construct given above can be generalized to the following arrangement:

```
modifier returndatatype methodname (List of Parameter) { //Statements }
```

The structure indicated above incorporates:

- Modifier: It characterizes the right to gain entrance to the method and it is non-compulsory to utilize.
- Returntype: Method may give back a value of this data type.
- Methodname: This is the method name, which comprise of the name and the

parameter list of the method.

- List of Parameters: The rundown of parameters, which entails data type, request, and number of parameters of a method. A method may contain zero parameters as well.
- Statements: The method body characterizes what the method does with explanations.

Sample Implementation:

Here is the source code of the above characterized method called maxval(). This technique takes two parameters number1 and number2 and furnishes a proportional payback between the two:

```
public static int minval(int num1, int num2) {

int minvalue;

if (num1 > num2)  minvalue = num2;

else  minvalue = num1;

return minvalue;

}
```

Calling A Method

For utilizing a method, it ought to be called. There are two courses in which a technique is called i.e. technique gives back a value or nothing (no return value). The methodology of system calling is basic. At the point when a project summons a method, the system control gets exchanged to the called method. This called method then returns control to the guest in two conditions. These conditions include:

- Reaches the method closure brace.
- Return articulation is executed.

The methods returning void is considered as call to an announcement. Lets consider a sample:

```
System.out.println(“This is the end of the method!”);
```

The method returning a value can be seen by the accompanying illustration:

```
double resultant = sumval(4.2, 2.5);
```

Sample Implementation:

```
public class Minnumber{
public static void main(string[] args) {
double x = 21.5;
double y = 2.0;
double z = minvalfunc (x, y);
System.out.println(“The returned value = ” + z);
}
public static double minvalfunc (double num1, double num2) {
double minval;
if (num1 > num2)
minval = num2;
else
minval = num1;
return minval;
}
```

This would create the accompanying result:

```
The returned value = 2.0
```

The void Keyword:

The void keyword permits us to make methods, which don’t give back a value. Here, in the accompanying illustration we’re considering a void method. This function is a void method, which does not give back any value. Call to a void system must be an announcement i.e. rankpoints(657.3);. It is a Java explanation which closes with a semicolon as appeared.

Sample Implementation:

```
public class Myexample {
public static void main(string[] args) {
rankpoints(657.3);
}
```

```
public static void rankpoints(double valfoc) {
if (valfoc >= 100.5) {
System.out.println(“A1 Rank”);
}
else if (valfoc >= 55.4) {
System.out.println(“A2 Rank”);
}
else {
System.out.println(“A3 Rank”);
}
}
```

This would deliver the accompanying result:

A1 Rank

Passing Parameters by Value

While working under calling procedure, contentions is to be passed. These ought to be in the same request as their particular parameters in the function call. Parameters can be passed by reference or value. Passing parameters by value means calling a method with a parameter. Through, this is the contention value is gone to the parameter.

Sample Implementation:

The accompanying project demonstrates an illustration of passing parameter by value.

```
public class Myswapping {
public static void main(string[] args) {
double x = 0.43;
double y = 34.65;
System.out.println(“Values of X and Y before swapping, x = ” + x + ” and y = ” + y);
myswapfunc (x, y);
System.out.println(“\nValues of X and Y after swapping: “);
```

```
System.out.println(“x = ” +  x + ” and y is ” + y);

}

public static void myswapfunc (int x, int y) {

System.out.println(“Inside the function: Values of X and Y before swapping, x = ” + x  + ” y = ” + y);

System.out.println(“Inside the function: Values of X and Y after swapping, x = ” + x  + ” y = ” + y);

}
```

Function Overloading

At the point when a class has two or more methods by same name however diverse parameters, it is known as method overloading. It is not the same as overriding. In overriding a method has same name, number of parameters, data type and so on.

The underneath illustration clarifies the same:

```
public class Myoverloading{

public static void main(string[] args) {

int x = 50;

int y = 34;

double s = 14.3;

double r = 13.6;

int resultant1 = minfunc (x, y);

double resultant2 = minfunc (s, r);

System.out.println(“Value of minfunc = ” + resultant1);

System.out.println(“Value of minfunc = ” + resultant2);

}

public static int minfunc(int num1, int num2) {

int minval;

if (num1 > num2)

minval = num2;
```

```
else

minval = num1;

return minval;

}

public static double minfunc(double num1, double num2) {

double minval;

if (num1 > num2)

minval = num2;

else

minval = num1;

return minval;

}
```

Over-loading systems makes program clear. Here, two systems are given same name yet with distinctive parameters.

Utilizing Command-Line Arguments

Frequently you will need to pass data into a system when you run it. This is expert by passing order line contentions to principle( ). A summon line contention is the data that specifically takes after the program's name on the order line when it is executed. To get to the order line contentions inside a Java system is truly easy.they are put away as strings in the String cluster went to fundamental( ).

Sample Implementation:

```
public class Mycommandline {

public static void main(string args[]){

for(int k=0; k<args.length; k++){

System.out.println("ARGS[" + k + "]: " +   args[k]);

}

}
```

The Constructors:

A constructor instates an item when it is made. It has the same name as its class and is linguistically like a system. On the other hand, constructors have no return data type. Regularly, you will utilize a constructor to give beginning values to the instance variables characterized by the class, or to perform whatever other startup methods are needed to make a completely structured item.

All classes have constructors, whether you characterize one or not, on the grounds that Java naturally gives a default constructor that instates all variables to zero. Then again, once you characterize your own constructor, the default constructor is no more utilized.

Sample Implementation:

```
class Myconsclass {

int i;

Myconsclass() {

i = 0;

}
```

You would call constructor to introduce objects as shown below:

```
public class Myconsdemo {

public static void main(string args[]) {

Myconsclass tcons1 = new Myconsclass ();

Myconsclass tcons2 = new Myconsclass ();

System.out.println(tcons1.i + ” + tcons2.i);

}
```

Regularly, you will require a constructor that acknowledges one or more parameters. Parameters are added to a constructor in the same way that they are added to a strategy, simply declare them inside the brackets after the constructor’s name.

Sample Implementation:

```
class MyNewclass {

int x;

MyNewclass(int k ) {

x = k;
```

}

You would call constructor to introduce objects in the manner shown below:

public class Myconsdemo {

public static void main(string args[]) {

Myconsclass tcons1 = new Myclass(35);

Myconsclass tcons2 = new Myclass(67);

System.out.println(tcons1.x + ” “+ tcons2.x);

}

This would create the accompanying result:

35 67

Variable Arguments(var-args)

Java Development Kit 1.5 empowers you to pass arguments, which can be of variable number. However, the data type of the parameters should be the same. The parameter in the system is declared in the following manner:

typename… nameofparameter

In the statement, you define the data type emulated by an ellipsis (…). Only one variable-length parameter may be determined in a method, and this parameter must be the last parameter. Any customary parameters must go before it.

Sample Implementation:

public class Mysampleclass {

public static void main(string args[]) {

printmaxval(33, 45, 43, 22, 5);

printmaxval(new double[]{5, 8, 1});

}

public static void printmaxval(double… mynum) {

if (mynum.length == 0) {

System.out.println(“No Arguments!”);

return;

}

double resultant = mynum [0];

for (int x = 1; x < mynum.length; x++)

if (mynum[x] > resultant) resultant = mynum[x];

System.out.println(“Max val = ” + resultant); }

This would deliver the accompanying result:

Max val = 45.0

Max val = 1.0

The finalize( ) Method:

It is conceivable to call a method that will be called just before an object’s last annihilation. This method is referred to as finalize(), and it can be utilized to guarantee that an item ends neatly. For instance, you may utilize finalize( ) to verify that an open record possessed by that object is shut. To add a finalizer to a class, you basically jus call finalize(). The Java runtime calls that technique at whatever point it is going to reuse an object of that class.

Inside this function, you will point out those activities that must be performed before an object is removed. The syntax of using and implementing this function is:

protected void finalize( ) {

//Statements

}

The access modifier used for the method ensures that the method cannot be accessed by elements outside the particular class. This implies that you can’t know when or how the method executes.

File Handling

All the classes that you may require on a day to day I/O programming basis are contained in the package java.io. The streams present in this package broadly represent output and input locations. Moreover, the streams supported in Java include object, primitives and localized characters. A stream can simply be described as data, arranged in a sequence. While the inputStream can be used for inputting data from a source, the OutputStream can be sued for outputting data to a sink. The support for I/O provided by Java is flexible and extensive. This chapter sims to cover all the basic facets of File Handling in Java.

Byte Streams

Byte streams in Java are utilized to perform output and input of 8-bit bytes. In spite of the fact that there are numerous classes identified with byte streams yet most utilized classes are, Fileoutputstream and Fileinputstream. Here is an example of they can be used in real-life programming.

```
import java.io.*;

public class Filecopy {

public static void main(string args[]) throws IOException  {

FileInputStream inputx = invalid;

FileOutputStream outputx = invalid;

try {

inputx = new FileInputStream(“inputfile.txt”);

outputx = new FileOutputStream(“outputfile.txt”);

int charx;

while ((charx = inputx.read()) != -1) {

outputx.write(charx);

}

}

finally {

if (inputx != invalid) {
```

```
inputx.close();
}
if (outputx != invalid) {
outputx.close();
}
}
}
```

Presently we should have a record inputfile.txt with the accompanying content:

This is for testing purpose only.

As an important step, compile and execute the code shown above. The execution of the code shall result in the creation of outputfile.txt file.

Character Streams

Java Byte streams are utilized to perform output and input of 8-bit bytes. On the other hand, Java Character streams are utilized to perform output and input for 16-bit unicode. In spite of the fact that there are numerous classes identified with character streams yet the most commonly used ones include Filereader and Filewriter.

It is worth mentioning here that the implementation of Filereader utilizes Fileinputstream and Filewriter utilizes Fileoutputstream. This may make you wonder as to what is the difference between the former and latter. Filereader peruses two bytes at once and Filewriter composes two bytes at once. We can re-compose above sample which makes utilization of these two classes to duplicate an info record (having unicode characters) into an outputfile.txt.

```
import java.io.*;
public class Mycopyfile {
public static void main(string args[]) throws IOException  {
FileReader inputx = invalid;
FileWriter outputx = invalid;
try {
inputx = new FileReader(“inputfile.txt”);
```

```
outputx = new FileWriter(“outputfile.txt”);
int charx;
while ((charx = inputx.read()) != -1) {
outputx.write(charx);
}
}
finally {
if (inputx != invalid) {
inputx.close();
}
if (outputx != invalid) {
outputx.close();
}
}
}
```

Presently how about we have a record inputfile.txt with the accompanying text:

This is for testing purpose only.

Compile and execute the file containing this code. The execution of this code should create an output file outputfile.txt.

Standard Streams

All the programming languages give backing to standard I/O where client’s code can take information from a console and afterward deliver appropriate output on the machine screen. On the off chance that you have some knowledge of C or C++, then you must be mindful of three standard tools namely, STDIN, STDOUT and STDERR. Java provides three standard streams, which are discussed below:

- Standard Error: This is utilized to yield the error information created by the client’s code and normally a machine screen is utilized as standard error stream and referred to as System.err.
- Standard Output: This is utilized to yield the information created by the client’s

code and normally a machine screen is utilized to standard output stream and referred to as System.out.

- Standard Input: This is utilized to encourage the information to client's code and normally a console is utilized as standard data stream and referred to as System.in.

Sample Implementation:

```
import java.io.*;

public class Myreadconsole {

public static void main(string args[]) throws IOException  {

InputStreamReader cinx = invalid;

try {

cinx = new InputStreamReader(system.in);

System.out.println("Input string, press "e" to exit.");

char charx;

do {

charx = (char) cinx.read();

System.out.print(charx);

} while(charx != 'e');

}

finally {

if (cinx != invalid) {

cinx.close();

}

}

}
```

The code mentioned above must be saved in a file named Myreadconsole.java. Upon compilation and execution of this code, the system must be able to receive and interpret characters.

Perusing and Writing Files

As mentioned previously, a stream can be defined as a sequence of information. The Inputstream is utilized to peruse information from a source and the Outputstream is utilized for outputting information to a terminus.

Here is a chain of importance of classes to manage Input and Output streams. The two essential streams are Fileinputstream and Fileoutputstream, which would be talked about in the following section:

Fileinputstream:

This stream is utilized for perusing information from the documents. Objects can be made utilizing the keyword new and there are a few sorts of constructors accessible. Inputstream can be used for reading files in the following manner:

Inputstream myfx = new Fileinputstream(“c:/java/hi”);

The constructor takes a record item to make a data stream object to peruse the document. Initially, we make a record item utilizing File() technique in the following manner:

File myfx = new File(“c:/java/hi”);

Inputstream myfx = new Fileinputstream(myfx);

When you have the object of Inputstream under control, there is a rundown of assistant methods, which can be utilized to peruse to stream or to do different operations on the stream.

- protected void finalize() throws IOException {}

  This system cleans up any association with the file and guarantees that the local method for this output stream for the file is called. Besides this, this method is also capable of throwing an exception.

- public void close() throws IOException {}

  This system shuts the output stream of the file and discharges any framework assets connected with the the same. It is also capable of throwing an exception.

- public int available() throws IOException{}

  This function returns an int, indicating the number of bytes that the input stream can still read.

- public int read(int r)throws IOException{}

  The read method is used for reading content from the InputStream and returns the

next byte of data in int data type. However, upon reaching the end of file, it returns -1.

- public int read(byte[] r) throws IOException{}

  This read method is similar in operation to the read method described above with the exception that it reads data length of r in the given array. The function returns the number of bytes read and -1 upon reaching the end of file.

Other input streams are also available for use. Some of these include:

- DataInputStream
- ByteArrayInputStream

FileOutputStream:

Fileoutputstream is utilized to make a file and write text into it. The stream would create a file, in the event that it doesn't as of now exist, before opening it for outputting. Here are two constructors which can be utilized to make a Fileoutputstream object.

Method 1:

OutputStream myfx = new FileOutputStream("c:/java/hi")

Method 2:

File myfx = new File("c:/java/hi");

OutputStream myfx = new FileOutputStream(myfx);

When you have OutputStream object under control, there is a rundown of aide methods, which can be utilized to keep in touch with stream or to do different operations on the stream.

- public void write(int w) throws IOException {}

  This method composes the tagged byte to the output stream.

- protected void finalize() throws IOException {}

  This strategy cleans up any associations with the record. Besides this, it also guarantees that the local method for this output stream for file is called. This method is capable of throwing an exception.

- public void close() throws IOException {}

  This method shuts the output stream of the file. Moreover, it discharges any

framework assets connected with the document. This method also throws an IOException.

- public void write(byte[] w)

  This method composes w.length bytes from the specified byte exhibit to the Outputstream.

