Session 1 – Java Features

OBJECTIVES

After completing this session you will be able to understand:

- ✓ Introduction to Java
- ✓ Features of Java
- ✓ Platform Neutrality

Java Features

Introduction to Java

Java is a simple and yet powerful object oriented programming language and it is in many respects similar to C++. It was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems, Inc. originally called OAK, Initially it was developed for their internal purpose. Later on it was renamed as the Java programming language in 1995 with some additional favor and given to web world. It was developed to provide a platform-independent programming language.Java supports web applications as the remaining languages which are popular in that time are cannot.

To date, the Java platform has attracted more than 6.5 million software developers. It's used in every major industry segment and has a presence in a wide range of devices, computers, and networks.

With the advancement of Java and its wide spread popularity, multiple configurations were built to suite various types of platforms. Ex: J2EE for Enterprise Applications, J2ME for Mobile Applications.

Sun Microsystems has renamed the new J2 versions as Java SE, Java EE and Java ME respectively. Java is guaranteed to be Write Once, Run Anywhere.

The Java Programming Language –

The Java Programming Language is a high-level language that we can categorize under the following striking keywordshaving some powerful features which makes it ever green:

Why Java so popular?

Java has some powerful features which makes it ever green.

- ➢ Platform independent
- ➢ Multi-threading
- ➢ Supports web enabled applications.
- ➢ Object oriented.
- ➢ More secured.

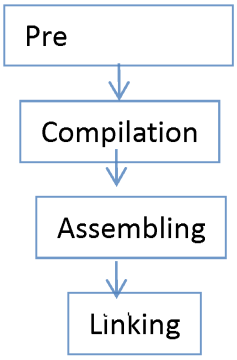
Platform Neutrality

Java is a platform-neutral language because java programs can run on any operating system with the support of JVM.A platform independent language is the one, which can be executed in any other platform without compilation and modifications on the compiled code. Java is Platform independent language. That means we can compile the java program in any OS. Then the compiled (.class file) code can be executed anywhere without compilation.

Platform Dependency:

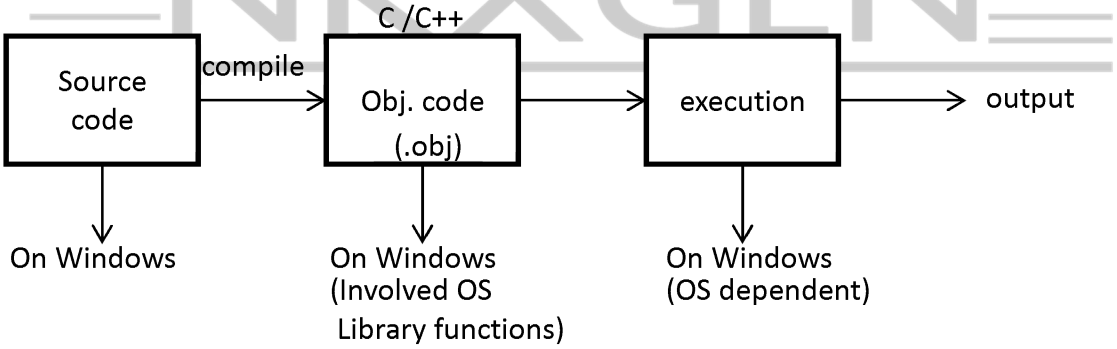
Before discussing how java is platform independent, we will discuss C/C++ compilation steps. As C/C++ are platform dependent languages.As we have already discussed, C/C++ is platform dependent languages.

The compilation steps for C/C++:



To start off, preprocessing replaces certain pieces of text by other text according to a system of macros. Next, compilation translates the source program into assembly instructions, which are then converted to machine instructions. Finally, the linking process establishes a connection to the operating system for primitives. This includes adding the runtime library, which mainly consists of memory management routines.

So when at the time of .obj file formation it has OS Library functions. As the API functionalities are different from one OS to another OS, this .obj cannot be read by other OS. That's why C/C++ is platform dependent.



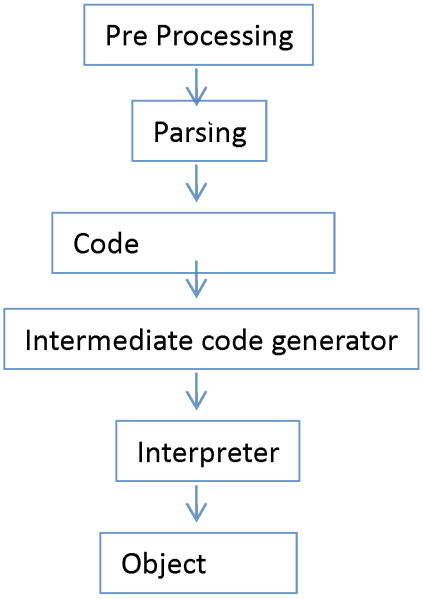
Java Neutrality / Java Platform Independency:

Java requires each class to be placed in its own source file, named with the same name as the class name if it is public specification and added suffix `.java`. This basically forces any medium sized program to be split in several source files. When compiling source code, each class is placed in its own `.class` file that contains the byte code.After compiling all source files, the result will be at least as much class files as the sources, which will combine to form your Java program. This is where the class loader comes into picture along with the byte code verifier - two unique steps that distinguish Java from languages like C/C++. The class loader is responsible for loading each class' bytecode. Java provides developers with the opportunity to write their own class loader, which gives developers great flexibility. When a class is

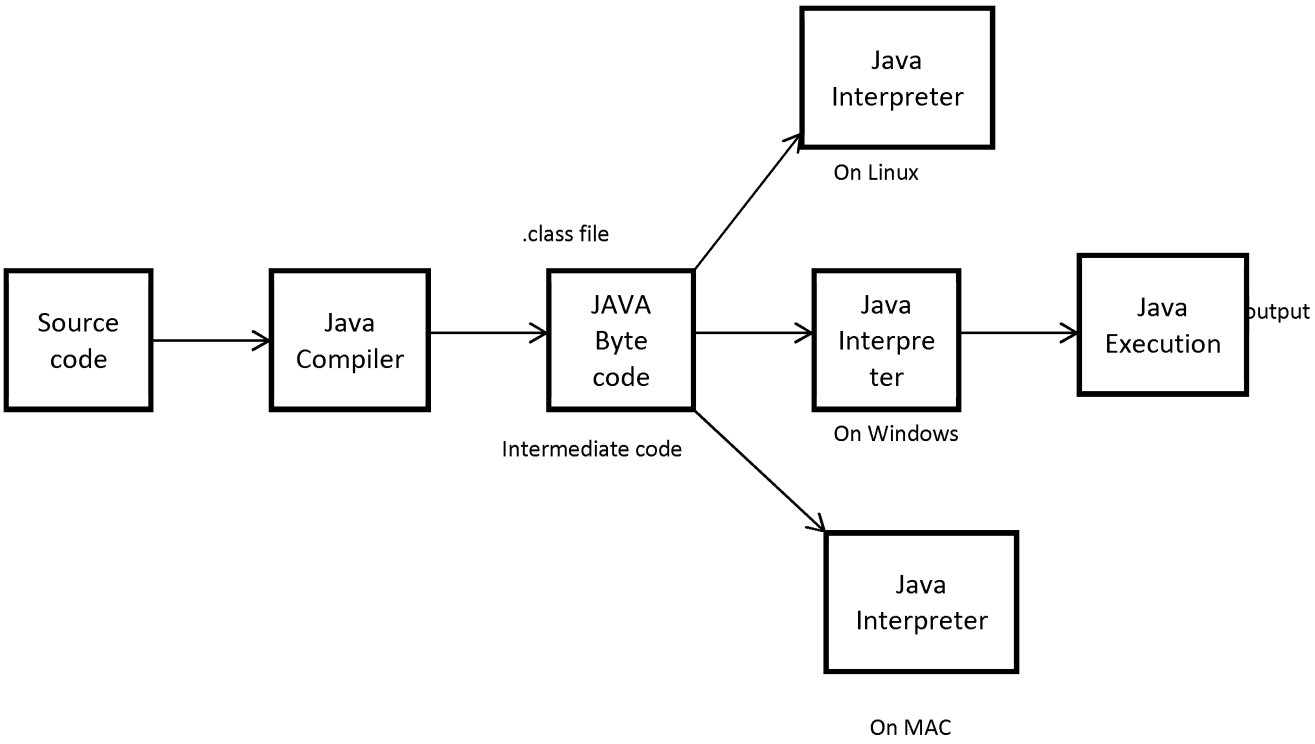
needed by the JVM the `loadClass(String name, Boolean resolve);` method is called passing the class name to be loaded. Once it finds the file that contains the bytecode for the class, it is read into memory

and passed to the `defineClass`. If the class is not found by the loader, it can delegate the loading to a parent class loader or try to use `findSystemClass` to load the class from local filesystem.

Hint: Java is Platform Independent



Unlike languages like C /C++ java has intermediate code generator. Here we will get .class file, which is in byte code. This byte code is nowhere related to OS library. Why because all the steps before intermediate code generator, can be done by Java compiler. In parsing syntax, semantic errors and early binding errors will be checked. For code optimization also we need not go for system Libraries. So up to here OS libraries are nowhere used. That's why it can be executed on any operating system.



