

## About the Tutorial

Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. This tutorial gives a complete understanding of Java.

This reference will take you through simple and practical approaches while learning Java Programming language.

## Audience

This tutorial has been prepared for the beginners to help them understand the basic to advanced concepts related to Java Programming language.

## Prerequisites

Before you start practicing various types of examples given in this reference, we assume that you are already aware about computer programs and computer programming languages.

## Execute Java Online

For most of the examples given in this tutorial, you will find a ‘Try it’ option, which you can use to execute your Java programs at the spot and enjoy your learning.

Try following the example using the ‘Try it’ option available at the top right corner of the following sample code box −

public class MyFirstJavaProgram {

public static void main(String []args) { System.out.println("Hello World");

}

}

## Table of Contents

[About the Tutorial i](#_TOC_250033)

[Audience i](#_TOC_250032)

[Prerequisites i](#_TOC_250031)

[Execute Java Online i](#_TOC_250030)

[Table of Contents ii](#_TOC_250029)

[JAVA – BASICS 1](#_TOC_250028)

1. Java – Overview 2

[History of Java 3](#_TOC_250027)

[Tools You Will Need 3](#_TOC_250026)

[Try It Option 4](#_TOC_250025)

[What is Next? 4](#_TOC_250024)

1. Java - Environment Setup 5

[Try it Option Online 5](#_TOC_250023)

[Local Environment Setup 5](#_TOC_250022)

[Popular Java Editors 6](#_TOC_250021)

[What is Next? 6](#_TOC_250020)

1. Java – Basic Syntax 7

[First Java Program 7](#_TOC_250019)

[Basic Syntax 8](#_TOC_250018)

[Java Identifiers 9](#_TOC_250017)

[Java Modifiers 9](#_TOC_250016)

[Java Variables 9](#_TOC_250015)

[Java Arrays 9](#_TOC_250014)

[Java Enums 10](#_TOC_250013)

[Java Keywords 10](#_TOC_250012)

[Comments in Java 11](#_TOC_250011)

[Using Blank Lines 12](#_TOC_250010)

[Inheritance 12](#_TOC_250009)

[Interfaces 12](#_TOC_250008)

[What is Next? 12](#_TOC_250007)

1. Java – Objects & Classes 13

[Objects in Java 13](#_TOC_250006)

[Classes in Java 14](#_TOC_250005)

[Constructors 14](#_TOC_250004)

[How to Use Singleton Class? 15](#_TOC_250003)

[Creating an Object 17](#_TOC_250002)

[Accessing Instance Variables and Methods 18](#_TOC_250001)

[Source File Declaration Rules 20](#_TOC_250000)