There are other imperative output streams accessible, which are as follows:

- ByteArrayOutputStream
- DataOutputStream

Sample Implementations:

```
import java.io.*;

public class Mytestfile{

public static void main(string args[]){

try{

byte bytewrite [] = {45,64,22,49,1};

OutputStream myos = new FileOutputStream(“mytest.txt”);

for(int i=0; i < bytewrite.length ; i++){

myos.write( bytewrite[x] );

}

myos.close();

InputStream myis = new FileInputStream(“mytest.txt”);

int sizex = myis.available();

for(int z=0; z< sizex; z++){

System.out.print((char)myis.read() + “  “);

}

myis.close();

}catch(IOException e){

System.out.print(“Exception Caught!”);
```

}

}

The above code would make a file mytest.txt and would compose given numbers in parallel organization. Same would be outputted to the stdout screen.

File Navigation and I/O

There are a few different classes that we would be experiencing to get to know the fundamentals of File Navigation and I/O.

- File Class
- FileWriter Class
- FileReader Class

Directories

A directory is a File, which can contains a rundown of different catalogs and files. You utilize the object File to make catalogs, to rundown down documents accessible in an index. For complete point of interest check a rundown of every last one of techniques which you can approach File item and what are identified with indexes.

Making Directories:

There are two valuable File utility methods, which can be utilized to make directories:

- The mkdirs() method makes both a directory and all the elements of the index.
- The mkdir( ) method makes a directory, returning valid on achievement and false on disappointment. Failure demonstrates that the way determined in the File object exists, or that the index can’t be made in light of the fact that the whole way does not exist yet.

Sample Implementation:

```
import java.io.File;

public class MyCreateDir {

public static void main(String args[]) {

String directoryname = “/tmp/user/java/bin”;

File dir = new File(directoryname);

dir.mkdirs();
```

}

}

Listing Directories:

You can utilize list( ) method provided by the File class to provide a list of all the records and directories accessible in an index.

Sample Implementation:

```
import java.io.File;

public class MyReadDir {

public static void main(String[] args) {

File myfile = null;

String[] paths;

try{

myfile = new File(“/tmp”);

mypaths = file.list();

for(String path:mypaths)

{

System.out.println(path);

}

}catch(Exception e){

e.printStackTrace();

}

}

}
```

Exception Handling

During the execution of your program, it may experience abnormal or exceptional conditions. As a result of these, the system may crash. An exception may occur due to a number of reasons. Some of these include:

- A file that needs to be opened can’t be found.
- A client has entered invalid information.
- A system association has been lost amidst correspondences or the JVM has used up all the available memory.

Some of these special cases are created by client mistake, others by developer blunder, and others by physical assets that have fizzled into your code in some way. To see how exception handling works in Java, you have to comprehend the three classifications of exceptions:

- Errors: These are not special cases whatsoever. Therefore, errors can be defined as issues that are beyond the understanding and the ability to control of the client or the software engineer. They are normally overlooked in your code on the grounds that you can once in a while take care of a mistake. Case in point, if a stack overflow happens, it is sure to result in an error. They are additionally disregarded at the time of compiling.
- Runtime Exceptions: It is a special case that most likely could have been dodged by the software engineer. Runtime exceptions are disregarded at the time of assemblage.
- Checked Exceptions: It is a special case that is regularly a client mistake or an issue that can’t be predicted by the developer. Case in point, if a file is to be opened, yet the file can’t be found, an exception of this type happens. These special cases can’t just be disregarded at the time of compilation and dry runs.

Hierarchy of Exceptions

All classes of exceptions are subtypes of the java.lang.exception class. This class is a subclass of the Throwable class. Other than the exception class, there is an alternate subclass called Error which is gotten from the Throwable class. These special case scenarios are not ordinarily caught by the Java programs. These conditions ordinarily happen if alternate scenarios are not taken care of by the java programs. Errors are produced to demonstrate lapses created by the runtime environment. A sample exception

is: Out of Memory or Stack Overflow. The Exception class has two primary subclasses: IOException and RuntimeException Classes.

Exception Methods:

Here is a list of methods that are available as part of the Throwable class.

- public Throwable getcause()

  This method gives back the cause of the exemption as mentioned by a Throwable item.

- public String getmessage()

  This method gives back the exception's complete message and details. This message is usually included in the Throwable constructor.

- public void printstacktrace()

  This method prints the aftereffect of tostring() alongside the stack follow to System.err, the output stream for error.

- public String tostring()

  The method gives back where its due of the class linked with the aftereffect of getmessage()

- public Throwable fillinstacktrace()

  The method fills the stack of this Throwable object with the current trace of stack, adding to any past data in the trace of stack.

- public Stacktraceelement [] getstacktrace()

  The method gives back a array containing every component on the trace of stack. The component at file 0 speaks to the highest point of the call stack, and the last component in the show speaks to the system at the base of the call stack.

Getting Exceptions:

A system discovers a special case utilizing a blend of the try and catch keywords. A try scope is set around the code that may produce an exemption. Code inside this scope is alluded to as secured code, and the structure for utilizing try/catch is given below:

try {

//Code that may produce an exception

}catch(nameofexception exp_1) {

//Code to be executed once an exception occurs

}

A try block includes announcing the kind of exception you are attempting to get. In the event that an exception happens in ensured code, the catch square that executes after the attempt is checked. In the event that this sort of special case that happened in a try block, the exception goes to the catch block, which is also passed as a system parameter.

Sample Implementation:

import java.io.*;

public class MyException{

public static void main(string args[]){

try{

int myarr[] = new int[2];

System.out.println("This statement attempts to access the third element of the array:" + a[3]);

}catch(arrayindexoutofboundsexception e_1){

System.out.println("The thrown exception is: " + e_1);

}

System.out.println("Exception: Out of Bounds");

}

This would deliver the accompanying result:

The thrown exception is: java.lang.arrayindexoutofboundsexception: 3

Exception: Out of Bounds

Using Multiple Try Blocks

A single piece of code can have a number of catch blocks for catching different exceptions. The structure of the multiple try/catch blocks is given below:

try {

//Statements to be tested

```
}catch(exceptiontype1 e_1) {
//Catch block 1
}
catch(exceptiontype2 e_2) {
//Catch block 2
}catch(exceptiontype3 e_3) {
//Catch block 3
}
```

This code uses three catches. However, you can use as many catch blocks as you need for your code. On the off chance that an exception happens in the protected code, the exemption is thrown and caught firstly by the first catch block. If the exception type matches, then the catch block executes. However, if the exception type doesn't match, the exception is open to be caught by the next catch block. This process continues until a matching exception type is found or all the catch blocks have been checked.

Sample Implementation:

```
try{
filex = new Fileinputstream(nameoffile);
num = (byte) filex.read();
}catch(IOException e_1) {
e_1.printstacktrace();
return -1;
}catch(filenotfoundexception f_1){
f_1.printstacktrace();
return -1;
}
```

Throws Keyword

On the off chance that a system does not handle a checked exception, the method must proclaim it utilizing the keyword throws. The throws keyword shows up toward the end of

a the method’s signature. You can throw an exemption, either a recently instantiated one or a special case that you simply found, by utilizing the keyword throw.

Finally Keyword

The keyword finally is utilized to make a piece of code that last code to be executed for a program. A finally square of code dependably executes, irrespective of whether an exemption has happened or not. Utilizing a finally piece permits you to run any cleanup-sort statements that you need to execute, regardless of what happens in the secured code.

Creating An Exception

You can make your own exemptions in Java. Remember the accompanying focuses when composing your classes for exceptions:

- All exemptions must be an offspring of Throwable.
- If you need to compose a checked exemption that is naturally authorized by the Handle or Declare Rule, you have to create an extension of the Exception class.
- If you need to compose a runtime exemption, you will have to create an extension of the Runtimeexception class.

You can create your own exceptions using the following structure:

class MyNewException extends Exception{ }

Common Exceptions

In Java, it is conceivable to characterize two categories of Exceptions and Errors.

- Programmatic exceptions: - These special cases are tossed unequivocally by the application or the API software engineers Examples: Illegalargumentexception, IllegalStateException.
- JVM Exceptions: - These are exemptions/mistakes that are solely or consistently thrown by the JVM. Some exceptions of this class are ArrayIndexOutOfBoundsException, NullPointerException and ClassCastException.

Interfaces and Packages

Abstract methods when brought together form a package. A class actualizes an interface, consequently inheriting the interface's abstract methods. An interface is not a class. Composing an interface is like composing a class. However, they are two separate ideas. A class portrays the properties and behaviours of an object. On the other hand, An interface contains behaviours that a class shall implement.

Unless the class that actualizes the interface is abstract, all the methods for the interface need to be implemented in the class. An interface is like a class in the accompanying ways:

- An interface is composed in a file with a .java augmentation, with the name of the interface matching the name of the file.
- An interface can contain any number of methods.
- Interfaces show up in packages, and their relating bytecode file must be in a directory structure that matches the name of the package.
- The bytecode of an interface shows up in a .class record.

On the other hand, an interface is unique and different from a class in a few ways. These are:

- An interface does not contain any constructors.
- Interface cannot be instantiated
- Instance fields cannot be contained in an interface. It is a requirement of interfaces that the main fields in them must be final and static.
- It is a requirement that all of the methods must be abstract methods.
- An interface can extend different interfaces.
- A class does not access an interface. Actually, a class implements an interface.

Declaring Interfaces

In order to declare an interface, you must use the interface keyword. Here is a straightforward illustration that can be used for declaring an interface. The standard structure and order of statements that can be used for this purpose are as follows:

```
import java.lang.*;

public interface Interfacename {

//Statements
```

}

Interfaces have the accompanying properties:

- An interface is verifiably dynamic. You don’t have to utilize the keyword abstract when declaring an interface.
- Each method in an interface is additionally dynamic, so the keyword abstract is not required.
- Methods in an interface are certainly public.

Sample Implementation

interface MyAnimal {

public void eatinghabits();

public void walkinghabits();

}

Packages

Packages are utilized as a part of Java so as to avert naming clashes, to control access, to make seeking/placing and utilization of classes, interfaces, identifications and annotations less demanding, in addition to several others. A Package can be described as a collection of related types (classes, interfaces, counts and annotations) giving access security and name space administration.

A few packages available in Java are::

- java.io – all the classes for output and input are available in this package
- java.lang – all the major classes are available in this package

Developers can create their own packages or package collections of classes/interfaces, and so on. It is a decent practice to collect related classes executed by you so that a software engineer can undoubtedly discover that the interfaces, classes, annotations and counts can be connected. Since the package makes another namespace, there won’t be any name clashes with names in different packages. Utilizing packages, it is less demanding to give access control and it is likewise simpler to find the related classes.

At the point when you create a package, you ought to pick a name and put a explanation with that name at the highest point of each source record that contains the classes, interfaces, lists, and annotation sorts that you need to incorporate in the package. The package declaration ought to be the first line in the source record. There can be one and

only declaration in each one source record, and it applies to various sorts in the file. In the event that a declaration of interface is not utilized, then the interfaces, class, annotation and specifications will be put in a package, which will be unnamed.

Java Applets

In order to run an applet, you must have a web browser. An applet can be a completely utilitarian Java application on the grounds that it has the whole Java API's available to it. There are some essential contrasts between an applet and a standalone Java application, including the accompanying:

- A main() function is not conjured on an applet, and an applet class won't define main().
- An applet is a Java class that extends and enhances the java.applet.applet class.
- When a client sees a HTML page that contains an applet, the code for the applet is automatically downloaded to the client's machine.
- Applets are intended to be inserted inside a HTML page.
- The JVM on the client's machine makes an instance of the applet class and conjures different routines amid the applet's lifetime.
- The security requirements for an applets are very strict. The security of an applet is frequently alluded to as sandbox security, contrasting the applet with a youngster playing in a sandbox with different decides that must be emulated.
- A JVM is a base requirement for viewing an applet. The JVM can be either a module of the Web program or a different runtime environment.
- Other classes that the applet needs can be downloaded in a solitary Java Archive (JAR) file.

Life Cycle of an Applet

The creation of any applet requires the implementation of four methods of the Applet class. These methods have been discussed in the text below.

- init: This method is planned for whatever introduction is required for your applet. It is called after the param labels inside the applet tag have been transformed.
- start: This method is naturally called after the program calls the init strategy. It is likewise called at whatever point the client comes back to the page containing the applet in the wake of having gone off to different pages.
- stop: This method is consequently called when the client leaves the page on which the applet sits. It can, accordingly, be called over and over in the same applet.
- destroy: This technique is just called when the program closes down. Since applets are intended to live on a HTML page, you ought not regularly desert assets after a

client leaves the page that contains the applet.

- paint: This method is invoked quickly after the start() method. Furthermore, whenever the applet needs to repaint itself in the program, this method needs to be called. The paint() method is inherited from the java.awt.

A "Welcome, World" Applet

The accompanying is a basic applet named BasicApplet.java:

```
import java.applet.*;

import java.awt.*;

public class BasicApplet extends Applet {

public void paint (Graphics gx)  {

gx.drawstring ("Say Hello To The World!", 35, 70);

}
```

These import explanations bring the classes into the extent of our applet class:

- java.awt.graphics
- java.applet.applet

Without those import explanations, the Java compiler would not perceive the classes Applet and Graphics, which the applet class alludes to.

The Applet Class

Each applet is an augmentation of the java.applet.applet class. The base Applet class gives techniques that a determined Applet class may call to get data and administrations from the program connection. These incorporate techniques that do the accompanying:

- Get parameters of the applet
- Get the system area of the HTML record that contains the applet
- Get the system area of the applet class registry
- Print a status message in the program
- Fetch a picture
- Fetch a sound
- Play a sound
- Perform resizing of the applet

Moreover, the Applet class gives an interface by which the viewer or program gets data

about the applet and controls the applet’s execution. The viewer might:

- Request data about the form, creator and copyright of the applet
- Request a depiction of the parameters the applet perceives
- Perform applet initialization
- Perform applet destruction
- Begin the execution of the applet
- Stop the execution of the applet

The Applet class allows default usage of each of these routines. Those executions may be overridden as essential. The “Say Hello To The World!” applet is complete in itself. However, as part of the implementation, only the paint() function is over-ridden.

A Simple Start to jQuery, JavaScript, and Html5 for Beginners

Written by a Software Engineer

By

Scott Sanderson

Table of Contents

Chapter 1: Introduction

Chapter 2: Basics of HTML5

Chapter 3: Basics of JavaScript

Chapter 4: Basics of CSS3

Chapter 5: HTML5 Explained

Chapter 6: JavaScript and jQuery

Chapter 7: Forms

Chapter 8: Web Services

Chapter 9: WebSocket Communications

Chapter 10: Managing Local Data With the Help of Web Storage

Chapter 11: Offline Web Applications

Appendix

Copyright 2015 by Globalized Healing, LLC - All rights reserved.

Click here to receive incredible ebooks absolutely free!

Chapter 1: Introduction

If there is one application development framework that can be termed as comprehensive, then HTML5 wins the bid hands down. Although, the specification for HTML5 is not yet complete, most modern web browsers support the popular features on device, which range from a Smartphone to a desktop. What that means is that you just have to write one application and it will run on your devices without any interoperability issues.