Hint: Availability of Java interpreter is everywhere.

Some other Interesting features:

- **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 virtual Machine (JVM) on whichever platform it is being run.
- **Simple**: Java is designed to be easy to learn. If you understand the basic concept of OOP java 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.
- **Architectural- neutral**: Java compiler generates an architecture-neutral object file format which makes the compiled code to be executable on many processors, with the presence Java runtime system.
- **Portable**: being architectural neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler and 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.
- **Multi-threaded**: With Java's multi-threaded feature it is possible to write programs that can do many tasks simultaneously? This design feature allows developers to construct smoothly running interactive applications.
- **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.

Session 2 – Applications Overview

OBJECTIVES

After completing this session you will be able to understand:

- ✓ Software Applications
- ✓ Types of software Applications
  - Windows Applications
  - Web Applications
  - Compare Windows vs. Web

Introduction to Software Applications

Introduction

To begin with, let us have an overview on different types of software applications we use in real time scenarios. As we learn the language to develop software applications we need to have a thorough

understanding on the application models. Broadly the types of applications can be categorized based on application model in the following way:

- ✓ Console Applications (Outdated)
- ✓ Desktop/windows Applications
- ✓ Web Applications
- ✓ Web Services Applications(SOA)
- ✓ Enterprise Applications
- ✓ Embedded Applications
- ✓ Mobile Applications

  And many more

Let us understand the Windows-based Applications & Web-based Applications – which are very commonly used and worked

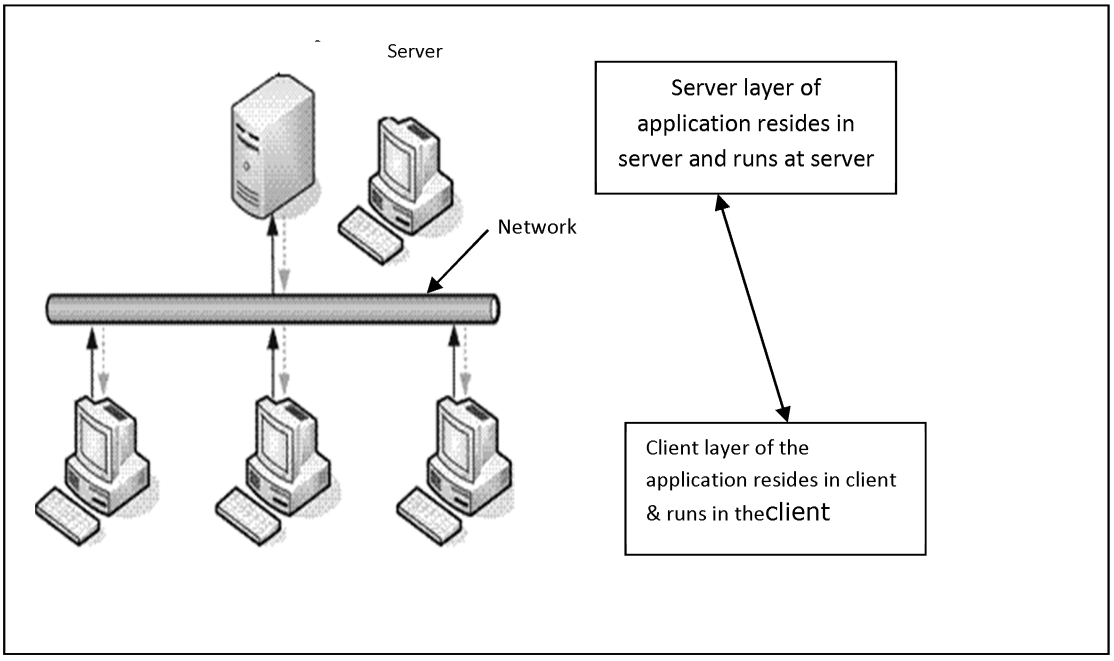
Desktop/Windows Applications

Windows-based applications, sometimes called **Thick Client applications**, are the more traditional way of doing things. They often make more sense in an **Intranet** setting because they can offer a more robust Graphical User Interface (GUI) and better features that increase the productivity. Simple things like keyboard shortcuts, client side caching, multi-threading, and advanced user interface controls can make all the difference in the usability of an application. In windows applications, the "server" side often has a "middle layer" in between the client and the database; this is called a "3-tier" solution. Sometimes there are multiple middle layers which interact, and this is called an "n-tier" solution.

Disadvantages of Windows Applications

- Installed in Systems (both Client & in Server)
- Separate Software engineers required to run the application
- Developed with many languages and technologies
- They are typically deployed on an intranet or extranet, with access restricted by passwords or other methods.

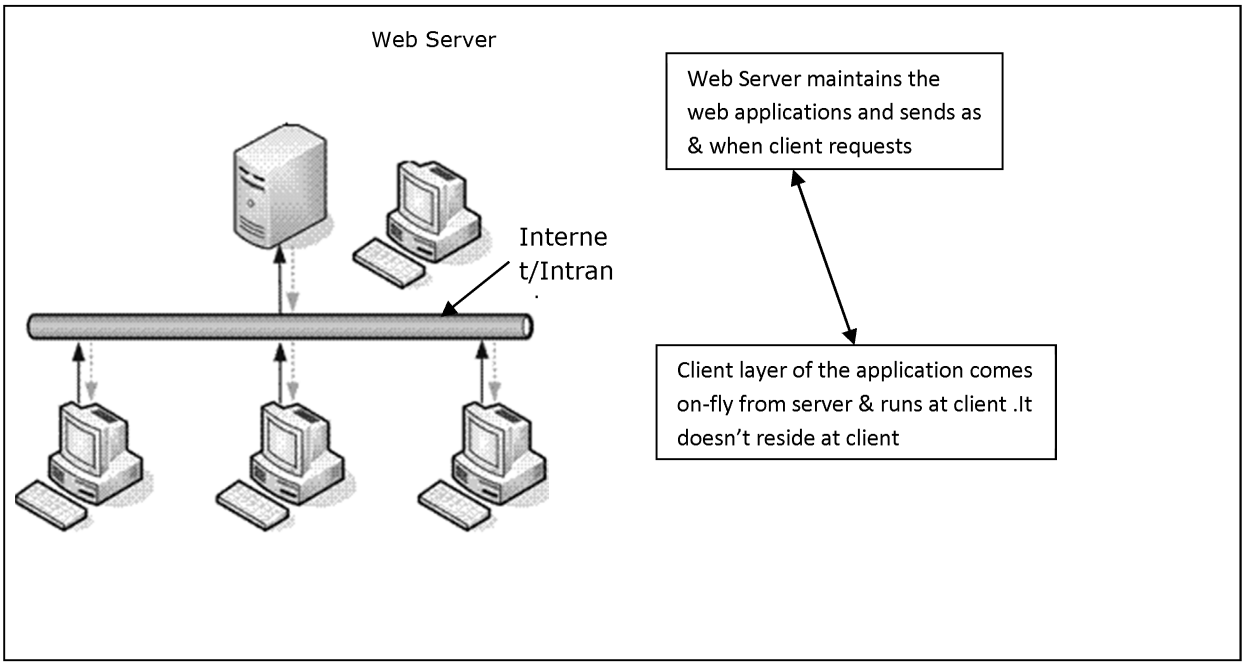
Windows Application [Thick Clients]



Web Applications

Web applications that utilize the Web browser for security, state management, and script execution (run-time). Most data processing and storage occurs on a remote server and not a user's local machine. Server request and response mostly occurs through the http protocol.

Web Application [Thin Clients]



- Applications are installed on servers (mostly in web server)
- Will not be installed in client
- Will be loaded to client on fly as and when the request is made
- Client doesn't require any additional software to run

Thin Client

The primary benefits of thin clients are of wide reach

- ✓ Accessible by anyone with a Web browser,
- ✓ Open development platform (built on popular open standards),
- ✓ No footprint (quick download, no artifact on user machines beyond browser cookies), and Deployment/manageability (distributed and maintained from a central source).

Thin Client Solutions:

- ✓ HTML, XHTML
- ✓ HTML, JavaScript, and CSS (DHTML)
- ✓ DHTML with Remote Scripting via iFrame
- ✓ DHTML with XMLHttpRequest (AJAX)

Desktop Applications vs. Web Applications

Sometimes there are thin-clients and thick clients interacting with the same back-end systems.
Larger systems will sometimes have a combination of thin-client and thick- client front-ends which interact with various bits of "middleware" that in turn interact with one or more databases.

Why Web applications are advantageous?