Java Package 20

Import Statements 21

A Simple Case Study 21

What is Next? 23

1. Java – Basic Datatypes 24

Primitive Datatypes 24

Reference Datatypes 26

Java Literals 26

What is Next? 28

1. Java – Variable Types 29

Local Variables 29

Instance Variables 31

Class/static Variables 33

What is Next? 34

1. Java – Modifier Types 35

Java Access Modifiers 35

Java Non-Access Modifiers 38

The Static Modifier 38

The Final Modifier 39

The Abstract Modifier 41

Access Control Modifiers 43

Non-Access Modifiers 44

What is Next? 44

1. Java – Basic Operators 45

The Arithmetic Operators 45

The Relational Operators 47

The Bitwise Operators 49

The Logical Operators 52

The Assignment Operators 53

Miscellaneous Operators 57

Precedence of Java Operators 59

What is Next? 59

1. Java – Loop Control 60

While Loop in Java 61

for Loop in Java 62

Do While Loop in Java 65

Loop Control Statements 67

Break Statement in Java 67

Continue Statement in Java 69

Enhanced for loop in Java 70

What is Next? 71

1. Java – Decision Making 72

If Statement in Java 73

If-else Statement in Java 74

The if...else if...else Statement 76

Nested if Statement in Java 77

Switch Statement in Java 78

The ? : Operator: 80

What is Next? 81

1. Java – Numbers Class 82

Number Methods 83

Java XXXValue Method 86

Java – compareTo() Method 87

Java – equals() Method 88

Java – valueOf() Method 89

Java – toString() Method 91

Java – parseInt() Method 92

Java – abs() Method 93

Java – ceil() Method 94

Java – floor() Method 95

Java – rint() Method 96

Java – round() Method 97

Java – min() Method 98

Java – max() Method 99

Java – exp() Method 100

Java – log() Method 101

Java – pow() Method 102

Java – sqrt() Method 103

Java – sin() Method 104

Java – cos() Method 105

Java – tan() Method 106

Java – asin() Method 107

Java – acos() Method 108

Java – atan() Method 109

Java – atan2() Method 110

Java – toDegrees() Method 111

Java – toRadians() Method 112

Java – random() Method 113

What is Next? 114

1. Java – Character Class 115

Escape Sequences 115

Character Methods 117

Java – isLetter() Method 117

Java – isDigit() Method 118

Java – isWhitespace() Method 119

Java – isUpperCase() Method 120

Java – isLowerCase() Method 121

Java – toUpperCase() Method 122

Java – toLowerCase() Method 123

Java – toString() Method 124

What is Next? 125

1. Java – Strings Class 126

Creating Strings 126

Java – String Buffer & String Builder Classes 126

StringBuffer Methods 127

Java – String Buffer append() Method 128

Java – String Buffer reverse() Method 129

Java – String Buffer delete() Method 130

Java – String Buffer insert() Method 131

Java – String Buffer replace() Method 132

String Length 135

Concatenating Strings 136

Creating Format Strings 136

String Methods 137

Java – String chartAt() Method 142

Java – String compareTo(Object o) Method 143

Java – String compareTo(String anotherString) Method 144

Java – String compareToIgnoreCase() Method 145

Java – String concat() Method 146

Java – String contentEquals() Method 147

Java – String copyValueOf(char[] data) Method 148

Java – String copyValueOf(char[] data, int offset, int count) Method 149

Java – String endsWith() Method 150

Java – String equals() Method 151

Java – String equalsIgnoreCase() Method 152

Java – String getBytes(String charsetName) Method 154

Java – String getBytes() Method 155

Java – String getChars() Method 156

Java – String hashCode() Method 157

Java – String indexOf(int ch) Method 158

Java – String indexOf(int ch, int fromIndex) Method 159

Java – String indexOf(String str) Method 160

Java – String indexOf(String str, int fromIndex) Method 161

Java – String Intern() Method 162

Java – String lastIndexOf(int ch) Method 163

Java – String lastIndexOf(int ch, int fromIndex) Method 164

Java – String lastIndexOf(String str) Method 165

Java – String lastIndexOf(String str, int fromIndex) Method 166

Java – String length() Method 167

Java – String matches() Method 168

Java – String regionMatches() Method 169

Java – String regionMatches() Method 171

Java – String replace() Method 173

Java – String replaceAll() Method 174

Java – String replaceFirst() Method 175

Java – String split() Method 176

Java – String split() Method 178

Java – String startsWith() Method 180

Java – String startsWith() Method 181

Java – String subsequence() Method 182

Java – String substring() Method 183

Java – String substring() Method 184

Java – String toCharArray() Method 186

Java – String toLowerCase() Method 187

Java – String toLowerCase() Method 188

Java – String toString() Method 189

Java – String toUpperCase() Method 189

Java – String toUpperCase() Method 190

Java – String trim() Method 191

Java – String valueOf() Method 192

1. Java – Arrays 196

Declaring Array Variables 196

Creating Arrays 196

Processing Arrays 198

The foreach Loops 199

Passing Arrays to Methods 199

Returning an Array from a Method 200