There is no doubt about the fact that HTML5 is the future of web application development and if you wish to remain in the league, you need to think futuristically and equip yourself to deal the technological challenges that the future is about to throw at you. The scope of HTML5 is evident from the fact that most of the major players of the industry and setting their eyes on this technology and giving in full support to the same.

If the multi-faceted functionality and high-on features characteristics of HTML5 intrigue you and you want to start writing your own applications right away, but you do not know how and where to begin, then this book is just for you. This book covers everything that you shall require to create working applications with the help of HTML, JavaScript/JQuery and CSS. However, it is not a reference guide. We hope to give you practical knowledge so that you can get to development as quickly as possible.

This book is a perfect start-up guide and covers all the basic facets of HTML5, JavaScript and CSS development. It covers everything from the very basics to all that you shall require in your tryst with this framework. The first three chapters introduce you to these three technologies, giving you some ground to start with.

Chapter 2: Basics of HTML5

HTML (Hyper Text Markup Language) is a language used for creating web pages. In fact, this language has been in use since the first webpage was made. However, the functionality has evolved as newer and better versions of the language were introduced. The language is known to have originated from SGML (Standard Generalized Markup Language), which was earlier used for document publishing. HTML has inherited the concept of formatting features and their syntax from SGML.

One of the most interesting and beneficial facet of HTML usage, as far as browsers are concerned, is that browsers support both backward as well as forward compatibility. While backward compatibility is usually easy to achieve, forward compatibility is tricky as the problem domain, in this case, is infinitely large. However, in order to implement this, browsers were designed to ignore tags that it did not recognize.

For years, HTML remained all that people wanted. However, with time, people felt the need for more, which was catalyzed by the presence of another technology called XML (eXtensible Markup Language). Although, XML shares a lot of similarities with HTML, there exist many fundamental differences between the two. Firstly, XML requires tag matching in the sense that for every starting tag, a closing tag must inevitably exist. Besides this, XML allow you to create your own tags as it does not possess a fixed set of tags like HTML.

The tags used in XML are meta-tags or tags that describe the data that is included between the starting and closing tag. In order to ensure the validity of the XML document, a technology called XSD (XML Schema Definition) is used. However, this technology cannot be used for validating HTML documents because HTML documents lack a well-defined structure.

The W3C (World Wide Web Consortium) introduced XHTML as an attempt to fix the

flaws of HTML. According to the XHTML specification, HTML documents were forced to adhere to the format specifications used for XML. Therefore, this allowed the use of XSD tools for validation of HTML documents. Although, the integration of XML in the framework fixed some issues, some issues continued to crop up. One of the staggering issues of the modern times was the growing need for integration of multimedia. While CSS did perform formatting of some level, it was becoming inadequate for the growing demands of users.

In order to provide support for interactivity and animated visuals, a programmable support called JavaScript was added to this ensemble. However, initial versions of this support were difficult for programmers to understand and slow on execution time incurred. This led to the introduction of plug-ins like Flash to get the attention that it did. These plugins did what was expected of them, but the browser-multimedia integration was still loose in nature.

HTML5 is not an evolved form of XHTML. On the contrary, HTML5 can be better described as the reinvented form of HTML 4.01 and how HTML, CSS and JavaScript can be used together to solve the growing needs of the user base.

Semantic Markup

The fundamental feature of HTML5 is that it stresses on separation of behaviour, presentation and structure. The semantic markup of a website development specifies the structure of the document. In other words, it specifies the meaning of tags and what they will do for you. On the other hand, behaviour and presentation are governed by CSS and JavaScript respectively.

HTML5 Elements

In HTML, an element is simply a statement that contains a beginning tag, content and a

closing tag. Therefore, when you write,

This is my world!

In this example, the div elements includes everything from to . therefore, the tag is also a part of the div element.

It is important to state here that HTML5 is not case sensitive. Therefore, regardless of whether you write <B> or for the bold tag, the browser will consider the two same. However, the use of lower case for tags is recommended by the W3C standards.

Working with Elements in HTML5

HTML5 defines more than a 100 elements. These elements, with their definitions are provided in Appendix.

How to add attributes to elements?

Additional data can be added to the begin tag in the form of attributes. An attribute can be generally represented as, name=”value”. The value for the attribute name is enclosed within quotes. There is no restriction on the number of attributes that can be added to the begin tag. For these attributes, the attribute has a name, but it does not have a value.

Example:

Here, id and class are attributes where id uniquely identifies the element and class specifies the CSS style to which this div belongs.

Boolean Attributes

Several types of attributes can be used. One of the most commonly used types is Boolean

attribute. The Boolean attribute can take a value from the allowable set of values. Some possible values include:

- Checked
- Disabled
- Selected
- Readonly

There are two ways to indicate the value of a Boolean attribute.

<input type=“checkbox” name=“vegetable” value=“Broccoli” checked=” />

<input type=“checkbox” name=“vegetable” value=“Broccoli” checked=‘checked’ />

In the first case, the value is not given and is assumed. Although, the latter seems like a redundant form, it is the more preferred form, particularly if you are using jQuery.

Global Attribute Reference

There is a set of named attributes available in HTML5, which can be used with any element. Examples of these attributes include accesskey, spellcheck and draggable, in addition to several others.

Void Elements

Some elements do not have to contain content under any circumstance. These elements include <link>, <br> and <area>, in addition to many others.

Self-closing Tags

If you are using an element that does not contain any content, then you can use a self closing tag. An example of this tag is <br/>. However, please note that other tags like have to be written as even if they do not have any content.

How to Add Expando Attributes

Any attribute that you as the author define is known as expando attribute. You can give this custom attribute a name and assign a value to the same as and when required.

How to Add Comments

Comments can be added using a ! and two hyphens (-). The syntax for adding comments is as follows:

<!—text —>

You can also add conditional comments using the following syntax:

<!—[if lte IE 7]> <html class=“no-js ie6” lang=“en”> <![endif]—>

This comment determines if the browser being used is an earlier version released earlier than IE7.

How to Create HTML Document

An HTML document can simply be described as a frame that contains metadata, content and a structure.

The HTML document must start with the tag <!DOCTYPE html>. In HTML5, the declaration is an indication to the browser that it should work in no-quirks mode or HTML5 compliant mode.

The next tag should be <html>. This element includes within itself the elements <head> and <body>. The <head> element can contain the metadata for the HTML document, which is declared and defined using the <meta> tag. The <head> element also contains the <title> element, which serves the following purposes:

- Used by search engines
- This content is displayed in the browser toolbar
- Gives a default name to the page.

The <body> element includes any other information that is to be displayed on the HTML document.

A sample HTML document is given below:

<!DOCTYPE html>

<html>

<head>

<meta charset=“utf-8” />

<title>TITLE OF DOC</title>

</head>

<body>

CONTENT

</body>

</html>

Note:

1. Special character like ‘>’ or ‘<’, which are a part of the HTML syntax, can be used on the HTML doc using their name or number. The syntax for their usage is:

   &entity_name or &entity_number

   The appendix contains the table of entity names and numbers to be used.

2. White space created using tabs, line breaks and spaces is normalized by HTML into a single space. Therefore, if you want to use multiple spaces, you must use the non-breaking space character. For example, if you want to display 10 mph such that 10 and mph are not separated by a newline, you can write 10 mph.

How to Embed Content

Now that you know how to create HTML documents with your content, you may want to embed content into the document from other sources. This content may be text from another HTML document or flash applications.

Inline Frames

Inline frames are used to embed HTML documents inline with the content of the current HTML document. Therefore, in a way, this element creates nested browsers as multiple web page are loaded in the same window. These are implemented using the <iframe> element. Nested browsing contexts can be navigated using:

- window.parent – This WindowProxy object represents the parent browsing context.
- window.top – This WindowProxy object represents the top-level browsing context
- window.frameElement – This represents the browsing context container and if there is no context container, it returns null.

The syntax of the <iframe> element is as follows:

<iframe src=“name”></iframe>

Here the name of the attribute defines the name of the browsing context. This name can be any string like ‘sample.html’. However, the strong should not start with an underscore as names starting with underscore are reserved for special key names like _blank.

Sandboxing

Sandboxing is used for avoiding the introduction of pop-ups or any other malware in your HTML document. In order to implement this, the attribute ‘sandbox’ is used. You can use this attribute in the following manner:

<iframe sandbox=”keywords” src=“name”>

</iframe>

When you use sandboxing, several restrictions are imposed on the called context. These restrictions include disabling forms, scripts, plugins and any frame other than itself. You can force the system to override these restrictions using keywords like:

- allow-same-origin
- allow-scripts
- allow-forms
- allow-top-navigation

Multiple keywords can be used by separating them using a space.

Seamless Embedding

The seamless attribute can be used in the <iframe> element for embedding of content in a manner than this content appears to be part of the main content. This attribute can be used in the following manner:

<iframe seamless=”” src=“name”></iframe>

<iframe seamless src=“name”></iframe>

<iframe seamless=“seamless” src=“name”></iframe>

This attribute may not be supported by many browsers. Therefore, you can use CSS to achieve a similar effect.

Hyperlinks

Hyperlinks can be implemented using the <a> element. This element can be used to link both external (a different HTML document) as well as internal (another location in the same HTML document) links.

All links are underlined and depending upon the nature of the link, the colour of the link changes.

- Blue - Unvisited link
- Purple - Visited link
- Red - Active link

The first attribute of the <a> element is href, which is given the value of the URL.

Syntax for external links:

Text

Syntax for internal links:

Text

The id here is the id of the tag to which you want the link to jump onto. However, if you use only the # value, the link jumps to the top of the page.

The other attribute used with the <a> element is target, which allows you to control the behaviour of the link. For instance, if you want the link to open in another window, then this requirement can be specified using this attribute. The following can be used:

- _blank

  This opens the link in a new browser window.

- _parent

  This opens the link in the parent window.

- _self

  This opens the link in the current window and is the default setting.

- _top

  This opens the link in the topmost frame of the current window.

- <iframe_name>

This opens the link in the <*iframe*> element of the specified name. This is generally used in menus.

Hyperlinks can also be used to send emails using the following syntax:

Text

When the user clicks on the link, an email will be sent to the specified email address.

Embedding Images

The <img> element is used for adding images to the HTML document. The <img> tag is a void element and does not require a closing tag.

The required tag for this element is *src*, which specifies the absolute or relative address at which the image is stored. Another attribute than can be used with the <img> tag is *target*, which is used to specify the text that must be displayed in case the image is not available.

Syntax:

<img src=”address” target=”alternative text”>

It is important to note that you only give references to images and they are not embedded into the HTML document, in actuality. The allowed image formats are jpeg, svg, png and gif.

How to Create Image Map

The <map> element can be used to create a clickable image map. In order to create a link between the map and image, you must set a name of the map and use this name in the usemap attribute of the img tag.

The area of the map element is defined using <area> element. This is a self-closing tag that is used to define the area of the map. The area of the map can be set using the attributes shape, href, alt and coords.

The shape attribute can be give the values poly, circle, rect or default. The default value sets the area as the size of the image. The href and alt attributes are used in the same manner as they are used in the <a> element. Lastly, the coords attribute are defined according to the shape chosen in the following manner:

- poly – x_1, y_1, x_2, y_2, … , x_n, y_n
- circle – x, y, radius
- rect - x_1, y_1, x_2, y_2

For the polygon, the starting and ending coordinates should be the same. In case a discrepancy in this regard exists, a closing value is added.

Example:

<img src =“worldmap.gif” width=“145” height=“126”

alt=“World Map” usemap =”#country” />

<map name=“country”>

<area shape=“circle” coords=“105,50,30”

href=“China.html” alt=“China” />

<area shape=“default” href=“InvalidCountry.html” alt=“Please Try Again” />

</map>

Embedding Plug-ins

Plugins can likewise be embedded into HTML documents using <embed> and <object> elements. Although, both these elements perform similar tasks, they were created by different browsers. As a result, they coexist in HTML5. While the <embed> element provides ease of use, the <object> element provides more functionality to the user. Syntax for these two elements is given below:

The <embed> tag

<embed src="Flash.swf"> </embed>

The src attribute specifies the url or address of the file to be embedded. Other attributes that are taken by the <embed> element includes:

- type - used to specify the MIME type of the content
- height – used to specify the content height in pixels
- width – used to specify the content width in pixels

As mentioned previously, some browsers may not support <embed> element. Therefore, if you are using it in your document, you must add fallback content specification. For instance, if you are embedding a flash file, you must redirect the user to the download link of flash player if the browser does not support it already.

You can do this in the following manner:

<embed src="Flash.swf" >

<img src="address of the image for download flash player"

alt="Download Adobe Flash player" />

</embed>

The <object> tag

The *<object>* tag allows you to embed a variety of multimedia files like video, audio, PDF, applets and Flash. The element accepts the following attributes:

- type – used to specify the MIME type of data
- form – used to specify the form id or ids of the object

- data – used to specify the URL of the resources that the object uses
- usemap – used to specify the name of a client-side image map that the object uses
- name – used to specify the object name
- height – used to specify the height of the object in pixels
- width – used to specify the width of the object in pixels

Of all the attributes mentioned above, it is necessary to mention either the data or type attribute.

Data can be passed into the <object> element using the <param> tag, which is a combination of name and value attributes. Multiple parameters can be declared using this tag.

<object data="file.wav">

<param name="autoplay" value="false" />

</object>

Note:

1. The <object> element must always be used inside the <body> element.
2. HTML5 supports only the attributes listed above and global attributes for the <object> element.

CHAPTER 3: BASICS OF JAVASCRIPT

Interaction is an important facet of any website. In order to connect with the audience in a better way, it is vital to add behaviour to the website. This can be as simple as buttons or as complex as animations. These tasks can be added using JavaScript, which is a web and programming language. This chapter introduces JavaScript and shall help you get started with JavaScript on an immediate basis.

Background

JavaScript is the preferred programming language for client side scripting. Contrary to popular belief, JavaScript is in no way related to Java. In fact, it finds resemblance to ECMAScript. Besides this, the only common thing between this programming language and other programming languages like C and C++ is that it uses curly braces. The international standard for JavaScript is given in ISO/IEC 16262 and ECMA-262 specification.

One of the most important features of this programming language is that it is untyped. In other words, specifying the type of a variable is not necessary for using it. For example, if you have assigned a string to a variable, you can later assign an integer to the same variable. Variables are declared using the keyword *var*.

Data and Expressions

Any program accesses, manipulates and represents data to the user. Data is available in different types and forms. This data can be easily decomposed into values. In JavaScript, data may be represented as a primitive value, object or function.

The data representation at the lowest level is known as primitive data type and includes null, undefined, number, string and Boolean.

Several built-in objects are defined in JavaScript. These entail the Object object, global object, Array object, Function object, Boolean object, String object, Math object, Number object, the RegExp object, Date object, Error object and JSON object.

Any object that is callable is called a function. It may also be referred to as a method if the function is associated with an object.

Data is produced by using expressions, which is a name given to any code that generates a value. It may be assigned a value directly or the value may be computed by substituting and computing an expression composed of operands and operators. It is important to note that an operand can be another expression as well.