Easy in Installation

- ✓ Once it is deployed in Server it will be automatically available
- ✓ No Additional software need to be installed. Just OS and browser will be enough

Easy in Maintenance

- ✓ Applications require frequent modifications. Each time when application is modified and deployed in server, it will be available to the clients from the next moment

Cross Platform

- ✓ Irrespective of the technology used the web applications are invoke able from any operating system and from any browser

Session 3 – Java Environment

OBJECTIVES

After completing this session you will be able to understand:

- ✓ Java History
- ✓ Frameworks
- ✓ Java Framework
- ✓ Java SE Overview
- ✓ Java EE Overview

Java Environment

Java History

Java is a programming language originally developed by **James Gosling** at **Sun Microsystems** (which is now a subsidiary of **Oracle Corporation**) and **released in 1995** as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. It is intended to let application developers "**write once, run anywhere**". Java is considered by many as one of the most influential programming languages of the 20th century, and is widely used from application software to web applications.

James Gosling initiated the Java language project in June 1991 for use in one of his much 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 renamed as *Java*, from a list of random words. Gosling aimed to implement a virtual machine and a language that had a familiar C/C++ style of notation. Sun Microsystems 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. Fairly secure and featuring configurable security, it allowed network- and file-access restrictions. Major web browsers soon incorporated the ability to run Java *applets* within web pages, and Java quickly became popular. With the advent of *Java 2* (released initially as J2SE 1.2 in December 1998), new versions had multiple configurations built for different types of platforms.

Java Versions

Java Version	**Description**
JDK 1.0 (January 23, 1996)	**Codename** Oak Initial release. The first stable version was the JDK 1.0.2. is called Java 1.
JDK 1.1 (February 19, 1997)	**Major additions included:** AWT event model, inner classes, JavaBeans, JDBC, Reflection.
J2SE 1.2 (December 8, 1998)	**Codename Playground** Swing graphical API was integrated, Sun's JVM was equipped with a JIT compiler for the first time, Collections framework
J2SE 1.3 (May 8, 2000)	**Codename Kestrel** Java Naming and Directory Interface (JNDI) included in core libraries (previously available as an extension)
J2SE 1.4 (February 6, 2002)	**Codename Merlin** assert keyword
J2SE 5.0 (September 30, 2004)	Codename Tiger. Generics, Metadata, Autoboxing/ Unboxing, Enumerations, Enhanced for each loop, Static imports, Scanner class.
Java SE 6 (December 11, 2006)	**Codename Mustang** JDBC 4.0 support (JSR 221)
Java SE 7 (July 07, 2011)	**Codename is *Dolphin*** Strings in switch, Automatic resource management in try-statement, Improved type inference for generic instance creation, Allowing underscores in numeric literals.

Frameworks

Framework is set of reusable software program that forms the basis for an application. Frameworks help the programmers to build the application quickly. Earlier it was very hard to develop complex web

applications. Now it is very easy to develop such application using different kinds of frameworks such as Struts, Struts 2, Hibernate, JSF, Tapestry, JUnit, Log4j, Spring etc.

A framework will often dictate the structure of your application. Some frameworks even supply so much code that you have to do very little to write your application. This can be good or bad, depending on how easy it is to use.

Frameworks are the substance of programming. You build on top of a good one; your program is solid and fast and comes together beautifully. You build on top of a bad one; your life is miserable, brutish, and short.

Java Framework

In Java technology there are so many frameworks that help the programmers to build complex applications easily. You can choose these frameworks for building your applications.

Very common examples are GUI frameworks, ex: Java's Swing and AWT classes. They have a huge amount of code to manage the user interface, and there is *inversion of control* because you start the GUI framework and then wait for it to call your listeners.

The Collections classes are sometimes called a framework, perhaps largely because of the size and complexity. But it is more properly referred to as a library because there is no inversion of control -- your program simply calls methods in these predefined or extended classes.

Java SE Overview

Java, Standard Edition also called Java SE is commonly used in Java programming. This Java Platform is instrumental in implementing portable applications that are in common practice. In other words, Java standard edition comprise of a virtual machine, which is put in practice for running Java programs in conjunctions with a set of libraries required to use different applications.

Before getting its present name Java Standard Edition 6.0, this platform has gone under various names like java 2 Platform, Standard Edition or J2SE. Java SE is similar with Java EE(enterprise edition in a number of applications). Java standard edition (Java SE) allows you to develop and implement Java applications on desktops and servers. In addition, it offers rich user interface, competence, adaptability, portability and last but not the least security that is a prerequisite besides all other applications. In July, 2011, the Java Standard Editions of 7.0 will be release.

Java EE Overview

Using the Java Platform, Enterprise Edition (Java EE) architecture, you can build distributed Web and enterprise applications. This architecture helps you focus on presentation and application issues, rather than on systems issues.

You can use the Java EE tools and features to create applications that are structured around modules with different purposes, such as Web sites and Enterprise Java beans (EJB) applications. When you use EJB 3.0 components, you can create a distributed, secure application with transactional support. When you develop applications that access persistent data, you can use the new Java Persistence API (JPA). This standard simplifies the creation and use of persistent entities, as well as adding new features. For developing presentation logic, you can use technologies such as Java Server Pages (JSP) or Java Server Faces (JSF).

Using the Java EE Platform Enterprise Edition (Java EE), you can develop applications more quickly and conveniently than in previous versions. Java EE significantly enhances ease of use providing-

- ✓ Reduced development time
- ✓ Reduced application complexity
- ✓ Improved application performance

Session 4 – Java Development Environment

OBJECTIVES

After completing this session you will be able to understand:

- ✓ Simple Java Program
- ✓ Java Development Environment
- ✓ Java IDE's
- ✓ JDK 1.7
- ✓ Runtime Environment
- ✓ JVM Architecture
- ✓ Steps to Execute Program

Java Development Environment

Simple Java Program

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

```
/*
This is a simple Java program.
Call this file as "SimpleProg.java".
*/
class SimpleProg{
        // Your program begins with a call to main().
        public static void main(String args[]) {
                //prints the statement
                System.out.println("This is a simple Java program.");
        }
}
```


Explanation:

Although SimpleProg.java is quite short, it includes several key features that are common toall Java programs. Let's closely examine each part of the program.

The program begins with the following lines:

/*

This is a simple Java program.

Call this file "Example.java".

*/

This is a *comment.* Like most other programming languages, Java lets you enter a remark intoa program's source file. The contents of a comment are ignored by the compiler. Instead, acomment describes or explains the operation of the program to anyone who is reading itssource code. In this case, the comment describes the program and reminds you that the sourcefile should be called Example.java.

Java supports three styles of comments. The one shown at the top of the program is calleda *multiline comment.* This type of comment must begin with /* and end with */. Anythingbetween these two comment symbols is ignored by the compiler. As the name suggests, amultiline comment may be several lines long.

The next line of code in the program is shown here:

class Example {

This line uses the keyword class to declare that a new class is being defined. Example is an*identifier* that is the name of the class. The entire class definition, including all of its members,will be between the opening curly brace ({) and the closing curly brace (}). For the moment,don't worry too much about the details of a

class except to note that in Java, all programactivity occurs within one. This is one reason why all Java programs are (at least a little bit)object-oriented.

The next line in the program is the *single-line comment,* shown here:

// Your program begins with a call to main().

This is the second type of comment supported by Java. A *single-line comment* begins witha // and ends at the end of the line. As a general rule, programmers use multiline commentsfor longer remarks and single-line comments for brief, line-by-line descriptions. The thirdtype of comment, a *documentation comment,* will be discussed in the "Comments" section laterin this chapter.

The next line of code is shown here:

public static void main(String args[]) {

This line begins the main( ) method. As the comment preceding it suggests, this is the lineat which the program will begin executing. All Java applications begin execution by callingmain( ).

The public keyword is an *access specifier,* which allows the programmer to control thevisibility of class members. When a class member is preceded by public, then that membermay be accessed by code outside the class in which it is declared. (The opposite of publicis private, which prevents a member from being used by code defined outside of its class.)In this case, main( ) must be declared as public, since it must be called by code outside ofits class when the program is started.

The keyword static allows main( ) to be called withouthaving to instantiate a particular instance of the class. This is necessary since main( ) iscalled by the Java Virtual Machine before any objects are made. The keyword void simplytells the compiler that main( ) does not return a value. As you will see, methods may alsoreturn values. If all this seems a bit confusing, don't worry. All of these concepts will bediscussed in detail in subsequent chapters.As stated, main( ) is the method called when a Java application begins. Keep in mind thatJava is case-sensitive. Thus, Main is different from main. It is important to understand thatthe Java compiler will compile classes that do not contain a main( ) method. But java has noway to run these classes. So, if you had typed Main instead of main, the compiler wouldstill compile your program. However, java would report an error because it would be unableto find the main( ) method.Any information that you need to pass to a method is received by variables specifiedwithin the set of parentheses that follow the name of the method. These variables are called*parameters.* If there are no parameters required for a given method, you still need to includethe empty parentheses. In main( ), there is only one parameter, albeit a complicated one. Stringargs[ ] declares a parameter named args, which is an array of instances of the class String.(*Arrays* are collections of similar objects.) Objects of type String store character strings. In thiscase, args receives any command-line arguments present when the program is executed.This program does not make use of this information, but other programs shown later in thismaterial.

The last character on the line is the {. This signals the start of main( )'s body. All of thecode that comprises a method will occur between the method's opening curly brace and itsclosing curly brace.