The Arrays Class 200

1. Java – Date & Time 202

Getting Current Date & Time 203

Date Comparison 204

Simple DateFormat Format Codes 205

Date and Time Conversion Characters 208

Parsing Strings into Dates 209

Sleeping for a While 210

Measuring Elapsed Time 211

GregorianCalendar Class 212

1. Java – Regular Expressions 218

Capturing Groups 218

Regular Expression Syntax 220

Methods of the Matcher Class 223

1. Java – Methods 230

Creating Method 230

Method Calling 231

The void Keyword 232

Passing Parameters by Value 233

Method Overloading 235

Using Command-Line Arguments 236

The Constructors 237

Parameterized Constructor 238

The this keyword 239

Variable Arguments(var-args) 242

The finalize( ) Method 243

1. Java – Files and I/O 244

Stream 244

Standard Streams 247

Reading and Writing Files 248

ByteArrayInputStream 250

DataInputStream 253

FileOutputStream 255

ByteArrayOutputStream 256

DataOutputStream 259

File Navigation and I/O 261

File Class 262

Directories in Java 272

Listing Directories 273

1. Java – Exceptions 275

Exception Hierarchy 276

Built-in Exceptions 277

Exceptions Methods 279

Catching Exceptions 280

Multiple Catch Blocks 281

Catching Multiple Type of Exceptions 282

The Throws/Throw Keywords 282

The Finally Block 283

The try-with-resources 285

User-defined Exceptions 287

Common Exceptions 290

1. Java – Inner Classes 291

Nested Classes 291

Inner Classes (Non-static Nested Classes) 292

Accessing the Private Members 293

Method-local Inner Class 294

Anonymous Inner Class 295

Anonymous Inner Class as Argument 296

Static Nested Class 297

JAVA - OBJECT ORIENTED 299

1. Java – Inheritance 300

extends Keyword 300

Sample Code 300

The super keyword 302

Invoking Superclass Constructor 305

IS-A Relationship 306

The instanceof Keyword 308

HAS-A relationship 309

Types of Inheritance 309

1. Java – Overriding 311

Rules for Method Overriding 313

Using the super Keyword 314

1. Java – Polymorphism 315

Virtual Methods 316

1. Java – Abstraction 320

Abstract Class 320

Inheriting the Abstract Class 323

Abstract Methods 324

1. Java – Encapsulation 326

Benefits of Encapsulation 328

1. Java – Interfaces 329

Declaring Interfaces 330

Implementing Interfaces 330

Extending Interfaces 332

Extending Multiple Interfaces 333

Tagging Interfaces 333

1. Java – Packages 334

Creating a Package 334

The import Keyword 336

The Directory Structure of Packages 337

Set CLASSPATH System Variable 339

JAVA – ADVANCED 340

1. Java – Data Structures 341

The Enumeration 341

The BitSet 343

The Vector 348

The Stack 355

The Dictionary 358

The Hashtable 362

The Properties 366

1. Java – Collections Framework 370

The Collection Interfaces 371

The Collection Interface 372

The List Interface 375

The Set Interface 378

The SortedSet Interface 380

The Map Interface 382

The Map.Entry Interface 384

The SortedMap Interface 386

The Enumeration Interface 388

The Collection Classes 389

The LinkedList Class 391

The ArrayList Class 395

The HashSet Class 399

The LinkedHashSet Class 402

The TreeSet Class 403

The HashMap Class 406

The TreeMap Class 409

The WeakHashMap Class 412

The LinkedHashMap Class 415

The IdentityHashMap Class 418

The Vector Class 422

The Stack Class 428

The Dictionary Class 430

The Map Interface 431

The Hashtable Class 433

The Properties Class 437

The BitSet Class 440

The Collection Algorithms 445

How to Use an Iterator ? 450

How to Use a Comparator ? 454

Summary 456

1. Java – Generics 457

Generic Methods 457

Bounded Type Parameters 459

Generic Classes 460

1. Java – Serialization 462

Serializing an Object 463

Deserializing an Object 464

1. Java – Networking 466

URL Processing 466

URL Class Methods 467

URLConnections Class Methods 470

Socket Programming 473

ServerSocket Class Methods 474

Socket Class Methods 475

InetAddress Class Methods 477

Socket Client Example 477

Socket Server Example 479

1. Java – Sending E-mail 481

Send a Simple E-mail 481

Send an HTML E-mail 483

Send Attachment in E-mail 485

User Authentication Part 487

1. Java – Multithreading 488

Life Cycle of a Thread 488

Thread Priorities 489

Create a Thread by Implementing a Runnable Interface 489

Create a Thread by Extending a Thread Class 491

Thread Methods 494

Major Java Multithreading Concepts 499

Thread Synchronization 499

Interthread Communication 504

Thread Deadlock 507

Thread Control 510

1. Java – Applet Basics 515

Life Cycle of an Applet 515

A "Hello, World" Applet 516

The Applet Class 516

Invoking an Applet 517

HTML <applet> Tag 517

HTML Attribute Reference 519

HTML Events Reference 521

Getting Applet Parameters 526

Specifying Applet Parameters 527

Application Conversion to Applets 527

Event Handling 528

Displaying Images 530

Playing Audio 532

1. Java – Documentation Comments 534

What is Javadoc? 534

The javadoc Tags 535

# Java – Basics

Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]).