Number Data Type

The number data type is a primitive data type and it is internally stored as a floating point number, which is a 64 bit, double precision number. This 64 bit field stores the sign, exponent and fraction. While the leftmost bit is reserved for sign, the bits 0 to 51 are reserved for storing the fraction and bits 52-62 are used for the exponent.

Because of memory limitation on the system, 253 is the highest integer that can be stored. It is important to note that integer calculations generate a precise value. However, fractional data calculation may give imprecise results. Therefore, these values have to be truncated.

In addition to numbers and strings, JavaScript also supports the use of the following special characters.

- undefined specifies that the value has not been assigned yet.
- *NaN stands for* 'Not a Number'
- -Infinity any number that is less than -1.7976931348623157E + 10308 is denoted by Infinity.
- Infinity - any number that exceeds the value 1.7976931348623157E + 10308 is denoted by Infinity.

String Data Type

A string can simply be described as a collection of characters. Whenever you declare character(s) within quotes, the system interprets as a string. Sample strings include:

'Hello World!'

"Hello World!"

However, if you want to include quotes as characters in the string, then you must add '\' before the character. Sample string:

‘You\’re Welcome’

“You\’re Welcome”

JavaScript also supports other escape sequences like \t for tab and \n for newline.

Boolean Data Type

The Boolean data type is a binary data type and return either true or false. These operators are commonly used to indicate results of comparisons. For example,

10 > 5 will give ‘true’ while 5 > 10 will give ‘false’.

Operations on Number data Type

In order to perform calculations, operators are used. JavaScript supports all the basic operators and the operator precedence is as follows:

- Addition and subtraction have the same precedence.
- Multiplication and division have the same precedence.
- The precedence of multiplication and division is higher than that of addition and subtraction.
- If an expression contains several operators of the same precedence, then the expression is evaluated from left to right.

In addition to the basic operators, JavaScript also supports modulo (%) operator. This operator performs division and returns the remainder as result. For example, 23%7 is equal to 2.

Unary Operators

While operators like addition and subtraction need to operands to execute, some operators require only one. An example of such operators is *typeof*, which returns the datatype of the data. For example, *typeof 12* returns ‘number’. Please note that ‘+’ and ‘-‘ can also be used as unary operators. This is the case when they are used to specify the sign of a number.

Logical Operators

Three logical operators are available for use in JavaScript namely, Not (!), Or (||) and And (&&). The results of operations that involve these operators are Boolean (true or false). The results of these operations are computed in accordance with the following:

AND (&&)	Binary operator Both the conditions must be true
OR (||)	Binary operator At least one of the conditions must be true
NOT (!)	Unary operator The value of the condition determined is complemented.

For example,

‘Red’ != ‘Blue’ && 5 > 1 = ‘true’

‘Red’ != ‘Blue’ && 5 < 1 = ‘false’

‘Red’ == ‘Blue’ && 5 < 1 = ‘false’

For conditional operators, JavaScript uses short-circuit evaluation. In other words, if the value of the first condition is computed to be ‘false’, the system does not evaluate the other condition and directly presents the result.

Writing Code in JavaScript

Any statement followed by a semicolon is referred to as a statement. This statement may or may not produce a value unlike expressions, which must inadvertently produce a value.

Variables

Manipulation of data is performed with the help of variables. Data is stored in the memory

and a named reference to this memory location is called a variable. Any identifier is declared as a variable by preceding it with a keyword *var*. A sample declaration of a variable is:

var result;

This statement declares a variable *result*. You may also define the variable as you declare it using a statement like this:

var result = 0;

or

*var result = 23*4+6;*

Certain rules have to be followed while naming variables. These rules are as follows:

- A variable name can be a combination of numbers and characters.
- A variable name cannot start with a number.
- The only special characters allowed in variable names is underscore (_) and dollar sign($).
- There should not be any whitespace in the variable name. For example, ‘result value’ is not allowed.
- JavaScript keywords are reserved and cannot be used.

Please note that JavaScript, unlike HTML is case sensitive. Therefore VAL and val are two different variables. Also, it is recommended that the variable name should be such that it describes the purpose for which it is being used. For example, if we name a variable result, it is evident that this variable will contain the result value computed by the code.

Another convention used in JavaScript is to name variables such that the first letter of the variable name is lowercase. However, every subsequent word in variable name starts with a capital letter. An example of this is the variable name, arrayResult. Besides this, the use

of underscore and dollar sign is discouraged. However, they are used in jQuery objects.

Environment

A complete set of variables and the values they contain form what is called the environment. Therefore, whenever you load a new webpage in the browser, you are creating a new environment. If you take the example of Windows 8, it creates an environment when an application starts and the same is destroyed when the application ends.

Functions

A set of statements that solve a purpose are referred to as a function. The purpose of using functions is code reuse. If your program uses functionality multiple times in the program, then it is implemented as a function, which can be called as and when required. Since, a function is to be called from within the code, parameters can be sent to the function from the code. Upon execution, the function returns a value to the calling function. The syntax for function declaration and definition is as follows:

```
function multiply(a, b){

return a*b;

}
```

The name of the function must always be preceded with the keyword function. The variables a and b are parameters passed into the function and the function return the value obtained by computing a*b. This is a simple function, but you can implement complex and large function depending upon the functionality desired.

Now that you have implemented the function, you must be wondering as to how the function is called. Here is an example:

```
var x=2;
```

var y=5

var c=multiply(x, y);

Here, x and y are arguments that the function multiply will receive as parameters.

JavaScript is a loosely typed language. What that means is that if you pass more arguments to a function than what it is expecting, the system simply uses the first n arguments required and discards the rest. The advantage of this functionality is that you can use already implemented functions and pass the extra argument to scale the function and add functionality to it. On the other hand, you will not be able to get any indication of error if you unintentionally pass the wrong number of arguments.

JavaScript also provides some built-in functions for interacting with the user. These functions are as follows:

- alert

  This function raises an alert with a message and the system resumes operation after the user clicks on the OK button. Sample implementation:

  alert(‘Alert message!’);

- prompt

  This function presents a textbox to the user and asks him or her to give input. You can supply the default value in the presented textbox and the user can click on the OK button to accept that the value entered is correct. Sample implementation:

  var result = prompt(‘Enter a value’, ‘default value’);

- confirm

  This message gives the user the choice to OK or CANCEL an action. Sample implementation:

```
var result = confirm('Do you wish to proceed?');
```

Function Scope

Each variable that you declare possesses a scope of operation, which is the function within which the variable has been declared. This is called the local scope. Unlike, many other languages, which define local scope by the curly braces within which the variable lies, JavaScript's local scope is same as function scope.

In addition to this, JavaScript also supports the concept of global scope, in which variables can be declared global and thus, can be used anywhere in the program.

Nesting Functions

Functions can be nested at any level. In other words, a function can be called from within another function, which could have been called from a different function. However, it is important to note that the scope of variable is within the function in which they are declared.

Conversion of One Data Type to Another

The prompt function discussed in the previous function returns a string. However, you had asked the user to enter a number. In such a scenario, a string to number conversion may be required. In JavaScript, a variable can be converted from one type to another using the following functions:

- Number Function

  This function converts the object supplied to it into number data type. However, if the function is unable to perform the conversion, *NaN* is returned.

- String Function

  This function converts the object supplied to it into string data type.

Conditional Programming

While writing code, you will faced with several situation where you need to execute a different set of instructions if the condition is true and another set of instructions if the same is false.

if-else

In order to implement such scenarios, you can use the if-else construct.

Syntax:

```
If(condition)
{
//code
}
else
{
//code
}
```

Consider a scenario in which you ask the user to enter his or her age using prompt function. Now, you must validate if the age is a valid number, before performing any computation on the value supplied. This is an ideal scenario of implementing conditional programming. Sample implementation for this scenario is:

```
var userAge = prompt('Enter your age: ', '');
if(isNaN(userAge))
{
```

```
alert(‘Age entered is invalid!’);
}
else
{
//code
}
```

In this sample code, the if clause checks if the entered value is a number. If the condition is true, that is the object entered is not a number, the user is given an alert message. However, if the condition is false, the code for else is executed.

It is important to note here that for single statements, it is not necessary to use curly braces. The above mentioned code can also be written as:

```
var userAge = prompt(‘Enter your age: ’, ‘’);
if(isNaN(userAge))
alert(‘Age entered is invalid!’);
else
//code
```

However, it is a good practice to use curly braces as there is scope of adding additional code later on.

if-else if

Another conditional programming construct is if-else if construct. This construct allows you to declare multiple conditions and the actions associated with them. The syntax is:

```
if(condition)
```

```
{
//code
}
else if(condition)
{
//code
}
else
{
//code
}
```

Switch

Multiple else ifs can be implemented using this construct. The execution overhead is high for this conditional programming construct as conditions are sequentially checked for validity. As an alternative, another keyword, switch, is available, which implements multiple conditions in the form of a jump table. Therefore, the execution overhead for switch is lesser than that of if-else if.

Sample implementation:

```
var userChoice = prompt(‘Choose an alphabet: a, b, c’, ‘e’);
switch (userChoice) {
case ‘a’:
alert(‘a chosen\n’);
```

```
break;

case ‘b’:

alert(‘b chosen\n’);

break;

case ‘c’:

alert(‘c chosen\n’);

break;

default:

alert(‘None of the alphabets chosen\n’);

break;

};
```

The switch construct matches that value entered by the user with the values presented in the cases. If a matching value is found, the case is executed. However, in case, none of the case values match the entered input, the default case is executed. Besides this, you can also use conditions in case values and the case for which the condition holds true is executed.

If you do not use the break statement after the code of a case, all the cases following the matching case will be executed. For example, if the user enters ‘b’ for the above example and there are no break statements after the case code, then the output will be:

```
b chosen

c chosen

None of the alphabets chosen
```

Also, it is a good practice to use a break statement in the default case as well.

Note:

If you wish to determine if a keyword has been assigned any value or not, you can use the following code:

```
if(c)
{
//code
}
else
{
//code
}
```

If the variable c has been assigned a not-null value, then the if condition is true and the corresponding code is executed. On the other hand, if the value of variable c is undefined or null, the code within the else construct is executed.

Note:

The value of the following conditions will always be true:

```
” == 0
null == undefined
‘123’ == 123
false == 0;
```

Please note that JavaScript converts the type of the variable concerned for compatibility in

comparisons.

However, if you want to compare both the value and type of two variables, then JavaScript provides another set of operators, === and !===. When the comparisons for the values mentioned in the above example are done using this operator, the resultant will always be false.

Implementing Code Loops

Looping is an important and commonly used construct of any programming language. You will be faced by several situations when you need to perform the same set of instructions, a given number of times. In order to implement this scenario, loops have to be used. Loop constructs available in JavaScript include for, while and do-while.

The while loop includes a condition and as long as the condition remains true, the loop continues to execute. The do – while loop is a modification of the while loop. If the condition in the while is false, the while loop will not execute at all. On the other hand, even if the while condition is false, the do-while loop executes at least once.

Syntax for while loop:

```
while(condition)
{
//code
}
```

Syntax for do-while loop:

```
do
{
//code
```

}

while(condition)

The last type of loop available in JavaScript is for loop. The for loop allows you to initialize the looping variable, check for condition and modify the looping variable in the same statement.

Syntax:

for(initialize; condition; modify)

{

//code

}

Sample code:

for(i=0; i<10; i=i+1)

{

//code

}

This loop will run 10 times.

Note:

1. If at any point in time, you wish the loop to break, you can use the break statement.
2. If you do not specify a condition or specify a condition that is always true, the loop will run infinitely.

Error Handling

Exceptions are expected to occur at several points in your code. Therefore, it is best to

implement a mechanism that can help you deal with these exceptions and avoid crashing.

An exception can be described as an illegal operation or any condition that is unexpected and not ordinary. A few examples of exceptions include unauthorized memory access.

You can perform exception handling at your own level by validating the values of variables before performing any operations. for instance, before performing division, it is advisable to check if the value of the denominator is equal to zero. Any operation that involves division of a number by zero raises the divide-by-zero exception.

However, there are other situations that cannot be handled in this manner. For instance, if the network connection breaks abruptly, you cannot do anything to pre-emptively handle the situation. Therefore, for situations like these, try, catch and finally keywords are used.

The code that is expected to throw an exception is put inside the try block. This exception, upon its occurrence, is caught by the catch block, which executes code that is supposed to be executed immediately after an exception is caught. The catch may also be followed by the finally block, which performs the final cleanup. This block is executed after the execution of try and catch blocks.

Syntax:

```
try
{
//code
}
catch(exception name)
{
//code
```

}

finally

{

//code

}

Working with Objects

JavaScript allows user access to a number of existing objects. One of these objects is an array. This section discusses all the basics related to this chapter. Dealing with objects in JavaScript also includes creation and handling of customized objects. However, this topic shall be covered in the chapter on JavaScript and jQuery.

Arrays

A collection of similar objects that are sequenced contiguously are referred to as an array. This array is given a name and each element can be accessed using the indexer, in the following form:

Let arrName[] be an array of names. The element arrName[2] refers to the third element of the array.

An array can be created using the following three methods:

- Insertion of Items Using Indexer

  An array can be created using the new keyword and then, elements can be added into the array by assigning values to independent elements of the array. The new keyword creates an instance of the object Array using the constructor for the same.

  Sample implementation:

  var arrName = new Array();

arrName [0] = ‘Jack’;

arrName [1] = ‘Alex’;

- Condensed Array

  The second type of implementation also uses the new keyword. However, in this case, the values are assigned to the elements as arguments to the constructor of the Array object.

  Sample implementation:

  var arrName = new Array(‘Jack, ‘Alex’);

- Literal Array

  In this type of array definition, values are provided within the square brackets.

  Sample implementation:

  var arrName = [‘Jack, ‘Alex’];

The advantage of using the first type of definition is that it allows you to assign values to the elements anywhere in the code. On the other hand, the second and third type of implementation requires you to have the exact list of elements with you beforehand.

There are some properties associated with all object. The one property that can come in handy to you is *length*, which is a read-only value and when called return the number of elements present in the array. You can use this property in loops and conditions.

Objects can also have their own functions. These functions are called methods. The methods available for Array include:

- concat

  Returns an array, which is the concatenation of the two arrays supplied to it.

- indexOf

Finds the location of the element concerned in the array and returns the index of the same.

- join

  This method concatenates all the values present in the array. However, all these values are separated by a comma by default. You can specify a delimiter of your choice as well.

- lastIndexOf

  This method works similarly as indexOf. However, it performs the search from the last element of the array. Therefore, it returns the index of the last element that matches the specified criterion.

- pop

  Removes the last element and returns its value.

- push

  Adds the concerned element to the end of the array and returns the changed value of length.

- reverse

  This method reverses the order of the array elements. The original array is modified by this method.

- shift

  Removes and returns the first value. If the array is empty, then undefined is returned.

- slice

  This method requires two arguments, start index and end index. A new array is

created with elements same as the elements present at indexes (start index) and (end index – 1).

- sort

  This method sorts the elements and modifies the original array.