One other point:main( ) is simply a starting place for your program. Acomplex programwill have dozens of classes, only one of which will need to have a main( ) method to getthings started. When you begin creating applets—Java programs that are embedded in webbrowsers—you won't use main( ) at all, since the web browser uses a different means ofstarting the execution of applets.

The next line of code is shown here. Notice that it occurs inside main( ).

System.out.println("This is a simple Java program.");

This line outputs the string "This is a simple Java program." followed by a new line on thescreen. Output is actually accomplished by the built-in println ( ) method. In this case, println ( )displays the string which is passed to it. As you will see, println ( ) can be used to displayother types of information, too. The line begins with System.out. While too complicated toexplain in detail at this time, briefly, System is a predefined class that provides access to thesystem, and out is the output stream that is connected to the console.As you have probably guessed, console output (and input) is not used frequently inmost real-world Java programs and applets. Since most modern computing environmentsare windowed and graphical in nature, console I/O is used mostly for simple utility programs and for demonstration programs. Later in this book, you will learn other ways togenerate output using Java. But for now, we will continue to use the console I/O methods. Notice that the println ( ) statement ends with a semicolon. All statements in Java endwith a semicolon. The

reason that the other lines in the program do not end in a semicolonis that they are not, technically, statements.The first } in the program ends main( ), and the last } ends the Example class definition.
Note : Every class we write doesn't require to have main() method unless we want to run that class directly. The classes we write to use it in other classes doesn't need main() at all.

Tip: Class can have two types of functions:instance and static . Instance functions(default functions) can only be called on objects where as static functions can be calledon the class itself. Instance Functions vs Static Functions will be discussed moreelaborately in the later chapters.
public static void main(**String[] args**)

Java Development Environment
Java development environment is used to develop different applications in java. It provides complete environment to develop and run java applications. JDK (java development kit) used as java development environment.

Java IDE's
An integrated development environment (IDE) is a programming environment that has been packaged as an application program, typically consisting of a code editor, a compiler, a debugger, and a graphical user interface (GUI) builder. The IDE may be a standalone application or may be included as part of one or more existing and compatible applications.

IDEs provide a user-friendly framework for many modern programming languages, such as Visual Basic, Java, and PowerBuilder.
Here are the best-of-breed Java IDEs

1) Eclipse
Eclipse is a free IDE that has taken the Java industry by storm. Built on a plug-in architecture, Eclipse is highly extensible and customizable. Eclipse is the base IDE, but there are many Java-related plug-ins for Eclipse, and several commercial IDEs built on top of Eclipse. Among all one of the famous is "MyEclipse"-nicely packaged version of Eclipse with many of the best free J2EE plug-ins already installed.

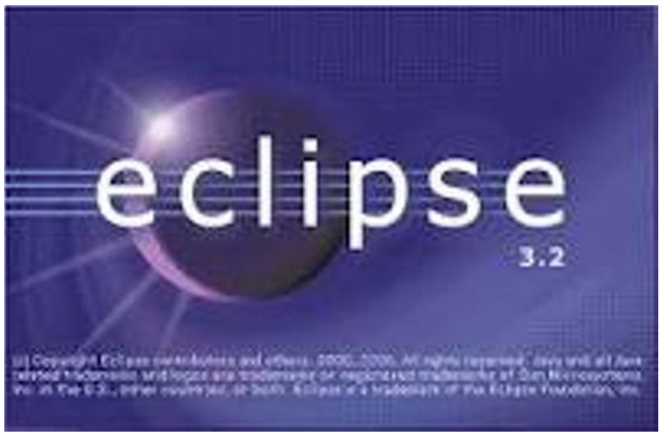
2) Netbeans
Netbeans is a free IDE backed by Sun Microsystems. It is the main competitor of Eclipse. Netbeans is built on a plugin architecture.

3) JBuilder
Jbuilder has long been the top commercial Java IDE. It is an excellent IDE, but it is not built to be open and extensible by third-part vendors.

4) IntelliJ IDEA
IntelliJ IDEA is a commercial IDE with a loyal following that swear by it. It has excellent J2EE and GUI support. It is extensible via plugins.

5)Oracle JDeveloper
Oracle JDeveloper is another powerful IDE with lots of support for Java EE capabilities (including EJB and Struts).

The famous java IDE's are Net Beans and Eclipse



JDK 1.7

The JDK is a development environment for building applications, applets, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java™ platform.

JDK directory Structure

Assuming that the JDK is installed in the d drive, below is the directory structure of the same:

D:\ Program Files\Java\JDK1.7.0


```
|_bin
   |____javac.exe
   |____java.exe
|_db
|_demo
|_include
|_jre
|_lib
|_sample
```

Bin:

The java development kit comes with a collection of tools that are used for developing and running programs.

These tools and utilities that will help one to develop, execute, debug, and document programs written in the java programming language are provided in the "bin" sub-directory.

These tools are the foundation of the JDK7 and they are designed to be used from command line.

Some of the basic tools are:

Javac	:It is the compiler for the Java programming language It compiles the source code and generates the byte code.
Java	:It is the launcher for java application.
Javap	:Java disassembler, which enables us to convert bytecode files into a program description.
Javadoc	:API document generator.
Jar	:Manage java archieve files.
Jdb	:java debugger, which helps us to find errors.
Javah	:Produces header files for use with native methods.

Runtime Environment

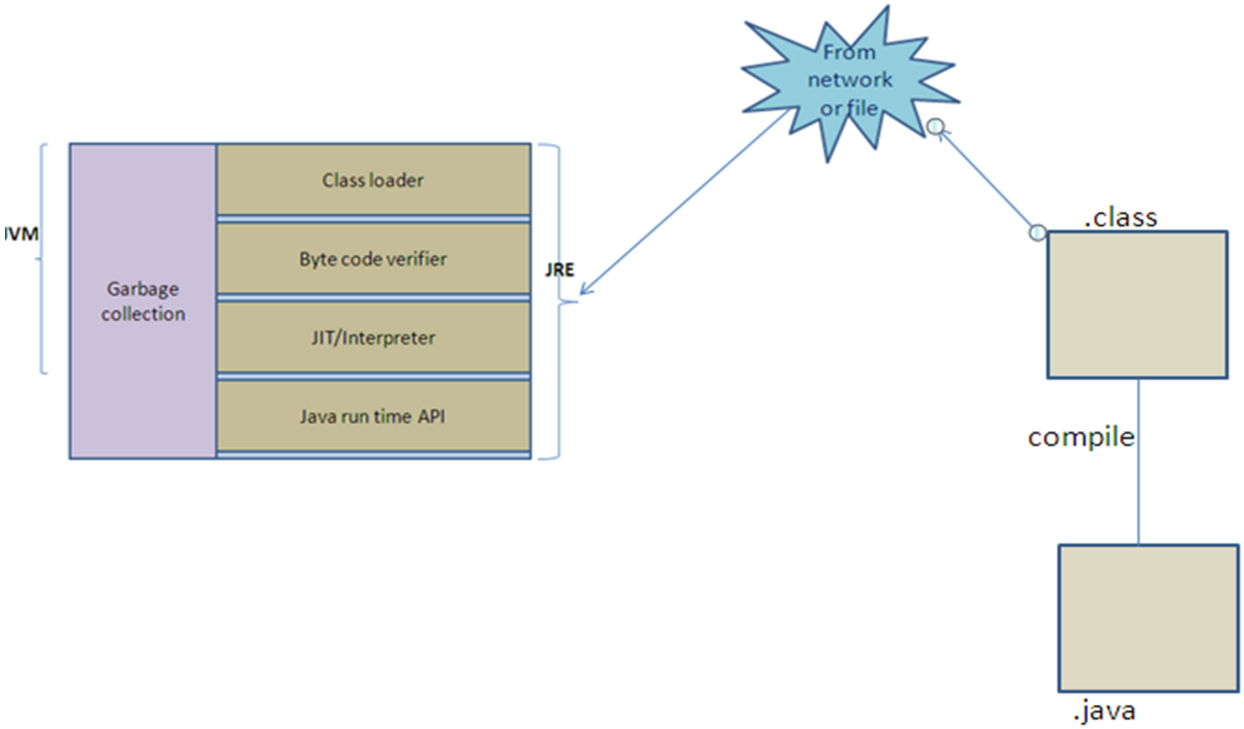
The Java(TM) Platform, Standard Edition Runtime Environment (JRE(TM)), excluding the JavaFX(TM) runtime, is intended for software developers and vendors to redistribute with their applications.
The Java SE Runtime Environment contains the Java virtual machine, runtime class libraries, and Java application launcher that are necessary to run programs written in the Java programming language.
It is not a development environment and does not contain development tools such as compilers or debuggers.Development environment is used to develop the application and Runtime environment is used to run the application.

JVM Architecture

A Java virtual machine is software that is implemented on virtual and non-virtual hardware and on standard operating systems. A JVM provides an environment in which Java bytecode can be executed, enabling such features as automated exception handling, which provides "root-cause" debugging information for every software error (exception), independent of the source code. A JVM is distributed along with a set of standard class libraries that implement the Java application programming interface (API). Appropriate APIs bundled together with JVM form the Java Runtime Environment (JRE).