Java

1. Java – Overview

The latest release of the Java Standard Edition is Java SE 8. With the advancement of Java and its widespread popularity, multiple configurations were built to suit various types of platforms. For example: J2EE for Enterprise Applications, J2ME for Mobile Applications.

The new J2 versions were renamed as Java SE, Java EE, and Java ME respectively. Java is guaranteed to be **Write Once, Run Anywhere**.

Java is:

* + **Object Oriented:** In Java, everything is an Object. Java can be easily extended since it is based on the Object model.
  + **Platform Independent:** Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
  + **Simple:** Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
  + **Secure:** With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
  + **Architecture-neutral:** Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
  + **Portable:** Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.
  + **Robust:** Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.
  + **Multithreaded:** With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
  + **Interpreted:** Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
  + **High Performance:** With the use of Just-In-Time compilers, Java enables high performance.
  + **Distributed:** Java is designed for the distributed environment of the internet.
  + **Dynamic:** Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

## History of Java

James Gosling initiated Java language project in June 1991 for use in one of his many set- top box projects. The language, initially called ‘Oak’ after an oak tree that stood outside Gosling's office, also went by the name ‘Green’ and ended up later being renamed as Java, from a list of random words.

Sun released the first public implementation as Java 1.0 in 1995. It promised **Write Once, Run Anywhere** (WORA), providing no-cost run-times on popular platforms.

On 13 November, 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL).

On 8 May, 2007, Sun finished the process, making all of Java's core code free and open- source, aside from a small portion of code to which Sun did not hold the copyright.

## Tools You Will Need

For performing the examples discussed in this tutorial, you will need a Pentium 200-MHz computer with a minimum of 64 MB of RAM (128 MB of RAM recommended).

You will also need the following softwares:

* + Linux 7.1 or Windows xp/7/8 operating system
  + Java JDK 8
  + Microsoft Notepad or any other text editor

This tutorial will provide the necessary skills to create GUI, networking, and web applications using Java.

## Try It Option

We have provided you with an option to compile and execute available code online. Just click the **Try it** button avaiable at the top-right corner of the code window to compile and execute the available code. There are certain examples which cannot be executed online, so we have skipped those examples.

public class MyFirstJavaProgram {

public static void main(String []args) { System.out.println("Hello World");

}

}

There may be a case that you do not see the result of the compiled/executed code. In such case, you can re-try to compile and execute the code using **execute** button available in the compilation pop-up window.

## What is Next?

The next chapter will guide you to how you can obtain Java and its documentation. Finally, it instructs you on how to install Java and prepare an environment to develop Java applications.

In this chapter, we will discuss on the different aspects of setting up a congenial environment for Java.

Java

2. Java - Environment Setup

## Try it Option Online

You really do not need to set up your own environment to start learning Java programming language. Reason is very simple, we already have Java Programming environment setup online, so that you can compile and execute all the available examples online at the same time when you are doing your theory work. This gives you confidence in what you are reading and to check the result with different options. Feel free to modify any example and execute it online.

Try the following example using **Try it** option available at the top right corner of the following sample code box:

public class MyFirstJavaProgram {

public static void main(String []args) { System.out.println("Hello World");

}

}

For most of the examples given in this tutorial, you will find the **Try it** option, which you can use to execute your programs and enjoy your learning.

## Local Environment Setup

If you are still willing to set up your environment for Java programming language, then this section guides you on how to download and set up Java on your machine. Following are the steps to set up the environment.

Java SE is freely available from the link [Download Java](http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html). You can download a version based on your operating system.

Follow the instructions to download Java and run the **.exe** to install Java on your machine. Once you installed Java on your machine, you will need to set environment variables to point to correct installation directories:

### Setting Up the Path for Windows

Assuming you have installed Java in *c:\Program Files\java\jdk* directory:

* + Right-click on 'My Computer' and select 'Properties'.
  + Click the 'Environment variables' button under the 'Advanced' tab.

Java

* + Now, alter the 'Path' variable so that it also contains the path to the Java executable. Example, if the path is currently set to 'C:\WINDOWS\SYSTEM32', then change your path to read 'C:\WINDOWS\SYSTEM32;c:\Program Files\java\jdk\bin'.

### Setting Up the Path for Linux, UNIX, Solaris, FreeBSD

Environment variable PATH should be set to point to where the Java binaries have been installed. Refer to your shell documentation, if you have trouble doing this.