- splice

  This method removes and adds elements to the specified array. The arguments taken by this method are start index (index from where the system should start removing elements), number of elements to be removed and elements to be added. If the value passed for number of elements is 0, then no elements are deleted. On the other hand, if this value is greater than the size of the array, all elements from the start index to the end of the array are deleted.

- toString

  This method creates a string, which is a comma separated concatenated string of all the elements present in the array.

- unshift

  This method adds an element at the first location of the array and return the modified value of length.

- valueOf

  This method returns a string, which is the concatenated, comma-separated string containing all the values present in the array.

Note:

1. When working with functions, you can pass the whole array (using the array name) or a particular element of the array (using array name[indexer]).

2. Array elements can be modified by accessing the element using the indexer. For example, arrName[1] = ‘Danny’; assigns the value ‘Danny’ to the second element of the array.

DOM objects

The primary objects that you need to access while building an application, are the DOM objects associated with it. This access is necessary for you to control and get notified about events that are occurring on the webpage.

The DOM is a representation of a hierarchy of objects. These objects can be accessed using the *document* variable, which is built-in. This variable references the DOM and performs a search, which may return an active or static *NodeList.* While the active NodeList contains a list of elements that keep changing, the static NodeList contains elements that do not change over time. Since the retrieval of the static NodeList takes longer, it is advisable to choose search methods that work with active NodeList.

The search methods available for DOM include:

- getElementById

  This method returns a reference to the first element that has the specified ID.

- getElementsByTagName

  This method returns the active NodeList, which has the specified tag name.

- getElementsByName

  This method returns the active NodeList, which has the specified name. This is a preferred method for option buttons.

- getElementsByClass

  This method returns the active NodeList, which has the specified class name.

However, this method is not supported by Internet Explorer version 8 and earlier.

- querySelector

  This method accepts CSS selector as parameter and return the first matched element. However, this method is not supported by Internet Explorer version 7 and earlier.

- querySelectorAll

  This method accepts CSS selector as parameter and return all the matched elements. Therefore, it returns a static NodeList. However, this method is not supported by Internet Explorer version 7 and earlier.

Events

If you look at JavaScript as an engine, then events are what give it the required spark. Events can occur in two situations. The first type of events are those that occur during user interactions. A user may click an image or enter text. All these are classified as events. Besides this, changes in state of the system are also considered an event. For instance, if a video starts or stops, an event is said to have occurred. The DOM allows you to capture events and execute code for the same.

In JavaScript, events are based on the publish-subscribe methodology. Upon creation of an object, the developer can publish the events that are related to this object. Moreover, event handlers can be added to this object whenever it is used. The event handler function notifies the subscribed events that the event has been triggered. This notification includes information about the event like location of the mouse and key-presses, in addition to several other details relevant to the event.

Capturing events:

There may be situations when an event may be attached to a button click, which may lie

inside a hyperlink. In this situation, there is nesting of elements. Therefore, the event, when triggered, is passed down the DOM hierarchy. This process is called event capturing. However, once the event has reached the element, this event is bubbled up the hierarchy. This process is called event bubbling. This movement of the event across the hierarchy gives the developer an opportunity to subscribe or cancel the propagation, on need basis.

Subscribing to event:

The function, addEventListener, can be used for the subscription process. This function requires three arguments, the event, the function that needs to be called for the event and a Boolean value that determines if the function will be called during the capture or bubble process (true – capture, false – bubble). Mostly, this value is set to false. This is the preferred method for subscription as it is mentioned in the W3C standard.

Sample Code:

```
var btn = document.getElementById(‘btnDownload’);

btn.addEventListener(‘click’, initiateDownload, false);
```

However, other methods also exist, which include giving an online subscription to the html tag. This subscribes the event to the bubble process. The advantage of using this method is that it is the oldest and most accepted method. therefore, you can be sure that this method will work, regardless of what browser you are using. Please see the tag below to understand how this can be done.

```
<button id=‘btnDownload’ onclick=‘initiateDownload();’ >Download</button>
```

You can also use the traditional subscription process that uses JavaScript for subscribing the event.

```
var btn = document.getElementById(‘btnDownload’);
```

```
btn.onclick = initiateDownload;
```

Unsubscribing:

Events can be unsubscribed using the function, removeEventListener, which takes the same set of parameters as addEventListener. For the btn variable used in the previous example, this can be done in the following manner:

```
var btn = document.getElementById('btnDownload');

btn.removeEventListener('click', initiateDownload, false);
```

How to cancel propagation?

The function, stopPropagation, is used for performing this operation. This can be done in the following manner:

```
var btn = document.getElementById('btnDownload');

btn.addEventListener('click', initiateDownload, false);

function initiateDownload (e){

//download

e.stopPropagation();

}
```

How to prevent the default operation?

This can be done by using the function, preventDefault, in the following manner:

```
var hyperlink = document.getElementById('linkSave');

hyperlink.addEventListener('click', saveData, false);

function saveData(e){

//save data
```

```
e.preventDefault();
}
```

JavaScript also provides the *this* keyword, which can be used if you wish to access the event causing element, on a frequent basis.

Window Event Reference

The current browser window is represented by the window variable, which is an instance of the Window object. The following events are associated with this object:

- afterprint
- beforeonload
- beforeprint
- error
- blur
- haschange
- load
- message
- focus
- online
- offline
- pageshow
- pagehide
- redo
- popstate
- storage
- resize
- unload

- undo

Form Event Reference

The actions that occur inside an HTML form trigger the flowing events:

- change
- blur
- focus
- contextmenu
- forminput
- formchange
- invalid
- input
- submit
- select

Keyboard Event Reference

The keyboard triggers the following events:

- keyup
- keypress
- keydown

Mouse Event Reference

The mouse triggers the following events:

- click
- drag
- drop
- scroll

- dblclick
- dragenter
- dragstart
- dragend
- dragover
- dragleave
- mousemove
- mousedown
- mouseover
- mouseout
- mousewheel
- mouseup

Media Event Reference

Media elements like videos, images and audios also trigger events, which are as follows:

- canplay
- abort
- waiting
- durationchange
- canplaythrough
- ended
- emptied
- loadeddata
- error
- loadstart
- loadedmetadata
- play

- pause
- progress
- playing
- readystatechange
- ratechange
- seeking
- seeked
- suspend
- stalled
- volumechange
- timeupdate

Chapter 4: Basics of CSS3

Cascading Style Sheets or CSS provide the presentation that webpages are known for. Although, HTML is capable of providing a basic structure to the webpage, CSS offers developers host of design options. Besides this, it is fast and efficient, which makes it an all more popular design tool.

CSS is known to have evolved from SGML (Standardized Generalized Markup

Language). The goal of efforts made in this direction was to standardize the manner in which web pages looked. The latest version of this technology is CSS3, which is a collection of 50 modules.

The most powerful characteristic of CSS is its cascading ability. Simply, it allows a webpage to take its styles from multiple sheets in such a manner that changes to the style in subsequently read sheets overwrite the style already implemented from one or more of the previous sheets.

How to Define and Apply Style

The definition and application of a style involves two facets or parts, selector and declaration. While the selector determines the area of the webpage that needs to be styled, the declaration block describes the style specifications that have to be implemented. In order to illustrate how it works, let us consider the following example,

```
body {
color: white;
}
```

In this example, the selector selects the body of the webpage and the declaration block defines that the font color should be changed to white. This is a simple example and declarations and selectors can be much more complex than this.

How to Add Comments

Comments can be added to the style sheet using the following format:

```
/*write the comment here*/
```

How to Create an Inline Style

Every element has an associated global attribute, style. This global attribute can be manipulated within the tag for that element to modify the appearance of that element. This type of styling does not require you to specify the selector. Only the declaration block is required. An example of how this is done is given below:

```
<body style='color: white;'>
</body>
```

This HTML tag performs the same functionality as the CSS code specified in the previous section. The advantage of using this approach is that the style information given in this manner overwrites any other styling information. Therefore, if you need to use different

style for one element while the rest of the document needs to follow a different style, then you can use a stylesheet for the document and specify the style for this element in its tag.

How to Use Embedded Style

Another approach for accomplishing the same outcome as inline styles is to use the <style> element within the element concerned, for defining its style specification. Here is how this can be done:

```
<!DOCTYPE html>
<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
<title></title>
<style>
body {
color: white;
}
</style>
</head>
<body>
</body>
</html>
```

How to Create External Style Sheet

For usages where you wish to use the same style for the complete webpage or a number of webpages, the best approach is to use an external style sheet.

This external style sheet can be linked to the HTML page in the following manner:

```
<!DOCTYPE html>

<html xmlns='http://www.w3.org/1999/xhtml'>

<head>

<title></title>

<link rel='stylesheet' type='text/css' href='Content/mainstyle.css' />

</head>

<body>

</body>

</html>
```

You must create a file mainstyle.css, in the Content folder, and put the style rule specified below into the file.

```
body {

color: white;

}
```

Defining Media

It is important to note that a style sheet can contain as many style rules as you want. Besides this, you can also link different CSS files for different media. The different media types are as follows:

- all
- embossed
-

- braille
- print
- handheld
- speech
- screen
- tv
- tty

The media used can be defined in the following manner:

<link rel=‘stylesheet’ type=‘text/css’ href=‘Content/all.css’ media=’all’ />

Defining Character Encoding

You can also define the character encoding used, using the following format:

Style sheet:

Place the following line above the style rule in the style sheet.

@charset ‘UTF-8’;

HTML page:

You must place this line above the link element.

<meta http-equiv=‘Content-Type’ content=‘text/html;charset=UTF-8’ >

Importing style Sheets

As your web pages becomes complex, the style sheets used shall also grow in complexity. Therefore, you may need to use many style sheets. You can import the style rules present in one style sheet to another by using:

@import url(‘/Content/header.css’);

Here, header.css is imported and the url gives the relative address of the style sheet to be

imported.

Importing Fonts

Fonts can be imported using the following format:

@font-face {

font-family: newFont;

src: url(‘New_Font.ttf’),

url(‘New_Font.eot’); /* IE9 */

Selectors, Specificity and Cascading

Selectors can be of three types, class selectors, ID selectors and element selectors. The element selector type is the simplest and requires you to simply name the element that needs to be used. For instance, if you wish to change the background color of the body, then the element selector used is *body.*

While declaring any element, you can assign an ID to it using the id attribute. You can use this ID prefixed with a # as a selector. For example, if you have created a button with ID btnID, then the ID selector for this will be #btnID. Similarly, you can assign a class name to an element using the class attribute. Class name can be used prefixed by a dot(.) in the following manner, .className.

However, if you wish to select all the elements of the webpage, then asterisk (*) to it.

Using Descendent and Child Selectors

You may wish to apply a particular style to a descendant of a selector. This can be done by specifying the complete selector change. It can be done in the following manner:

li a {

text-color: black;

}

On the other hand, you may want to apply to an element only if it is a direct child of the selector. This can be implemented by specifying the parent and child separated by a greater than (>) sign, in the following manner:

li > a {

color: white;

}

Pseudo-element and Pseudo-class Selectors

Now that you know how to apply styles to specific elements, let us move on to implementing styles to more specific sections like the first line of the second paragraph. In order to style elements that cannot be classified on the basis of name, content or is not a part of the DOM tree can be styled using pseudo-classes. The available pseudo-classes include:

- :visited
- :link
- :hover
- :active
- :checked
- :focus
- :nth-last-child(n)
- :not
- :only-child
- :nth-child(formula)
- :lang(language)
- :first-of-type

- :only-of-type

If you want to access information of the DOM tree that is not accessible otherwise, you can use pseudo-elements. Pseudo-elements include:

- ::first-letter
- ::first-line
- ::after
- ::before

Grouping Selectors

Multiple selectors can be used for a style rule. These selectors must be separated by commas. Sample implementation:

```
body, button {
color: white;
}
```

Using Adjacent Selectors

If you want to style the first heading in a div or any similar adjacent elements, the selector is constructed using a plus sign (+) between the two selectors. Sample implementation:

```
div + h1 {
color: white;
}
```

Sibling Selectors

Sibling selectors are similar to adjacent selectors except for the fact that all the matching elements are styled as against adjacent selectors, which only style the first matching element. The selector is constructed using a ~ sign between the two selectors. Sample

implementation:

```
div ~ h1 {

color: white;

}
```

Using Attribute Selector

This selector selects all the elements for which the specified attribute exists. The selector is written in this form:

```
a[title]
```

This selector will select all the links for which the title attribute has been specified. Moreover, this selector type can be modified into attribute-value selector by specifying the attribute value in the following manner:

```
a[title = value]
```

In-Built Styles of Browsers

Every browser has a built-in stylesheet, which is applied to all the webpages opened using this browser. In fact, this stylesheet is applied before any other style sheet. You can define your own style sheet for the browser using the Accessibility option in Tools. However, user style sheets are browser specific. Therefore, if you open a different browser, the style sheet you defined may not be accessible.

In case, you want your user-defined stylesheet to override any other style specified in the HTML page, then you can use the '!important' modifier. This modifier sets highest priority for the specified style statement. Sample implementation:

```
body {

color: white !important;
```

}

Cascading of Styles

The precedence and priority of the styles are decided on the basis of the following parameters.

- Importance
- Specificity
- Textual Order

Working with CSS Properties

Now that you are thorough with the use of selectors, the next step is to look at CSS properties.

Color

One of the most crucial properties that are used in a web page is color, which can be defined using ARGB, RGB and color names.

RGB value are typically defined using a decimal number, which lies between 0-255.

- white #ffffff
- red #ff0000
- black #000000
- green #008000

Color values can also be used instead of the color name. An example of how this can be used is given below.

```
body {

color: #ffffff;

}
```

Another way to specify the color is using the RGB function, which specifies the values of parameters using a number between 0-255 or percentage. Example of this type of declaration is given below:

h1 { color: rgb(255,0,0); }

Other ways to specify color are RGBA, which accepts 4 values and HSL, which defines values for hue, saturation and lightness.

Transparency

The transparency or opacity are defined by a value between 0.0 (invisible) and 1.0 (opaque).

Text

As far as text is concerned, font-face and font-size can be specified. These properties can be defined in the following manner:

h1 { font-family: arial, verdana, sans-serif; }

h1 { font-size: 12px; }

The CSS Box Model

The CSS Box Model assumes that a webpage can be considered to be made up of boxes. The spacing between these boxes are given by margins and padding settings. These properties can be given values in the following manner:

margin: 15px;

padding: 25px;

border: 10px;

Positioning elements

The element used for creating page layouts is . Although, HTML5 recommends the

use of semantic markup instead of div elements, there are still used for content that cannot be styled using semantic markup. A div element can be imagined as a rectangular block and is declared in the following manner:

<!—other elements are enclosed within this area—>

Properties used to define the position of a div element include:

- The position of the div element can be defined using the properties, top, bottom, left and right, in pixels.
- A property, position, is used to define if the position specified is static or relative.
- The float property can be used to allow elements to float to the right or left and is defined as float: left or float: right.
- The clear property places a clear element right after the floating element.
- You can also change the manner in which the browser calculates width with the help of the box-sizing property. This property can take three values: content-box (default setting), border-box and padding-box.