JVMs are available for many hardware and software platforms. The use of the same bytecode for all JVMs on all platforms allows Java to be described as a "write once, run anywhere" programming language, as opposed to "write once, compile anywhere", which describes cross-platform compiled languages. Thus, the JVM is a crucial component of the Java platform.
Java bytecode is an intermediate language which is typically compiled from Java, but it can also be compiled from other programming languages. For example, Ada source code can be compiled to Java bytecode and executed on a JVM.

The Java Class loader is a part of the Java Runtime Environment that dynamically loadsJava classes into the Java Virtual Machine.[1] Usually classes are only loaded on demand. The Java run time system does not need to know about files and file systems because of class loaders. Delegation is an important concept to understand when learning about class loaders.
The Byte code verifier's job is basically to check that the bytecode isn't going to reference a nonexistent piece of data, or access an uninitialized variable, or perform some other illegal operation. A valid Java compiler won't generate code that does any of these things, so in general the verifier is just redundant. It's good for checking bytecode that comes from other places, though.



The JVM runtime executes `.class` or `.jar` files, emulating the JVM instruction set by interpreting it, or using a just-in-time compiler **(JIT)** such as Oracle's HotSpot. JIT compiling, not interpreting, is used in most JVMs today to achieve greater speed. There are also ahead-of-time compilers that enable developers to precompiled class files into native code for particular platforms.

Steps to execute program

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

1. Open notepad and add the code as above.
2. Save the file as : MyFirstJavaProgram.java.
3. Open a command prompt window and go o the directory where you saved the class. Assume its C:\.
4. 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).
5. Now type ' java MyFirstJavaProgram ' to run your program.
6. You will be able to see ' Hello World ' printed on the window.

C : >javac MyFirstJavaProgram.java

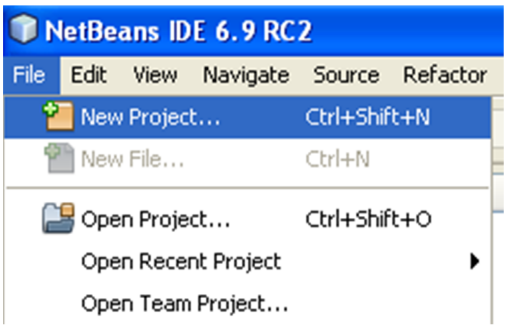
C : > java MyFirstJavaProgram

Hello World

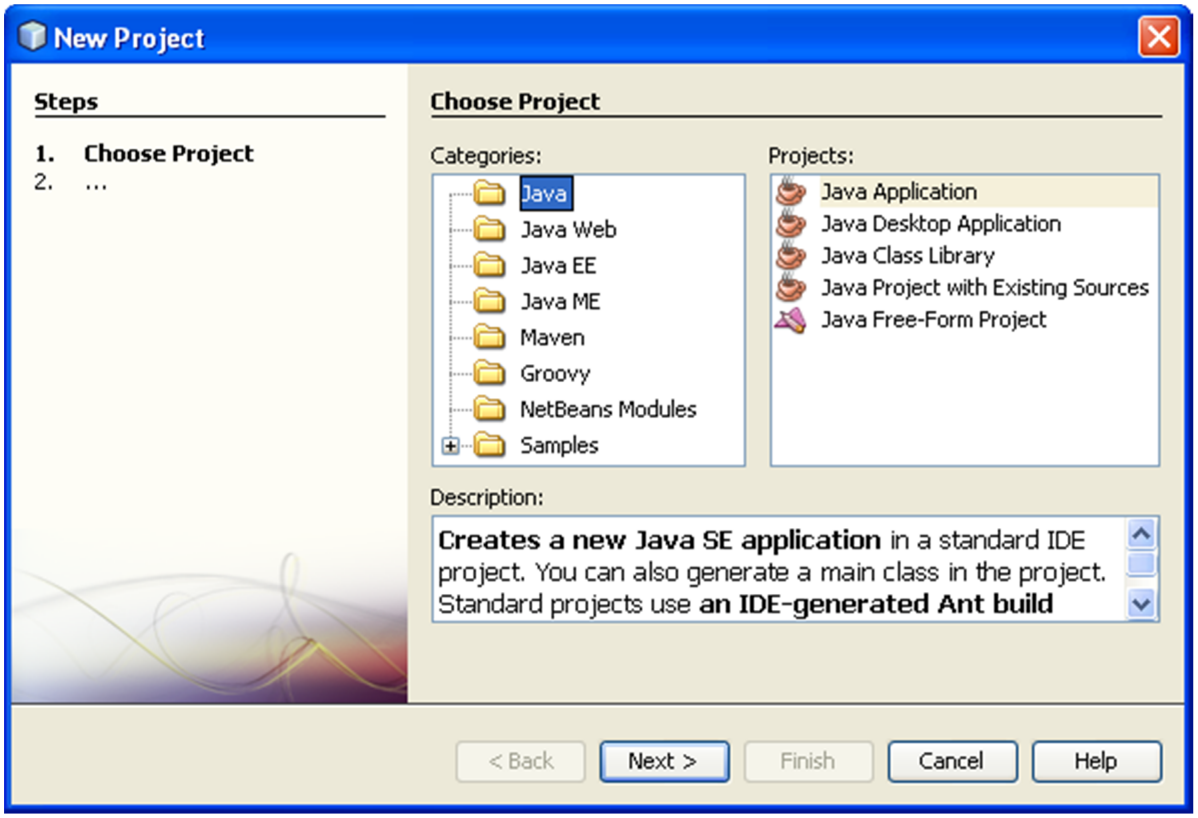
Simple program in net beans:

To create anNetbeans IDE project:

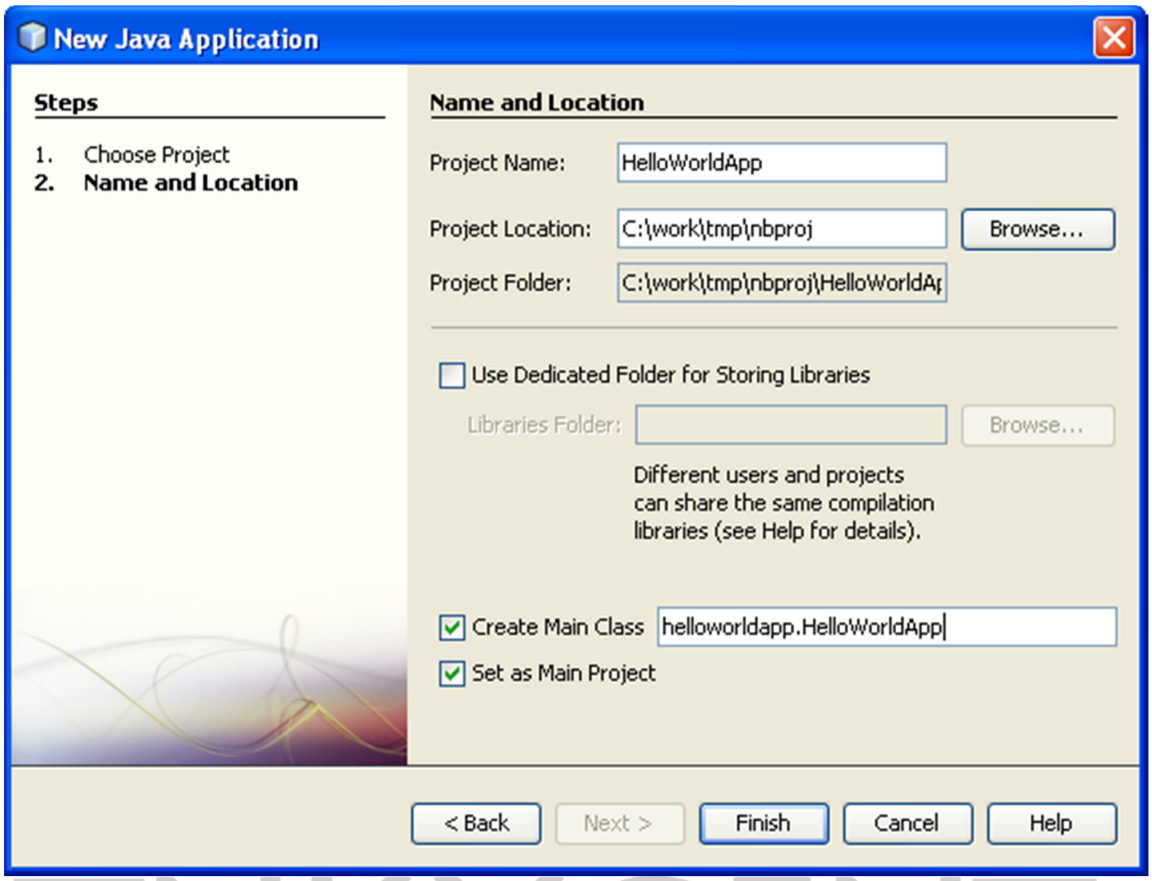
1. Start NetBeans IDE.
2. In the IDE, choose File > New Project (Ctrl-Shift-N), as shown in the figure below.