Example, if you use ***bash*** as your shell, then you would add the following line to the end of your **'.bashrc: export PATH=/path/to/java:$PATH'**

## Popular Java Editors

To write your Java programs, you will need a text editor. There are even more sophisticated IDEs available in the market. But for now, you can consider one of the following:

* + **Notepad:** On Windows machine, you can use any simple text editor like Notepad (Recommended for this tutorial), TextPad.
  + **Netbeans:** A Java IDE that is open-source and free, which can be downloaded from <http://www.netbeans.org/index.html>.
  + **Eclipse:** A Java IDE developed by the eclipse open-source community and can be downloaded from <http://www.eclipse.org/>.

## What is Next?

Next chapter will teach you how to write and run your first Java program and some of the important basic syntaxes in Java needed for developing applications.

When we consider a Java program, it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into what do class, object, methods, and instance variables mean.

Java

3. Java – Basic Syntax

* + **Object -** Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behavior such as wagging their tail, barking, eating. An object is an instance of a class.
  + **Class -** A class can be defined as a template/blueprint that describes the behavior/state that the object of its type supports.
  + **Methods -** A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
  + **Instance Variables -** Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

## First Java Program

Let us look at a simple code that will print the words ***Hello World***.

public class MyFirstJavaProgram {

/\* This is my first java program.

\* This will print 'Hello World' as the output

\*/

public static void main(String []args) { System.out.println("Hello World"); // prints Hello World

}

}

Let's look at how to save the file, compile, and run the program. Please follow the subsequent steps:

* + Open notepad and add the code as above.
  + Save the file as: MyFirstJavaProgram.java.
  + Open a command prompt window and go to the directory where you saved the class. Assume it's C:\.
  + Type 'javac MyFirstJavaProgram.java' and press enter to compile your code. If there are no errors in your code, the command prompt will take you to the next line (Assumption : The path variable is set).
  + Now, type ' java MyFirstJavaProgram ' to run your program.
  + You will be able to see ' Hello World ' printed on the window.

C:\> javac MyFirstJavaProgram.java C:\> java MyFirstJavaProgram Hello World

## Basic Syntax

About Java programs, it is very important to keep in mind the following points.

* + **Case Sensitivity -** Java is case sensitive, which means identifier **Hello**and **hello** would have different meaning in Java.
  + **Class Names -** For all class names the first letter should be in Upper Case.

If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.

**Example**: *class MyFirstJavaClass*

* + **Method Names -** All method names should start with a Lower Case letter.

If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.

**Example**: *public void myMethodName()*

* + **Program File Name -** Name of the program file should exactly match the class name.

When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match, your program will not compile).

**Example**: Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as *'MyFirstJavaProgram.java'*

* + **public static void main(String args[]) -** Java program processing starts from the main() method which is a mandatory part of every Java program.

## Java Identifiers

All Java components require names. Names used for classes, variables, and methods are called **identifiers**.

In Java, there are several points to remember about identifiers. They are as follows:

* + All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (\_).
  + After the first character, identifiers can have any combination of characters.
  + A key word cannot be used as an identifier.
  + Most importantly, identifiers are case sensitive.
  + Examples of legal identifiers: age, $salary, \_value, 1\_value.
  + Examples of illegal identifiers: 123abc, -salary.

## Java Modifiers

Like other languages, it is possible to modify classes, methods, etc., by using modifiers. There are two categories of modifiers:

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

We will be looking into more details about modifiers in the next section.

## Java Variables

Following are the types of variables in Java:

* + Local Variables
  + Class Variables (Static Variables)
  + Instance Variables (Non-static Variables)

## Java Arrays

Arrays are objects that store multiple variables of the same type. However, an array itself is an object on the heap. We will look into how to declare, construct, and initialize in the upcoming chapters.

## Java Enums

Enums were introduced in Java 5.0. Enums restrict a variable to have one of only a few predefined values. The values in this enumerated list are called enums.

With the use of enums it is possible to reduce the number of bugs in your code.

For example, if we consider an application for a fresh juice shop, it would be possible to restrict the glass size to small, medium, and large. This would make sure that it would not allow anyone to order any size other than small, medium, or large.

### Example

class FreshJuice {

enum FreshJuiceSize{ SMALL, MEDIUM, LARGE } FreshJuiceSize size;

}

public class FreshJuiceTest {

public static void main(String args[]){ FreshJuice juice = new FreshJuice();

juice.size = FreshJuice.FreshJuiceSize.MEDIUM ; System.out.println("Size: " + juice.size);

}

}

The above example will produce the following result:

Size: MEDIUM

**Note:** Enums can be declared as their own or inside a class. Methods, variables, constructors can be defined inside enums as well.