Centering Content

If you are using a fixed width, the div element can be centered using the properties, margin-left and margin-right. If you fix the width and set the margins to *auto,* the extra space on the margins is equally divided. It can be done in the following manner:

#container {

width: 850px;

margin-left: auto;

margin-right: auto;

}

Chapter 5: HTML5 Explained

The chapter focuses on the basics of HTML5 and how they can be used for creating high performance, new-generation pages. However, most of the elements explained in that chapter included elements that HTML5 has received from its predecessors. This chapter takes you a step further in your quest of learning HTML5, introducing concepts that are newly added to this technology.

In the previous chapter on CSS, we introduced the concept of element to you. This element is preferred over all its alternatives as far as page layout creation is concerned. While some people also use the <table> element, it is usually not a recommended solution as it is difficult to maintain as well use. However, both the concepts are elaborated upon this chapter.

Semantic Markup

The and elements are the most commonly used and recommended elements for positioning and formatting content. However, it is recommended that you should use different elements for specifying different sections of the page like header and footer. This shall allow you to position them individually and in a more organized manner. Therefore, the W3C has named these new elements with names like <footer> and <header>.

Browser Support

It is true that your HTML code will not be read by any of your users. However, there are other tools and machines that are constantly parsing your code. These tools include web crawlers, which indexes webpages for searching and NVDA (Non-Visual Desktop Access) devices, which are used by many users with special needs as an alternative for reading and comprehending web pages. Therefore, they require you to use tags that are

understandable.

How to Create HTML5 Documents

Although, the above discussion clearly mentions the importance of using meaning tags and prohibits the use of tags like and , you may still have to use them for styling purposes. As you read on, you will realize how semantic tags must be supplied for providing meaning to your tags. It is important to mention here that semantic tags should be used carefully, and if you realize that there is a need to define custom elements, then go ahead and use the and elements. However, be sure to add an ID and class-name to the elements that describe their meaning as well as possible.

How to Create HTML5 Layout Container

A layout container, as the name suggests, is a container that entails the layout of a page. In other words, the container contains the different sections of the layout or its children in such a manner that they can be positioned in a flexible manner. As a developer, you can easily distinguish between elements on the basis of their class names and IDs. However, this is not true for browsers.

Therefore, there has got to be a way by which you can ask the browser to interpret elements. For instance, you may want to ask the browser to focus on a certain element upon opening. All this and more can be done with the help of layout containers that express elements in such a manner that they are understandable to both the browser and the user.

Some of the commonly used elements for creating a layout container include:

- *<header>*

  It is used to define the header section or the topmost section of the HTML document. This element can also be defined inside the <article> element.

- *<footer>*

  It is used to define the footer section or the bottom-most section of the HTML document. This element can also be defined inside the <article> element.

- *<nav>*

  It is used to define the section that contains all the navigational links.

- *<aside>*

  This element is generally used for sidebars and separates the content in the <aside> element from the content that is outside this element.

- *<section>*

  This element defines a part of the whole section and is named with the elements <h1>, <h2>, <h3>, <h4>, <h5> and <h6>.

- *<article>*

  This element defines a complete unit of content, which you can copy from one location to another. An example of such a unit is a blog post.

Using Roles

The role attribute can be declared inside the and <aside> elements. The role class hierarchy and the usage of roles for providing specific meanings, as far as far as accessibility is concerned, is defined in WAI-ARIA (Web Accessible Initiative - Accessible Rich Internet

Applications).

One of the parent role classes is the landmark role class, which defines the navigational landmarks on the webpage. The child classes of the parent role class include:

- banner

This defines website specific content that is not expected to change from one webpage to another like headers.

- application

  This defines that the specified area is a web application.

- contentinfo

  This defines the information included in the webpage about the webpage like copyright information. This information is mostly a part of the footer content.

- complementary

  This defines a section of the page that is meaningful when detached from the page as well.

- main

  The main web page content is defined using this child role class.

- form

  This defines the area of the webpage that is expected to take webpage inputs.

- search

  This child role class is used to define the location on the webpage that is used for getting the search query from the user and displaying the results of the search.

- navigation

  The area containing navigational links is a part of this child role class.

These roles can be used for providing meaning. However, the new elements included in HTML5 are meaningful themselves. Yet, there are some utilities that are not available in HTML5 and for these, role attribute can be used.

How to Control format using <*div*> element?

As mentioned previously, the element is essentially invisible and does not provide any meaning to the element. However, if you wish to use it for formatting purposes only, then it is perfect for this purpose.

How to Add Thematic Breaks?

The void element <hr/> can be used for adding thematic breaks, which are used for denoting a transition from one set of content to another.

How to Annotate Content?

There are several elements available for annotation. These include <b> and <i>, which you have been using for ages. However, they have new meanings now, in addition to the style that they denote. For instance, the <b> element denotes the style 'bold'. In addition to this, HTML5 adds the meaning 'span of text with special focus, but no change in mood, importance level or implication.'

Although, the use of the bold style makes more sense in this context, but you can still use this element for denoting names of products in reviews or keywords. Similarly, the element indicates the relative importance of content and <i> denotes a change in mood or implication of the content concerned. Besides this, the element is used for text that will be alternatively pronounced by the reader.

How to Use <abbr> for Acronyms and Abbreviations?

In the previous versions of HTML, the <acronym> element was used for this purpose. However, this element has become obsolete and the new element used for this purpose is <abbr>. It is an inline element, which is generally used with other inline elements like and .

Element - <address>

This element is used for defining the contact information of the owner or the author of the webpage.

Quotations and Citations

You can indicate that a particular text is a quote by using the element <blackquote>, which is used for a long quotation, and <q>, which is used for an inline quotation. You can mention the source of the quotation using the cite attribute or the <cite> element. However, using the <cite> element inside <q> and <blackquote> elements is considered a better approach. Please remember that the <cite> element can only mention the name of the work and other information elements like author's name and location of publishing, are not included here.

How to Document Code in HTML5?

There are two elements, <code> and <samp>, are used for documenting code. While the element <code> is used for documenting the source code, the element <sample> is used for the output of the code. A sample HTML for how this is done is given below:

```
<code class="maintainWhiteSpace">

echoContent('Screen');

function echoContent(name)

{

alert('This is' + name + '.');

}

</code>

<samp class="maintainWhiteSpace">

This is Screen.
```

</samp>

The <pre> Element

It is important to mention here that these elements do not preserve the whitespace. Therefore, a class needs to be implemented for this purpose. This class should look like:

style rule.

.maintainWhiteSpace {

white-space: pre;

}

Some browsers may not support this style rule. Therefore, for such browsers, it is advisable to use the element <pre>. Therefore, <pre> element can be used for defining the pre-formatting rules.

The <var> Element

This element is used to declare that the text specified inside it, is a variable. Example:

The variable <var>i</var> represents the number of iterations for the loop to perform.

The <*br* /> and <*wbr* /> Elements

The <br/> element implements a line break. On the other hand, the <wbr/> implements a word break.

The <dfn> Element

There may be occasions when you wish to define a term. This can be done using the <dfn> element, which takes title as one of its attributes.

Working with Figures

Images and figures are an integral part of any web page content. Therefore, every figure can also be viewed as a unit of content, which may consist of a caption and a reference from the page to which it may belong. In order to define one or more images, the <figure> element is used. The element <figurecaption> can be used for defining the caption of the figure.

However, it is important to mention here that the <figure> element does not give any importance to the position of the figure and the same is included along with the content. However, if the position and location of the figure is of importance to you., then you must consider using the element.

The <summary> and <details> Elements

The element <summary> contains the summary of the content of the webpage, which can be displayed in the form of a collapsible list using the <details> element. Therefore, when you load a page, only the contents of the <summary> element will be displayed, while the contents of the <details> element are displayed when the user clicks on the summary.

Other Annotations

In addition to the above mentioned, there are a few more annotations available, which include:

- <*s*> - Used for striking out text
- <*u*> - Used for underlining text
- <*mark*> - Used for highlighting text
- <*ins*> - Used for indicating that the text has been inserted
- <*del*> - Used for indicating that the text has been deleted
- <*small*> - used for indicating that the text must be printed in fine letters
- <*sub*> - Indicates that the text is a subscript

- <*sup*> - Indicates that the text is a superscript
- <*time*> - Used for indicating that the text denotes time and date
- <*kbd*> - used for indicating that the text is a user input

Language Elements

You may need to use characters of Chinese, Japanese or Korean origin in your text. In order to support this inclusion, the element <ruby> can be used. Inside this element, other elements like <bdi> and <bdo>, for defining the isolation and direction of text. Besides this, <rt> and <rp> elements can also be used for placing notation or parentheses in the text of <ruby> element.

Working with Lists

In HTML5, several elements for defining unordered, ordered and descriptive lists exist. A fourth category of 'Custom lists' is also present to allow customization by the developer. The list items for all these are declared using the <li> element. Moreover, all the three types of lists support list nesting.

Ordered Lists

Ordered lists are declared using the element <ol> and the elements of 'order' in this list are brought about by an automatic numbering of the elements that are included in this list. The attributes that can be used with ordered lists include:

- start - Used to set the starting number of the list
- reversed - Used for declaring if the list has to be ordered in an ascending or descending order
- type – Used for declaring the type of the list, which can be A, a, 1 or I.

Unordered Lists

This type of a list is declared using the <ul> element and there is no numbering of

elements in this case. The elements of the lists are simply represented as bullet points.

Description Lists

This type of a list is declared using the <dl> element. Using this element, you can give a description containing zero or more terms. Besides this, the elements of the list are declared using the <dt> element, which specifies the term, and <dd>, which gives a description of the term.

Custom Lists

The developer can make use of CSS styles to create custom lists. In this case, a different style rule can be created for each level of a nested list.

Working with Tables

Another format for arranging and presenting data in webpages is tables. Tables are declared using the <table> element and represents data in a rows-columns format. The cells of the tables are defined using the <tr> and <td> elements. While <tr> is used for rows, <td> is used for columns.

Despite that fact that HTML5 tables are one of the most powerful constructs available to the developer, it is important to understand how and where tables can be most appropriately used. Here are the reasons why tables should not be used:

- The table created for a web page is not rendered until the </table> tag is read. On the other hand, if the same construct is created using the element, the content will be rendered as it is read.
- Tables are extremely difficult to maintain.
- Tables are difficult to interpret for accessibility devices.

Sample Implementation:

<table>

```
<tr>
<td>Frank</td>
<td>1978</td>
</tr>
<tr>
<td>David</td>
<td>1967</td>
</tr>
<tr>
<td>Alex</td>
<td>1989</td></tr>
</table>
```

The table created above will look like:

Frank	1978
David	1967
Alex	1989

As you observe the table, you must have realized that the table is not complete unless we define what is called the table headers. This can be done using the <th> element. You can create table headers both vertically and horizontally. For example, we can define the following table header in the code used above.

```
<table>
<tr>
```

```
<th>Name</td>

<th>Year of Birth</td>

</tr>

<tr>

<td>Franky</td>

<td>1978</td>

</tr>

<tr>

<td>David</td>

<td>1967</td>

</tr>

<tr>

<td>Alex</td>

<td>1989</td></tr>

</table>
```

The resulting table for this code will look like this:

Name	Year of Birth
Franky	1978
David	1967
Alex	1989

In the above case, the table headers are simply of a larger font size. However, you can

style these as you want by style rules. This can be done in the following manner:

```
th {
background-color: black;
color: white
}
```

This style rule will color the cells of the table headers with black color and the text will be written in white.

The normal behavior of most browsers is to automatically place all the <tr> elements in the <tbody>, indicating that this is the body of the table. However, it is a good practice to define the <tbody> explicitly. Besides this, the <thead> and <tfoot> can also be explicitly defined. The header, body and footer of the table can be individually styled using CSS. As a rule, you can have one header element, one or more body elements and one footer element.

The next important content feature that must be added to the table created above is the table caption. In order to define the table caption, the <caption> element is used.

In some cases, you may feel the need to style individual columns. This may seem like a difficult task considering the fact that tables are essentially row centric in HTML. While you have <tr> elements to identify rows, there are no <tc> elements for identifying columns. The <td> element identifies a cell. However, you can still style individual columns using the <colgroup> or <col> elements. The <table> element can have the <colgroup> element, which includes the <col> elements for columns that are a part of this group of elements. In addition, the <col> element also has a *span* attribute for defining the columns that are a part of this group. A sample implementation to explain how this works is given below:

<colgroup>

<col span=“2” class=“vHeader” />

</colgroup>

The CSS style rule for styling this group of columns can be something like this –

.vHeader {

color: red;

}

While the tables discussed till now are regular tables, HTML5 also supports irregular tables, which can be described as tables that have a different number of columns for each row. The rowspan and colspan attributes can be used for managing the layout of the table.

Chapter 6: JavaScript and jQuery

You must have got a hang of the power of JavaScript already. In the chapter on JavaScript, you have already learnt how to create and adding JavaScript for making web pages dynamic. The biggest challenge of web development is to design webpage elements that can run just as well on any browser as different browsers provide support for different elements, making it difficult to find a common ground.

This chapter takes you a step further in JavaScript development by teaching you how to create objects and use them. Besides this, it also introduces you to the concept of jQuery, which attempts at creating browser-compatible code. Although, it cannot promise 100% browser compatibility, but it certainly solves the day-to-day issues regarding the same.

How to Create JavaScript Objects

Anything from numbers to strings are objects in JavaScript. Therefore, it is essential to

know how to create and deal with these effectively. The simplest way to create objects in JavaScript is using the object literal syntax. The following example illustrates how it is done.

```
var customer1 = {

yearOfBirth: 2000,

name: ‘Alex’,

getCustomerInfo: function () {

return ‘Customer: ‘ + this.name + ‘ ‘ + this.yearOfBirth;

}

};
```

This code creates an object customer1, with the data members, name and yearOfBirth and the member function getCustomerInformation. It is also important to note the use of *this* keyword, which accesses the values correctly being used or referenced for the object concerned.

Besides this, you can also create objects dynamically using the new keyword. The methods inherited include:

- constructor
- isPrototypeOf
- hasOwnProperty
- toLocalString
- propertyIsEnumerable
- valueOf
- toString

Once the object has been created, properties can be added to the same in the following

manner:

```
function getCustomer(myName, myYearOfBirth) {

var newCust = new Object();

newCust.name = myName;

newCust.yearOfBirth = myYearOfBirth;

newCust.getCustomerInfo = function () {

return ‘Customer: ‘ + this.name + ‘ ‘ + this.yearOfBirth;

};

return newCust;

}
```

This code creates an object newCust dynamically. Several instances of this can be created in the following manner:

```
var cust1 = getCustomer (‘Alex’, 1978);

var cust2 = getCustomer (‘David’, 1986);
```

Although, JavaScript doesn’t support a particular keyword ‘class’, but you can simulate classes using the method mentioned above.