3. In the New Project wizard, expand the Java category and select Java Application as shown in the figure below. Then click Next.



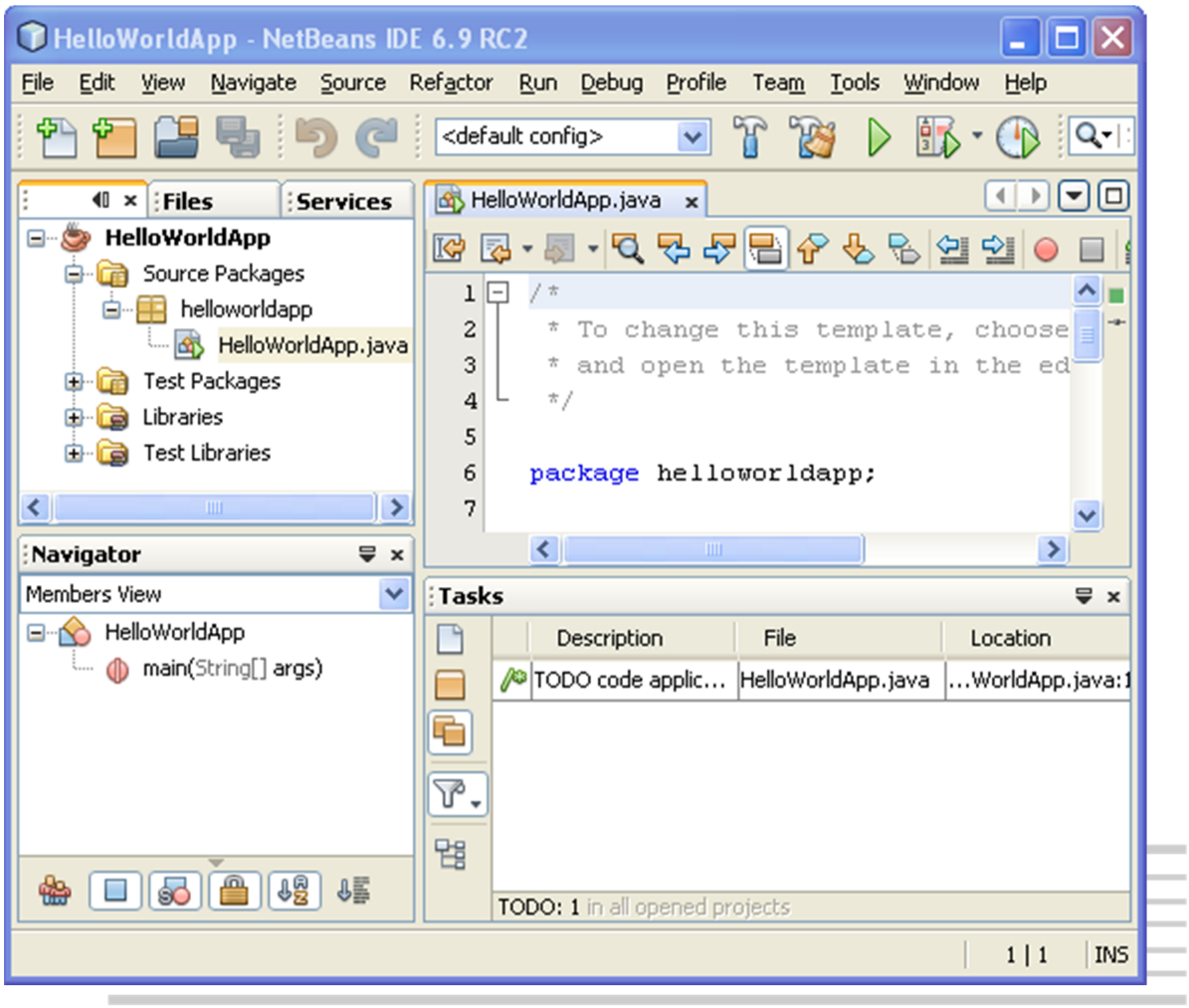
4. In the Name and Location page of the wizard, do the following (as shown in the figure below):
   - In the Project Name field, type `HelloWorldApp`.
   - Leave the Use Dedicated Folder for Storing Libraries checkbox unselected.
   - In the Create Main Class field, type `helloworldapp.HelloWorldApp`.
   - Leave the Set as Main Project checkbox selected.



5. Click Finish.

The project is created and opened in the IDE. You should see the following components:

- The Projects window, which contains a tree view of the components of the project, including source files, libraries that your code depends on, and so on.
- The Source Editor window with a file called `HelloWorldApp` open.
- The Navigator window, which you can use to quickly navigate between elements within the selected class.
- The Tasks window, which lists compilation errors as well other tasks that are marked with keywords such as XXX and TODO.



Compiling and Running the Program

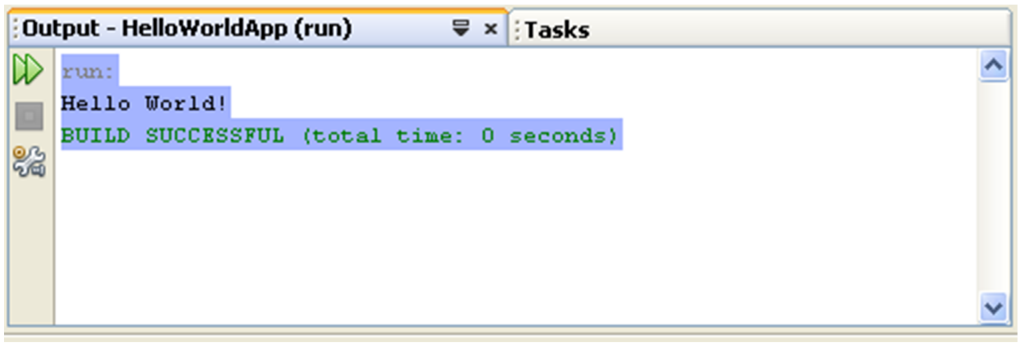
Because of the IDE's Compile on save feature, you do not have to manually compile your project in order to run it in the IDE. When you save a Java source file, the IDE automatically compiles it.

The Compile on Save feature can be turned off in the Project Properties window. Right-click your project, select Properties. In the Properties window, choose the Compiling tab. The Compile on Save checkbox is right at the top. Note that in the Project Properties window you can configure numerous settings for your project: project libraries, packaging, building, running, etc.

To run the program:

- Choose Run > Run Main Project (F6).

The next figure shows what you should now see.



If there are compilation errors, they are marked with red glyphs in the left and right margins of the Source Editor. The glyphs in the left margin indicate errors for the corresponding lines. The glyphs in the right margin show all of the areas of the file that have errors, including errors in lines that are not visible. You can mouse over an error mark to get a description of the error. You can click a glyph in the right margin to jump to the line with the error.

Building and Deploying the Application

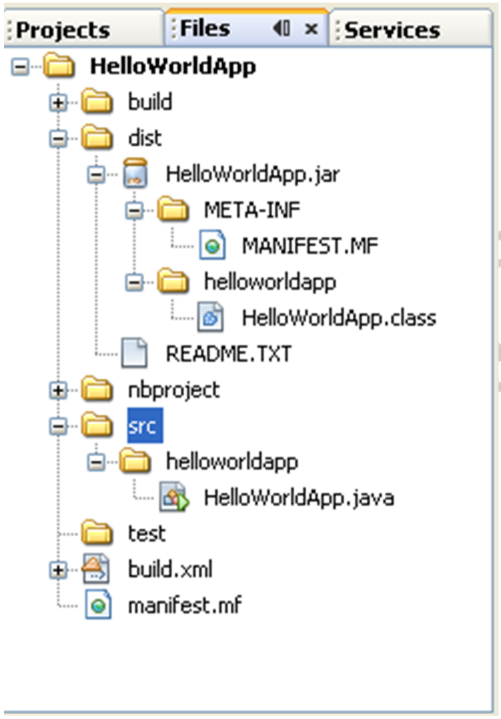
Once you have written and test run your application, you can use the Clean and Build command to build your application for deployment. When you use the Clean and Build command, the IDE runs a build script that performs the following tasks:

- Deletes any previously compiled files and other build outputs.
- Recompiles the application and builds a JAR file containing the compiled files.

To build your application:

- Choose Run > Clean and Build Main Project (Shift-F11)

You can view the build outputs by opening the Files window and expanding the `HelloWorldApp` node. The compiled bytecode file `HelloWorldApp.class` is within the `build/classes/helloworldapp`subnode. A deployable JAR file that contains the `HelloWorldApp.class` is within the `dist` node.



Session 5 – Java Language-I

OBJECTIVES

After completing this session you will be able to understand:

- ✓ Datatypes
- ✓ Stack and Heap
- ✓ Operators
- ✓ Increment & Decrement
- ✓ Type Casting

Data Types

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals, or characters in these variables.