## Java Keywords

The following list shows the reserved words in Java. These reserved words may not be used as constant or variable or any other identifier names.

|  |  |  |  |
| --- | --- | --- | --- |
| abstract | assert | boolean | break |
| byte | case | catch | char |

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

## Comments in Java

Java supports single-line and multi-line comments very similar to C and C++. All characters available inside any comment are ignored by Java compiler.

public class MyFirstJavaProgram{

/\* This is my first java program.

* This will print 'Hello World' as the output
* This is an example of multi-line comments.

\*/

public static void main(String []args){

// This is an example of single line comment

/\* This is also an example of single line comment. \*/ System.out.println("Hello World");

}

}

## Using Blank Lines

A line containing only white space, possibly with a comment, is known as a blank line, and Java totally ignores it.

## Inheritance

In Java, classes can be derived from classes. Basically, if you need to create a new class and here is already a class that has some of the code you require, then it is possible to derive your new class from the already existing code.

This concept allows you to reuse the fields and methods of the existing class without having to rewrite the code in a new class. In this scenario, the existing class is called the **superclass** and the derived class is called the **subclass**.

## Interfaces

In Java language, an interface can be defined as a contract between objects on how to communicate with each other. Interfaces play a vital role when it comes to the concept of inheritance.

An interface defines the methods, a deriving class (subclass) should use. But the implementation of the methods is totally up to the subclass.

## What is Next?

The next section explains about Objects and classes in Java programming. At the end of the session, you will be able to get a clear picture as to what are objects and what are classes in Java.

Java is an Object-Oriented Language. As a language that has the Object-Oriented feature, Java supports the following fundamental concepts:

Java

4. Java – Objects & Classes

* + Polymorphism
  + Inheritance
  + Encapsulation
  + Abstraction
  + Classes
  + Objects
  + Instance
  + Method
  + Message Parsing

In this chapter, we will look into the concepts - Classes and Objects.

* + **Object -** Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating. An object is an instance of a class.
  + **Class -** A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.

## Objects in Java

Let us now look deep into what are objects. If we consider the real-world, we can find many objects around us, cars, dogs, humans, etc. All these objects have a state and a behavior.

If we consider a dog, then its state is - name, breed, color, and the behavior is - barking, wagging the tail, running.

If you compare the software object with a real-world object, they have very similar characteristics.

Software objects also have a state and a behavior. A software object's state is stored in fields and behavior is shown via methods.

So in software development, methods operate on the internal state of an object and the object-to-object communication is done via methods.

## Classes in Java

A class is a blueprint from which individual objects are created. Following is a sample of a class.

public class Dog{ String breed; int ageC String color;

void barking(){

}

void hungry(){

}

void sleeping(){

}

}

A class can contain any of the following variable types.

* + **Local variables:** Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.
  + **Instance variables:** Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
  + **Class variables:** Class variables are variables declared within a class, outside any method, with the static keyword.

A class can have any number of methods to access the value of various kinds of methods. In the above example, barking(), hungry() and sleeping() are methods.

Following are some of the important topics that need to be discussed when looking into classes of the Java Language.

## Constructors

When discussing about classes, one of the most important sub topic would be constructors. Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.

Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

Following is an example of a constructor:

public class Puppy{ public Puppy(){

}

public Puppy(String name){

// This constructor has one parameter, name.

}

}

Java also supports [Singleton Classes](http://www.tutorialspoint.com/java/java_using_singleton.htm) where you would be able to create only one instance of a class.

**Note:** We have two different types of constructors. We are going to discuss constructors in detail in the subsequent chapters.

## How to Use Singleton Class?

The Singleton's purpose is to control object creation, limiting the number of objects to only one. Since there is only one Singleton instance, any instance fields of a Singleton will occur only once per class, just like static fields. Singletons often control access to resources, such as database connections or sockets.

For example, if you have a license for only one connection for your database or your JDBC driver has trouble with multithreading, the Singleton makes sure that only one connection is made or that only one thread can access the connection at a time.

### Implementing Singletons

**Example 1**

The easiest implementation consists of a private constructor and a field to hold its result, and a static accessor method with a name like getInstance().