Namespaces

There is no specific keyword like *namespace* for implementing namespaces. However, namespaces can be implemented using the concepts of classes and objects. If you classify variables and methods into objects and access them as instances of these objects, then you are placing only the names of these objects in the global namespace, reducing the scope of the variables to the object that they belong.

Implementing Inheritance

You can define ‘is-a’ relationships between objects in JavaScript by creating objects and then classifying those objects on the basis of their common characteristics. For instance, if you are implementing an object for employee of a company. You can create objects for specific types of objects like managerTechnical, managerGeneral, technicalStaff, recruitmentStaff and officeStaff and then classify them into objects, technicalStaff, which includes the objects managerTechnical and technicalStaff, and adminStaff, which includes the managerGeneral, recruitmentStaff and officeStaff. In a similar manner, new functions can also be defined.

Working with jQuery

JQuery is a library of browser-compatible helper functions, which you can use in your code to minimize the efforts required for typing, implementation and testing. These functions are essentially written in JavaScript. Therefore, you can also call jQuery, a JavaScript library.

The list of functionalities that are available in jQuery include:

- Attributes, which are a group of methods that can be used for getting and setting attributes of the DOM elements.
- Ajax, which is a group of methods that provide support for synchronous and asynchronous server calls.
- Core Methods are the fundamental jQuery functions.
- Callbacks object is an object provided for management of callbacks.
- Data Methods are methods that facilitate the creation of association between DOM elements and arbitrary data.
- CSS Methods are methods that can be used for getting and setting CSS-related properties.

- Dimensions are methods that can be used for accessing and manipulating the dimensions of DOM elements.
- Deferred object is an object that is capable of registering multiple callbacks while maintaining the data of state change and propagating the same from one callback to the next.
- Forms are methods that are used for controlling form-related controls.
- Traversing, this is a group of methods that provide support for traversing the DOM.
- Effects are methods that can be used for creating animations for your webpage. Events are methods used to perform event-based execution.
- Selectors are methods that can be used for accessing elements of DOM in CSS selectors.
- Offset are methods that are used to position the DOM elements.
- Utilities, which is a group of utility methods

Before getting to use jQuery, you will need to include it into your project. Once you have installed it and you are ready to use it to your project, the next step is to learn how to use it.

First things first, you need to reference the jQuery library on the webpage that needs to use it in the following manner:

<script type="text/javascript" src="Scripts/qunit.js"></script>

<script src="Scripts/jquery-1.8.2.js"></script>

The next thing to know is that the jQuery code that you are hoping to use in your HTML page lies in the jQuery namespace, which has an alias $. Therefore, you can write either jQuery.jFeature or $.jFeature when referring to a feature of jQuery.

Before, you can start using it in your webpages, you will also need to change the default.js file as follows:

```
function initialize() {

txtInput = $(‘#txtInput’);

txtResult = $(‘#txtResult’);

clear();

}
```

This allows you to use jQuery and CSS selectors by matching them using their IDs.

Also, as you move ahead with coding using jQuery, remember to refresh the screen using Ctrl+F5 after making any changes as the browser may not be able to catch the JavaScript modification right away. Moreover, use jQuery objects as much as possible because the cross-browser compatibility that they offer.

A DOM object can be referenced from a jQuery wrapper in the following manner:

```
var domElement = $(‘#txtInput’)[0];
```

Here is a simple code that checks if the element exists before referencing it.

```
var domElement;

if($(‘#txtInput’).length > 0){

domElement = $(‘#txtInput’)[0];

}
```

How to Create a jQuery wrapper for Referencing a DOM element

A jQuery wrapper can be created from a DOM element reference in the following manner:

```
var myDoc = $(document);

var inText = $(this).text();
```

The first statement assigns the wrapped object to the variable. On the other hand, the

second statement wraps the object referenced using *this*.

How to Add Event Listeners

jQuery provides the .on method for subscribing to events. Besides this, you can unsubscribe using the .off method. These methods can be used in the following manner:

$(‘#btnSubmitInfo’).on(‘click’, onSubmit);

$(‘#btnSubmitInfo’).off(‘click’, onSubmit);

How to Trigger Event Handlers

JQuery provides triggers or the method, triggerHandler, for triggering event handlers or handler code execution. This can be done in the following manner:

$(‘#btnSubmitInfo’).triggerHandler(‘click’);

Initialization Code

You will often be faced with the requirement to run an initialization code upon the loading of an HTML document. You can do this using jQuery in the following manner:

<script>

$(document).ready(function () {

initializationFunction();

});

</script>

This can be placed at the bottom of the HTML document. It will call the initializationFunction.

CHAPTER 7: FORMS

In the previous chapters, you have already studied how HTML documents can be created

and manipulated. Taking a lead from them, we can now move on to understanding forms, which is one of the most crucial and commonly used units of content in webpage development.

Simply, a form is a way in which data is collected and sent to a location where it can be processed, which is a server in most cases. However, since we are yet to discuss server side scripting, we will focus on sending the data to an email address. However, it is important to note that we do not recommend this practice and it is used only for understanding purposes.

Web Communications

Before moving to the working of forms, it is important to know some basics of HTTP. A typical web communication consists of the following activities:

1. When a user browses a webpage, a request for access to a web server resource is initiated by sending a GET HTTP request.
2. The request is processed by the server and sends a response to the browser using HTTP protocol.
3. The browser process the server response and presents it to the user in the form of a ‘form’.
4. The user enters inputs to the form and upon hitting the submit or enter button, this data is sent to the server, using HTTP protocol again.
5. This data is again processed by the server and the server posts its response to the browser, which is displayed on the webpage.

Web Servers

Originally, web servers were designed to receive and process requests and send the results back to the browser using HTTP. Initially, when the web pages were simple, such web servers were able to process a good number of requests per unit time. There were no states

involved as sending and receiving requests were as simple as opening a connection, transferring data and closing the connection.

The new age web servers are much more equipped that these simple web servers. The web servers of today implement what is called the ‘keep alive’ features, which ensures that the connection remains open for a definite period of time in which subsequent requests by the same browser to the server can be entertained.

Web Browsers

The web browser is a desktop application, which displays web pages and manages user interactions between the webpages and the server. The communication between the web servers and pages is established using technologies like AJAX and Asynchronous JavaScript.

How is Data Submitted to the Web Server

An HTML form can be created using the <form> element in the following manner:

<form method=“post” action=“getCustomerInformation.aspx” >

Enter Customer Number:

<input type=“text” name=“Number” />

<input type=“submit” value=“Get Customer Information” />

</form>

This form takes in the customer number and returns a page that displays the details of the customer.

However, it is important to note that not all elements can be used for submitting data in a form. The allowed elements for this purpose are:

- <textarea> - It takes a multi-line input

- <button> - It is a clickable button, which can be placed on any content or image.
- <select> - It is a drop-down list, which allows multiple selections. The selected options can be identified using jQuery: $(‘option:selected’)
- <input type=’checkbox’> - It is a checkbox, which has a value attribute used for setting as well as reading the status of the checkbox. The jQuery for identifying checked checkboxes is: $(‘input[type=checkbox]:checked’)
- <input type=’button’> - It is a clickable button, which has a text prompt.
- <input type=’datetime’> - It is a control, which is date and time (UTC).
- <input type=’date’> - It is a control, which is date-only.
- <input type=’email’> - It is a file-select field, which has a browse button for uploading a file.
- <input type=’color’> - It is a color picker.
- <input type=’hidden’> - It is a hidden input field.
- <input type=’datetime-local’> - It is a control, which is date and time (any timezone).
- <input type=’month’> - It is a month-year control.
- <input type=’image’> - It is an image submit button.
- <input type=’password’> - It is a password field, with masked characters.
- <input type=’number’> - It is a numeric field.
- <input type=’range’> - It is a control, which accepts a numeric value and defines the allowed range of values.
- <input type=’radio’> - It is an option button, which has a value attribute for setting and reading the status of the button. The jQuery used for identifying the marked radio buttons is $(‘input[type=radio]:checked’)
- <input type=’search’> - It is a text field, which is used for entering the search string.

- <input type=’reset’> - It is a button that can be used for resetting the fields of a form.
- <input type=’url’> - It is a URL field.
- <input type=’tel’> - It is a telephone number field.
- <input type=’submit’> - It is a submit button.
- <input type=’time’> - It is a control, which accepts a time value.
- <input type=’text’> - It is a single-line text field.
- <input type=’week’> - It is a week-year control.

The <*label*> Element

It is the element that is used to convey the meaning of an element to the user. The text in this element is displayed inside the textbox, which is auto-removed when the user clicks on it. You can also specify the style of a label.

Specifying Parent Forms

There may be situations where the submission elements of a form may not lie inside the same construct. Therefore, gathering data, in this case, can be quite a challenge. In order to address this issue, HTML5 provides an attribute, id, which can be set for multiple form elements. This shall allow data collection from different form elements in one go.

How to Trigger Form Submission

Upon triggering, all the data collected from the submission elements of the form or forms of the same id is sent to the server using an HTTP method. The <input> element can be used for triggering form submission. Besides this, you can also use JavaScript for this purpose. In order to implement this, you must give an id to the concerned form, myFirstForm. The default.js file, which is linked to the HTML document, must contain the following code:

```
$(document).ready(function () {
```

```
$(‘#myFirstButton’).on(‘click’, submitMyFirstForm);

});

function submitMyFirstForm() {

$(‘#myFirstForm’).submit();

}
```

If the method attribute of the form is not given any value, then it is set to a default GET. Moreover, the action attribute will also have a default value. Therefore, the button click will reference the page to same page. However, the URL will now include a QueryString, which is a combination of values selection or entered by the user. For instance, if the form requests the user to enter Customer ID and the user enters 1245, then the QueryString will be:

customerID=1245

This QueryString will be appended to the URL in the following manner:

Mywebpage.asp? customerID=1245

It is also important to mention here that the QueryString is URI encoded. Therefore, special characters are represented by specific values. For instance, a space is represented as ‘+’ and exclamation mark (!) as ‘%21’. Name-value pair is represented as ‘name=value’ and name-value pairs are separated by ‘&’.

How to Serialize the Form

You can serialize the form using the jQuery serialize method. This can be done in the following manner:

```
var myFormData = $(‘#myFirstForm’).serialize();
```

You can decode the URI-encoded string using the following:

var data = decodeURIComponent(myFormData);

Using Autofocus Attribute

By default, the focus is not set to any element of the form. However, you can set focus using the focus method, which can be implemented in the following manner:

$(‘input[name=“firstName”]’).focus();

However, you can also set autofocus in the following manner:

<input type=“text” name=“firstName” autofocus=“autofocus”/>

How to Use Data Submission Constraints

A form can send data only if it satisfies the following constraints:

- Name attribute must be set.
- You must have set the value of form submission element.
- The <form> element must have its form submission element defined and form submission elements should not be disabled.
- If multiple submit buttons have in implemented, the values will be submitted only on the click of the activated submit button.
- Select the check boxes
- Select the Option buttons
- The <option> elements must have set <option> elements.
- If there is a file selection field, one of the fields must be selected.
- The declare attribute of the object elements must be set

Always remember that the reset buttons don’t send any data and the form need not have an ID for its data to be sent.

How to Use POST or GET

There are two HTTP methods available for submitting data to the server. These methods

are GET and POST. In the former case, the URL is appended with the QueryString. However, in case of the latter, the information is sent within the message body.

Form Validation

It is beneficial to understand that the root of all security issues in web application is user data. The moment you decide to open your application to user data and interactions, you are making your application vulnerable to security threats. Therefore, you need to validate any data that you receive before processing it to prevent any issues from cropping up. Validation can be provided at the browser or server end. However, server side validation is recommended as browser-level validation can be easily manipulated.

The simplest form of validation that you can implement is using the required attribute in the <select> element. You can set this attribute in the following manner:

<select name=“dateOfBirth” required=“required”>

The validation error generated is browser dependent and each browser has a different method of communication to the user that a field is required for submission.

Besides this, the placeholder attribute is also available, which keep the prompt fixed on the unfilled field until a value for the same is provided by the user. It can be implemented in the following manner:

<input type=“text” name=“Date of birth” required=“required” placeholder=“enter the date of birth”/>

The addition of time, date and type based inputs in HTML5 makes validation much simpler as they can directly be matched to see if they have valid input values or not. Besides this, email, numbers and ranges can also be easily validated.

HTML5 performs validation and matches it with the :valid or :invalid pseudoclasses. If the validation is successful, the value is matched to :valid, else it is matched to :invalid.

However, if a value is not ‘required’, it is matched to :optional pseudoclass.

Chapter 8: Web Services

All the chapters discussed till now dealt with browser level coding and scripting. However, now it is time to move on to server side scripting. This chapter focuses on the use of JavaScript at the server level, which is possible with the help of Node.js, and how you can work around with web services.

Basics of Node.js

Node.js is a platform, which is made on Google Chrome, and can be used for creating scalable and flexible applications. It allows you to write JavaScript code for the server. However, before you can begin, you must download and install Node.js on your system.

Writing a Basic Code

The first step is to open any text editor and create a file named myFile.js. In the file, write the code:

```
var http = require(‘http’);

http.createServer(function (request, response) {

response.writeHead(200, {‘Content-Type’: ‘text/plain’});

response.end(‘Hello World!\n’);

console.log(‘Handled request’);

}).listen(8080, ‘localhost’);

console.log(‘Server running at http://localhost:8080/’);
```

The first line loads the http module while the second line creates a server object. The function createServer takes in two parameters, request and response. All the website handling is done from these functions. In this example, the response function ends by

writing ‘Hello World’ on the screen.

The function createServer, returns a server object, which call the function, listen. This function listens at the port 8080 and the IP address of the host is set to 127.0.0.1. therefore, if there is a network adapter installed on your system, your web server will start listening to web requests rights away. The last line prints a line on the screen to let the user know that the server is running and listening to requests.

Once you have created the file and saved the contents of the file as mentioned above, you must open the command prompt and write the command:

Node myFile.js

Now, keeping the command prompt active, you must open the web browser and type the address: http://localhost:8080/

As soon as the request is sent and a response is received, the same is communicated to the user using the console window. If you have been able to do this successfully, then you have just created your first node.js website. If you wish to stop the running of the code, you can just press Ctrl+C.

Now that you know how requests are received, it is time to look at how these requests are processed and responses are generated. You may need to use the *url* module for parsing the QueryString.

The code mentioned below shows how you can parse the URL string and generate a response in accordance with it.

```
var http = require(‘http’);

var url = require(‘url’);

http.createServer(function (request, response) {

var url_parts = url.parse(request.url, true);
```

```
response.writeHead(200, {‘Content-Type’: ‘text/plain’});

response.end(‘Hey ‘ + url_parts.query.name + ‘.\n’);

console.log(‘Handled request from ‘ + url_parts.query.name);

}).listen(8080, ‘localhost’);

console.log(‘Server is running at: http://localhost:8080/’);
```

You can test the running of this code in the similar manner as the previous code.