There are two data types available in Java:

- Primitive Data Types
- Reference/Object Data Types

A primitive type is predefined by the language and is named by a reserved keyword. There are eight primitive data types that are supported by the Java programming language. It is a simple non-object data type that represents a single value.

The following table summarizes the sizes and ranges of the primitive data types:

Data Type	Size (Bytes)	Range
Byte	1	-128 to 127(-2^7 to 2^7-1)
Short	2	-32,768 to 32,767(-2^{15} to $2^{15}-1$)
Int	4	-2,147,483,648 to 2,147,483,647
Long	8	(-2^{31} to $2^{31}-1$)
Float	4	
Double	8	
Boolean	Dependent On JVM	true or false
Char	2	

- **byte:** The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive). The byte data type can be useful for saving memory in large arrays, where the memory savings actually matters. They can also be used in place of int where their limits help to clarify your code; the fact that a variable's range is limited can serve as a form of documentation.Example : byte a = 100 , byte b = -50
- **short:** The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive). As with byte, the same guidelines apply: you can use a short to save memory in large arrays, in situations where the memory savings actually matters.Example : short s= 10000 , short r = -20000
- **int:** The int data type is a 32-bit signed two's complement integer. It has a minimum value of -2,147,483,648 and a maximum value of 2,147,483,647 (inclusive). For integral values, this data type is generally the default choice unless there is a reason (like the above) to choose something else. This data type will most likely be large enough for the numbers your program will use, but if you need a wider range of values, use long instead.Example : int a = 100000, int b = -200000
- **long:** The long data type is a 64-bit signed two's complement integer. It has a minimum value of -9,223,372,036,854,775,808 and a maximum value of 9,223,372,036,854,775,807 (inclusive). Use this data type when you need a range of values wider than those provided by int.
  Example :int a = 100000L, int b = -200000L
- **float:** The float data type is a single-precision 32-bit IEEE 754 floating point. It ranges from 1.40129846432481707e-45 to 3.40282346638528860e+38 (positive or negative). Use a float (instead of double) to save memory in large arrays. We do not use this data type for the exact values such as currency.Example : float f1 = 234.5f

- **double:** The double data type is a double-precision 64-bit IEEE 754 floating point. It ranges from 4.94065645841246544e-324d to 1.79769313486231570e+308d (positive or negative). For decimal values, this data type is generally the default choice. As mentioned above, this data type should never be used for precise values, such as currency.Example : double d1 = 123.4
- **boolean:** The boolean data type has only two possible values: true and false. Use this data type for simple flags that track true/false conditional statements.Example : boolean one = true
- **char:** The char data type is a single 16-bit Unicode character. It ranges from 0 to 65,535. They are not same as ints, shorts etc.Example: char letterA ='A'

Note on Data Types

- ✓ All integer data types are two's complement integers.
- ✓ The float data type is a single-precision 32-bit IEEE 754 floating point.
- ✓ The double data type is a double-precision 64-bit IEEE 754 floating point.
- ✓ Variables of type boolean may only take the values true or false. Their representation size depends on JVM implementation.
- ✓ A 'signed data type' is a numeric type whose value can be positive, zero or negative; an 'unsigned data type' is a numeric type whose value can only be positive or zero.
- ✓ All numeric primitive types are signed.
- ✓ Char type is integral but unsigned.
- ✓ When we declare a variable inside a method of a class i.e. local variable, we need to initialize before use. But when we declare a variable as class level member i.e. instance variable without initialization that will be initialized to its default value by compiler.

The following table summarizes the default values for the java built-in data types:

Data Type	**Default Value (for class fields)**
Byte	0
Short	0
Int	0
Long	0L
Float	0.0F
Double	0.0d
Boolean	False
Char	'\u0000'

Note: For reference data types the default value will be null.

Reference Data Types:

- Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Employee, Puppy etc.
- Class objects, and various type of array variables come under reference data type.
- Default value of any reference variable is null.
- A reference variable can be used to refer to any object of the declared type or any compatible type.
- Example : Animal a = new Animal("tiger");

Strings

In addition to the boolean, character, integer, and floating-point data types, Java also has a data type for working with strings of text (usually simply called *strings*). The `String` type is a class, however, and is not one of the primitive types of the language. Because strings are so commonly used, though, Java does have a

syntax for including string values literally in a program. A `String` literal consists of arbitrary text within double quotes. For example:
"Hello, world"
"'This' is a string!"

Stack and Heap

Stack – Data is stored in stack using the Last in First out (LIFO) method. This means that storage in the memory is allocated and reallocated at only one end of the memory called the top of the stack. Stack is a section of memory and its associated registers that is used for temporary storage of information in which the most recently stored item is the first to be retrieved.

Heap -Heap is an area of memory used for dynamic memory allocation. Blocks of memory are allocated and freed in this case in an arbitrary order. The pattern of allocation and size of blocks is not known until run time. Heap is usually being used by a program for many different purposes.

- The stack is much faster than the heap but also smaller and more expensive.
- All primitive data types stored in stack and reference types stored in heap.

Operators

Java provides a full set of operators, most of which are taken from C and C++. Java's operators differ from their counterparts in some aspects. They are used to manipulate the primitive data types.The operators in Java fall into eight different categories:

❖ Simple Assignment Operator

It is the most common operator almost used with all the programming languages. It is represented by '=' symbol in Java which is used to assign a value to a variable lying to the left side of the assignment operator.

Syntax:-

<variable>=<expression>;

For Example:

intsno=1; booleansid=true;

Note: You can also assign a value to the more than one variable simultaneously. For example, see these expressions shown as:

a = b = c = 7;

In all cases a value of right side is being assigned to its type of variable lying to the left side.

❖ Arithmetic Operator

The Java programming language provides operators that perform addition, subtraction, multiplication, division and modulo.

Operator	Name of the Operator	Example
+	additive operator	x=x+1;
-	subtraction operator	x=x-1;
*	multiplication operator	x=x*1;
/	division operator	x=x/1;
%	modulo operator	x=x%1;

Note: Modulo operator divides one operand by another and returns the remainder as its result.

❖ Unary Operator

The unary operators require only one operand; they perform various operations such as incrementing/decrementing a value by one, negating an expression, or inverting the value of a Boolean.

Symbol	Name of the Operator	Operation	Example
+	Unary plus	Indicates positive value	int number= +1;
-	Unary minus	Negates an expression	number= -number;

++	Increment	Increments a value by 1	number= ++number;
--	Decrement	Decrements a value by 1	number=--number;
!	Logical compliment	Inverts a boolean value	

❖ Equality and Relational Operator

The equality and relational operators determine if one operand is greater than, less than, equal to, or not equal to another operand.

Symbol	Name of the Operator	Operation	Example
==	equal to	x is equal to y	x==y;
!=	not equal to	x is not equal to y	x!=y;
>	Greater than	x is greater than y	x>y;
>=	Greater than or equal to	x is greater than or equal to y	x>=y;
<	Less than	x is less than y	x<y;
<=	Less than or equal to	x is less than or equal to y	x<=y;

Note: Keep in mind that you must use "==", not "=", when testing if two primitive values are equal.

❖ Conditional Operators

Conditional operators return a true or a false value based on the state of the variables i.e. the operations using conditional operators are performed between the two boolean expressions. These operators exhibit "short-circuiting" behavior, which means that the second operand is evaluated only if needed.

Symbol	Name of the Operator
&	AND
&&	Conditional AND
|	OR
||	Conditional OR
!	NOT
?:	Ternary

Note: Each argument to a logical operator must be a boolean data type, and the result is always a boolean data type.

❖ The Type Comparison Operator instanceof

Java provides a run-time operator instanceof to compare a class and an instance of that class. This operator "instanceof" compares an object to a specified class type. The instanceof operator is defined to know about an object's relationship with a class. It evaluates to true, if the object or array is an instance of the specified type; otherwise it returns false.

Its signature is written as:

objectinstanceof**type**

Note: The instanceof operator can be used with the arrays and objects. It can't be used with primitive data types and values.

❖ Bitwise and Bit Shift Operators

In Java the bitwise and bit shift operators are used to manipulate the contents of variables at a bit level according to binary format. These operators perform bitwise and bit shift operations on integral type variables.

Symbol	Name of the Operator
~	Unary bitwise complement
&	Bitwise AND
|	Bitwise Inclusive OR
^	Bitwise Exclusive OR
<<	Signed left shift
>>	Signed right shift
>>>	Unsigned right shift

The unary bitwise complement operator "~" inverts a bit pattern; it can be applied to any of the integral types, making every "0" a "1" and every "1" a "0". For example, a byte contains 8 bits; applying this operator to a value whose bit pattern is "00000000" would change its pattern to "11111111".

The signed left shift operator "<<" shifts a bit pattern to the left and the signed right shift operator ">>" shifts a bit pattern to the right. The bit pattern is given by the left-hand operand and the number of positions to shift by the right-hand operand. The unsigned right shift operator ">>>" shifts a zero into the leftmost position, while the leftmost position after ">>" depends on sign extension.