The private field can be assigned from within a static initializer block or, more simply, using an initializer. The getInstance( ) method (which must be public) then simply returns this instance −

// File Name: Singleton.java public class Singleton {

private static Singleton singleton = new Singleton( );

/\* A private Constructor prevents any other

\* class from instantiating.

\*/

private Singleton(){ }

/\* Static 'instance' method \*/

public static Singleton getInstance( ) { return singleton;

}

/\* Other methods protected by singleton-ness \*/ protected static void demoMethod( ) {

System.out.println("demoMethod for singleton");

}

}

Here is the main program file, where we will create a singleton object:

// File Name: SingletonDemo.java public class SingletonDemo {

public static void main(String[] args) { Singleton tmp = Singleton.getInstance( ); tmp.demoMethod( );

}

}

This will produce the following result −

demoMethod for singleton

### Example 2

Following implementation shows a classic Singleton design pattern:

public class ClassicSingleton {

private static ClassicSingleton instance = null; private ClassicSingleton() {

// Exists only to defeat instantiation.

}

public static ClassicSingleton getInstance() { if(instance == null) {

instance = new ClassicSingleton();

}

return instance;

} }

The ClassicSingleton class maintains a static reference to the lone singleton instance and returns that reference from the static getInstance() method.

Here, ClassicSingleton class employs a technique known as lazy instantiation to create the singleton; as a result, the singleton instance is not created until the getInstance() method is called for the first time. This technique ensures that singleton instances are created only when needed.

## Creating an Object

As mentioned previously, a class provides the blueprints for objects. So basically, an object is created from a class. In Java, the new keyword is used to create new objects.

There are three steps when creating an object from a class:

* + **Declaration:** A variable declaration with a variable name with an object type.
  + **Instantiation:** The 'new' keyword is used to create the object.
  + **Initialization:** The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Following is an example of creating an object:

public class Puppy{

public Puppy(String name){

// This constructor has one parameter, name. System.out.println("Passed Name is :" + name );

}

public static void main(String []args){

// Following statement would create an object myPuppy Puppy myPuppy = new Puppy( "tommy" );

}

}

If we compile and run the above program, then it will produce the following result:

Passed Name is :tommy

## Accessing Instance Variables and Methods

Instance variables and methods are accessed via created objects. To access an instance variable, following is the fully qualified path:

/\* First create an object \*/ ObjectReference = new Constructor();

/\* Now call a variable as follows \*/ ObjectReference.variableName;

/\* Now you can call a class method as follows \*/ ObjectReference.MethodName();

### Example

This example explains how to access instance variables and methods of a class.

public class Puppy{

int puppyAge;

public Puppy(String name){

// This constructor has one parameter, name. System.out.println("Name chosen is :" + name );

}

public void setAge( int age ){ puppyAge = age;

}

public int getAge( ){

System.out.println("Puppy's age is :" + puppyAge ); return puppyAge;

}

public static void main(String []args){

/\* Object creation \*/

Puppy myPuppy = new Puppy( "tommy" );

/\* Call class method to set puppy's age \*/ myPuppy.setAge( 2 );

/\* Call another class method to get puppy's age \*/ myPuppy.getAge( );

/\* You can access instance variable as follows as well \*/ System.out.println("Variable Value :" + myPuppy.puppyAge );

}

}

If we compile and run the above program, then it will produce the following result:

Name chosen is :tommy Puppy's age is :2 Variable Value :2

## Source File Declaration Rules

As the last part of this section, let's now look into the source file declaration rules. These rules are essential when declaring classes, *import* statements and *package* statements in a source file.

* + There can be only one public class per source file.
  + A source file can have multiple non-public classes.
  + The public class name should be the name of the source file as well which should be appended by **.java** at the end. For example: the class name is *public class Employee{}* then the source file should be as Employee.java.
  + If the class is defined inside a package, then the package statement should be the first statement in the source file.
  + If import statements are present, then they must be written between the package statement and the class declaration. If there are no package statements, then the import statement should be the first line in the source file.
  + Import and package statements will imply to all the classes present in the source file. It is not possible to declare different import and/or package statements to different classes in the source file.

Classes have several access levels and there are different types of classes; abstract classes, final classes, etc. We will be explaining about all these in the access modifiers chapter.

Apart from the above mentioned types of classes, Java also has some special classes called Inner classes and Anonymous classes.

End of ebook preview

If you liked what you saw…

Buy it from our store @ **https://store.tutorialspoint.com**