How to Create Node.js Module

You can create modules by writing code in the form of functions and then, calling these modules from the main code.

```
var myHttp = require(‘http’);

var myUrl = require(‘url’);

function start(){

http.createServer(function (request, response) {

var url_parts = url.parse(request.url, true);

response.writeHead(200, {‘Content-Type’: ‘text/plain’});

response.end(‘Hello ‘ + url_parts.query.name + ‘!\n’);

console.log(‘Handled request from ‘ + url_parts.query.name);

}).listen(8080, ‘localhost’);

console.log(‘Server running at http://localhost:8080/’);

}

exports.start = start;
```

when you save this piece of code in a file, a module is created. This module can be used by other functions using the require function. For instance, if the file is saved as sample1.js, then the start() can be used in another function using:

var samplex = require('./sample1.js');

sample1.start();

How to Create a Node.js package

A collection of modules is referred to as an application. Once you have published your package, it can be installed and used. Consider for example, a package of different mathematical modules.

The root folder must have the following:

README.md

\packName

\lib

main.js

\bin

mod1.js

mod2.js

Creating Aggregate Module

You may wish to make only one object for the user to access. The user should be able to access all the modules of the package through this object. In order to accomplish this, a main.js module must be created in the bin folder that must define the modules to be included in the module.exports.

How to Create README.md File

The README.md file is a help file that can be used by the developer as a startup guide for using your package. The extension of this file is .md, which is a short form for mark-down. This format gives readability to the text written in this file.

A sample file of this type is given below:

samplePackage package

==========

In samplePackage, the following functions are available:

- **add** Performs addition of two numbers and presents the result.

- **sub** Performs subtraction of one number from the other and presents the result.

How to Create package.json File

This file contains the metadata for the package and can be created manually using the command:

npm init

This command creates the file, which can later be edited. A sample file is given below:

{

“name”: “sampleFile”,

“version”: “0.0.0”,

“description”: “This is a sample file “,

“main”: “bin/main.js”,

“scripts”: {

“test”: “echo "This is a test file" && exit 1”

},

“repository”: ””,

“keywords”: [

“sample”,

“example”,

“add”,

“sub”

],

“author”: “XYZ”,

“license”: “ABC”

}

In addition to test scripts, you can also give git scripts, which are the best available source control managers.

How to Publish a Package

As mentioned previously, a package can be defined in terms of a folder structure. When you publish your package, you make it accessible to all users. In order to perform this operation, you must use the npm command, which is also the command used for searching and installing packages. However, you shall be required to create an account for yourself before publishing any of your packages using the command: npm adduser. After you enter the required information, your account is created. However, what this also means us that there is no validation required. Therefore, anyone can add code to the repository. Therefore, you should be careful while downloading and installing packages from the registry.

In order to publish a package, you simply need to go to the root directory of the package

and enter the command npm publish in the command prompt.

How to Install and Use the Package

A package that is published can be downloaded and installed by any user. You simply need to go to the folder and give the command, npm install samplePackage. This installs the package locally. On the other hand, if you wish to install the package globally, you can give the command, npm install –g samplePackage. For a global installation, you will need to create a link from each application to the global install using the command, npm link samplePackage.

The route to a global install is a junction. You can get into the node_modules folder and back using the cd command. Once you are inside the folder, you can give the command: npm install contoso, to initiate an install. You can now write some code that uses the package. A sample is given below:

```
var samplePackage = require('samplePackage');

var finalResult = 0;

console.log();

finalResult = samplePackage.add (5,10);

console.log('add (5,10) = ' + finalResult);

console.log();

result = samplePackage.sub (50,10);

console.log('sub(50,10) = ' + finalResult);

console.log();

console.log('done');
```

This code tests the modules of the package. You can execute the code using:

node main

The package can be uninstalled locally using:

npm uninstall samplePackage

However, if you wish to uninstall the package globally, you can do it using

npm uninstall -g samplePackage

How to Use Express

1. The first step is to install Node.js and create a sample for keeping all .js files and projects. Besides this, you must also install Express, which is a web application framework for Node.js development.
2. You can create a package using the following set of commands and instructions.

1. npm init
2. You can create myPackage.js file containing the following contents:

```
{
“name”: “Sample”,
“version”: “0.0.0”,
“description”: “This is a sample website.”,
“main”: “main.js”,
“scripts”: {
“test”: “echo "Error: Test not specified" && exit 1”
},
“repository”: ””,
“author”: “XYZ”,
```

“license”: “BSD”

}

“private”: true,

“dependencies”: {

“express”: “3.0.0”

}

}

In order to use the file in Express, dependencies have to be added. Moreover, if you do not define it to be private, you may get an error from the firewall of your computer as it tries to load the page.

3. Give the install command: npm install

4. You can use the command, npm ls, to see if the package is present in the registry.

4. You can create a simple application using the following set of instructions:

0. Create a file myApp.js and add the following to the file:

var express = require(‘express’);

var app = express();

2. You can define the route using the myApp.Method() syntax.

app.get(‘/’, function(request, response){

response.send(‘Hey World!’);

});

The code mentioned above will send the response ‘Hey World!’ as and when a request is received.

3. The last section of code that must be added is for listening to the request. This code is as follows:

   var port = 8080;

   app.listen(port);

   console.log('Listening on port: ' + port);

4. Once the file is complete, you can save it and run it using the command, node app. So, now if you open the browser and enter the address http://localhost:8080/, you will get the response *Hey World!*.

5. You can add webpages to applications by replacing the app.get statement with app.use(express.static(__dirname + '/public')); This will allow you to use the same code for a number of webpages. Sample implementation of this concept is given below:

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title></title>

</head>

<body>

<form method="get" action="/submitHey">

Enter First Name: <input type="text" name="firstName" />

<input type="submit" value="Submit" />

</form>

</body>
```

```
</html>
```

Please note that the action attribute is set to /submitHey. In other words, this resource is called at the server for handling the data that is passed to it using the QueryString. The myApp.js file should contain the following:

```
var express = require('express');

var app = express();

app.use(express.static(__dirname + '/public'));

app.get('/SubmitHey', function (request, response) {

response.writeHead(200, { 'Content-Type': 'text/html' });

response.write('Hey ' + request.query.userName + '!<br />');

response.end('Enjoy.');

console.log('Handled request from ' + request.query.userName);

});

var port = 8080;

app.listen(port);

console.log('Listening on port: ' + port);
```

The app can be run in the manner mentioned above.

5. The *formidable* package can be used for posting back data. While the previous method used the GET method, this method uses the POST method.

1. To illustrate how it works, create an HTML as follows:

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
```

<head>

<title></title>

</head>

<body>

<form method=”post” action=”/SubmitHeyPost”>

Enter Name: <input type=“text” name=“firstName” />

<input type=“submit” value=“Submit” />

</form>

</body>

</html>

2. Now, in the command prompt, you need to a give a command for retrieving the formidable package, which is:

   npm info formidable

3. You can also modify the package.jason file in the following manner:

   {

   “name”: “HelloExpress”,

   “version”: “0.0.0”,

   “description”: “Sample Website”,

   “main”: “index.js”,

   “scripts”: {

   “test”: “echo "Error: test not specified" && exit 1”

```
},
“repository”: ””,
“author”: “XYZ”,
“license”: “BSD”,
“private”: true,
“dependencies”: {
“formidable”: “1.x”,
“express”: “3.0.0”
}
}
```

4. Now you can install the *formidable* package by typing the following command into the command prompt:

```
npm install
```

   This command installs the package locally. Therefore, you will need to add a line to myApp.js that allows the file to reference the package:

```
var formidable = require(‘formidable’);
```

5. A sample myApp.js file shall look like this:

```
var express = require(‘express’);
var app = express();
var formidable = require(‘formidable’);
app.use(‘/forms’, express.static(__dirname + ‘/public’));
app.post(‘/SubmitHeyPost’, function (request, response) {
```

```
if (request.method.toLowerCase() == ‘post’) {
var form = new formidable.IncomingForm();
form.parse(request, function (err, fields) {
response.writeHead(200, { ‘Content-Type’: ‘text/html’ });
response.write(‘Hey ‘ + fields.userName + ‘!<br />’);
response.end(‘Enjoy this POST.’);
console.log(‘Handled request from ‘ + fields.userName);
});
}
});
app.get(‘/SubmitHey’, function (request, response) {
response.writeHead(200, { ‘Content-Type’: ‘text/html’ });
response.write(‘Hey ‘ + request.query.userName + ‘!<br />’);
response.end(‘Enjoy. ‘);
console.log(‘Handled request from ‘ + request.query.userName);
});
var port = 8080;
app.listen(port);
console.log(‘Listening on: ‘ + port + ‘port’);
```

6. You can now run the application in a similar manner as you did for the previous example.

Working with web services

One of the biggest drawbacks of a typical website scenario is that the HTML page is repainted even if the new page is the same as the previous page. This causes you to lose bandwidth and resources. This drawback can be addressed using web services, which can be used for sending and receiving data, with the benefit that the HTML page is not repainted. The technology used for sending requests is AJAX or Asynchronous JavaScript and XML. This technology allow you to perform the data sending operation asynchronously.

Before moving any further, it is important to know the basics of web services and how they can be used. A client needs to communicate with the web server on a regular basis and this communication is facilitated by the web service. In this case, the client can be anyone from a machine using the web service to the web service itself. Therefore, the client, regardless what it is, needs to create and send a request to the web service, and receive and parse the responses.

You must have heard of the term mashups, which is a term used to describe applications that pierce together web services. Two main classes of web services exist, which are arbitrary web services and REST or representational state transfer. While the set of operations are arbitrary in the first case, there exists a uniform operations set in the second.

Representational State Transfer (REST)

This framework uses the standard HTTP operations, which are mapped to its create, delete, update and retrieve operations. Moreover, REST does not focus on interacting with messages. Instead, its interactions are focused towards stateless resources. This is perhaps the reason why REST concept is known for creation of clean URLs. Examples of REST URLs include http://localhost:8080/Customers/2, which deletes a customer and

http://localhost:8080/Vehicles?VIN=XYZ12, which is used to retrieve the information about a vehicle for which a parameter is passed using GET method.

Some firewalls may not allow the use of POST and GET methods. Therefore, it is advisable to use ‘verb’ in the QueryString. An example of how the URL will look like is:

http://localhost:8080/Vehicles?verb=DELETE&VIN=XYZ987

The HTTP protocol also allows you to implement security using the HTTPS version. REST provides several benefits like easy connection, faster operation and lesser consumption of resources. However, many developers prefer to use JSON or (JavaScript Object Notation) because it is compact in size. Besides this, REST only supports GET and POST, which restricts its capabilities, Therefore, some developers switch to RESTFUL.

Arbitrary Web Services

This type of web services is also referred to as big web services. An example of such services is WCF or Windows Communication Foundation. Arbitrary web services expand their realm of operations by not mapping their operations to only aspects of the protocol. As a result, they provide more functionality, which include many security mechanisms and message routing.

This type of web services possess a typical interface format, which can be used by the client for reading and parsing information. As a result, the client can make calls to the service immediately. A common API format is the Web Services Description Language (WSDL). In case of arbitrary web services, the client must assemble its request with the help of a SOAP (Simple Object Access Protocol) message. This web service does not use the HTTP protocol and instead uses the TCP.

How to Create RESTful Web Service using Node.js

In the example mentioned previously, the samplePackage can be exposed as web service.

The GET method can be used on the package and the operation is passed as a parameter. A good RESTful implementation of this package can look something like this:

http://localhost:8080/samplePackage?operation=add&x=1&y=5

How to Use AJAX to Call Web Service

Web services can be called asynchronously using AJAX, which is in actuality a JavaScript. Instead of making a call to the server and repainting the HTML document, AJAX just calls back to the server. In this case, the screen is not repainted. A sample implementation is given below:

A MyPage.html can be created with the following code:

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title></title>

<script type="text/javascript" src="/scripts/jquery-1.8.2.min.js"></script>

<script type="text/javascript" src="/scripts/default.js"></script>

</head>

<body>

<form id="myForm">

Enter Value of X:<input type="text" id="x" />

Enter Value of Y:<input type="text" id="y" />

Result of Operation:

<button id="btnAdd" type="button">Add the Numbers</button>

</form>

</body>

</html>

The default.js file must be modified to contain the code required for processing these functions. Be sure to check the version of jQuery and whether it matches the version name that you have mentioned in your default.js file. The <form> element used here is only a means of arranging the format of data and the data is not actually sent via the form to the server. The JavaScript and jQuery access the data entered and perform the AJAX call.

How to Use XMLHttpRequest

The object that actually makes an AJAX call is XMLHttpRequest, which can be used for sending/receiving XML and other types of data. This object can be used in the following manner:

```
var xmlhttp=new XMLHttpRequest();

xmlhttp.open("GET","/add?x=50&y=1",false);

xmlhttp.send();

var xmlDoc=xmlhttp.responseXML;
```

The first line creates the object while the second line sets up the use of GET method with the specified QueryString and the use of 'false' indicates that the operation must be performed asynchronously. The next line sends the request and the last line sets the response to a variable, which can be later read and parsed for processing the response.

However, the output generated is JSON and not XML, therefore, the default.js file must be changed to:

```
$(document).ready(function () {
```

```
$(‘#btnAdd’).on(‘click’, addTwoNum)

});

function addTwoNum() {

var x = document.getElementById(‘x’).value;

var y = document.getElementById(‘y’).value;

var result = document.getElementById(‘finalResult’);

var xmlhttp = new XMLHttpRequest();

xmlhttp.open(“GET”, “/add?x=” + x + “&y=” + y , false);

xmlhttp.send();

var jsonObject = JSON.parse(xmlhttp.response);

result.innerHTML = jsonObject.result;

}
```

The code extracts the x and y values from the <input> element for the same. After this, the XMLHttpObject is created and the open method is called using the QueryString. After the execution of the send function, the response string is parsed. In order to test the page, you can give the command:

```
node app
```

This command starts the web service, after which you can open the browser window with the link:

```
http://localhost:8080/SamplePage.html
```

this code is operational now. However, you may wish to perform the AJAX call in an asynchronous manner. For this, you must locate the open method and change the ‘false’

parameter to ‘true’. Besides this, you will also be required to subscribe to *onreadystateschange* for managing asynchronous call. This can be implemented in the following manner:

```
function addTwoNum () {

var a = document.getElementById(‘a’).value;

var b = document.getElementById(‘b’).value;

var result = document.getElementById(‘finalResult’);

var xmlhttp = new XMLHttpRequest();

xmlhttp.onreadystatechange = function () {

if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {

var jsonObject = JSON.parse(xmlhttp.response);

result.innerHTML = jsonObject.result;

}

}

xmlhttp.open(“GET”, “/add?a=” + a + “&b=” + b , true);

xmlhttp.send();

}
```

The codes for states are as follows:

- 0 - Uninitialized
- 1 - Loading
- 2 - Loaded
- 3 - Interactive

- 4 - Completed

If progress events are provided by the server, you can subscribe to the browser’s progress event. Then, an event listener can be added to initiate the execution of the code when the event is triggered. This can be done in the following manner:

Click Here To Read More…