The table below shows the list of operators that follow the precedence:

Operators	Precedence
Postfix	expr++, expr--
Unary	++expr, --expr, +expr, -expr, ~, !
Multiplicative	* , /, %
Additive	+, -
Shift	<<, >>, >>>
Relational	<, >, <=, >=, instanceof
Equality	==, !=
bitwise AND	&
bitwise Exclusive OR	^
bitwise Inclusive OR	|
Logical AND	&&
Logical OR	||
Ternary	? :

Tech Tips on Operators

- ✓ In Simple Assignment Operator, if the value already exists in that variable then it will be overwritten by the assignment operator (=).
- ✓ This operator can also be used on objects to assign object references;
- ✓ The Equality operator "= =" differs from an assignment operator "=" i.e. the equality operator is used to determine whether one operand is equal to another operand or not while the assignment operator is used to assign a value to a variable.

Sample program on Increment – Decrement Operators

```
//ArithmeticOperatorsDemo
public class ArithmeticOperatorsDemo {
        publicArithmeticOperatorsDemo() {
                int x, y = 10, z = 5;
                x = y + z;
                System.out.println("+ operator resulted in " + x);
                x = y - z;
                System.out.println("- operator resulted in " + x);
                x = y * z;
                System.out.println("* operator resulted in " + x);
                x = y / z;
                System.out.println("/ operator resulted in " + x);
                x = y % z;
                System.out.println("% operator resulted in " + x);
                x = y++;
                System.out.println("Postfix ++ operator resulted in " + x);
                x = ++z;
                System.out.println("Prefix ++ operator resulted in " + x);
                x = -y;
                System.out.println("Unary operator resulted in " + x);
                // Some examples of special Cases
                inttooBig = Integer.MAX_VALUE + 1; // -2147483648 which is
                // Integer.MIN_VALUE.
                inttooSmall = Integer.MIN_VALUE - 1; // 2147483647 which is
                // Integer.MAX_VALUE.
                System.out.println("tooBig becomes " + tooBig);
                System.out.println("tooSmall becomes " + tooSmall);
                System.out.println(4.0 / 0.0); // Prints: Infinity
                System.out.println(-4.0 / 0.0); // Prints: -Infinity
                System.out.println(0.0 / 0.0); // Prints: NaN
                double d1 = 12 / 8; // result: 1 by integer division. d1 gets the value
                // 1.0.
                double d2 = 12.0F / 8; // result: 1.5
```

```
            System.out.println("d1 is " + d1);
            System.out.println("d2 iss " + d2);
        }
        public static void main(String args[]) {
            newArithmeticOperatorsDemo();
        }
```

Type Casting

Type Casting refers to changing an entity of one datatype into another. This is important for the type conversion in developing any application. If you will store aint value into a byte variable directly, this will be illegal operation. For storing your calculated int value in a byte variable you will have to change the type of resultant data which has to be stored. This type of operation has illustrated below :

In this example we will see that how to convert the data type by using type casting. In the given line of the code `c = (char)(t?1:0);` illustrates that if t which is `boolean type variable is true then value of c which is the char type variable will be 1 but 1 is a numeric value. So, 1 is changed into character according to the Unicode value. But in this line c = (char)(t?'1':'0');` 1 is already given as a character which will be stored as it is in the `char` type variable c.

Code block showing the Type Casting:

```
//Conversion.java
public class conversion{
  public static void main(String[] args){
  boolean t = true;
  byte b = 2;
  short s = 100;
  char c = 'C';
  int i = 200;
  long l = 24000;
  float f = 3.14f;
  double d = 0.000000000000053;
  String g = "string";
  System.out.println("Value of all the variables like");
  System.out.println("t = " + t );
  System.out.println("b = " + b );
  System.out.println("s = " + s );
  System.out.println("c = " + c );
  System.out.println("i = " + i );
  System.out.println("l = " + l );
  System.out.println("f = " + f );
  System.out.println("d = " + d );
  System.out.println("g = " + g );
  System.out.println();
  //Convert from boolean to byte.
  b = (byte)(t?1:0);
  System.out.println("Value of b after conversion : " + b);
  //Convert from boolean to short.
  s = (short)(t?1:0);
  System.out.println("Value of s after conversion : " + s);
  //Convert from boolean to int.
  i = (int)(t?1:0);
  System.out.println("Value of i after conversion : " + i);
  //Convert from boolean to char.
  c = (char)(t?'1':'0');
  System.out.println("Value of c after conversion : " + c);
  c = (char)(t?1:0);
  System.out.println("Value of c after conversion in unicode : " + c);
```

```
  //Convert from boolean to long.
  l = (long)(t?1:0);
  System.out.println("Value of l after conversion : " + l);
  //Convert from boolean to float.
  f = (float)(t?1:0);
  System.out.println("Value of f after conversion : " + f);


  //Convert from boolean to double.
  d = (double)(t?1:0);
  System.out.println("Value of d after conversion : " + d);
  //Convert from boolean to String.
  g = String.valueOf(t);
  System.out.println("Value of g after conversion : " + g);
  g = (String)(t?"1":"0");
  System.out.println("Value of g after conversion : " + g);
  int sum = (int)(b + i + l + d + f);
  System.out.println("Value of sum after conversion : " + sum);
  }
}
```

Session 6 – Java Language-II

OBJECTIVES

After completing this session you will be able to understand:

- ✓ Control Structures
- ✓ If
- ✓ Switch..case
- ✓ Loops : while loop
- ✓ for loop
- ✓ Explain Other Structures
  - o Input & Output
  - o Output
  - o Command Line Arguments
- ✓ Arrays

Java Language -II

Control Structures

The control statements are used to control the flow of execution of the program. This execution order depends on the supplied data values and the conditional logic. The statements inside your source files are generally executed from top to bottom, in the order that they appear. Control flow statements, however, break up the flow of execution by employing decision making, looping, and branching, enabling your program to conditionally execute particular blocks of code.

❖ **The if-then Statement**

The if-then statement is the most basic of all the control flow statements. It tells your program to execute a certain section of code only if a particular test evaluates to true. If this test evaluates to false, control jumps to the end of the if-then statement.

Syntax:-if (<conditional expression>)

<statement action>;

❖ **The if-then-else Statement**

The if-then-else statement provides a secondary path of execution when an "if" clause evaluates to false i.e. else block is executed if "if" statement is false.

```
Syntax:- if(conditional_expression){
<statements>;
        ...;
        ...;
    }
else{
<statements>;
        ....;
        ....;
    }
```

❖ **switch Statement**

This is an easier implementation to the if-else statements. The keyword "switch" is followed by an expression that should evaluates to byte, short, char or int primitive data types, only.

- ✓ The expression that creates labels for the case must be unique.
- ✓ The switch expression is matched with each case label.
- ✓ Only the matched case is executed, if no case matches then the default statement (if present) is executed.
- ✓ The default section handles all values that aren't explicitly handled by one of the case sections.


```
Syntax:-
switch(control_expression){
case expression 1:
<statement>;
break;
case expression 2:
<statement>;
                    break;
        ...
case expression n:
<statement>;
                    break;
default:
<statement>;
                    break;    }//end switch
```

Ex: - The program displays the name of the day, based on the value of week.

```
public class SwitchDemo {
public static void main(String args[]){
int week = 5;
switch(week){
case 1:  System.out.println("monday"); break;
case 2:  System.out.println("tuesday"); break;
case 3:  System.out.println("wednesday"); break;
case 4:  System.out.println("thursday"); break;
case 5:  System.out.println("friday"); break;
case 6:  System.out.println("saturday"); break;
case 7:  System.out.println("sunday"); break;
default: System.out.println("Invalid week");break;
      }
   }
}
```

Output: - Friday

❖ The while and do-while Statements

The while statement continually executes a block of statements while a particular condition is true.

- ✓ The while statement evaluates expression, which must return a boolean value i.e. true or false.
- ✓ If the expression evaluates to true, the while statement executes the statement(s) in the while block.
- ✓ The while statement continues testing the expression and executing its block until the expression evaluates to false.

Syntax:-

```
while (expression) {
statement(s)
}
```

Ex:-The following program demonstrates the `while` statement to print the values from 1 through 4

```
public class WhileDemo {
public static void main(String[] args){
int count = 1;
while (count < 5) {
System.out.println("Count is: " + count);
count++;
       }
    }
}
```

Output:-

```
Count is: 1
Count is: 2
Count is: 3
Count is: 4
```

❖ do-while Statements

This is another looping statement that tests the given condition past so you can say that the do-while looping statement is a past-test loop statement.

- ✓ First the do block statements are executed then the condition given in while statement is checked.
- ✓ Even the condition is false in the first attempt, do block of code is executed at least once.

Syntax:-

```
do {
statement(s)
} while (expression);
```

Ex:-The following program demonstrates the do-while loop used to print numbers from 1 to 5.

```
public class DoWhileDemo {
public static void main(String[] args){
int count = 1;
System.out.println("Printing numbers from 1 to 5");
do {
System.out.println("Count is: " + count);
count++;
        } while(count<=5);
    }
}
```

Output:-

```
Printing numbers from 1 to 5
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Count is: 5
```

Note: The difference between do-while and while is that do-while evaluates its expression at the bottom of the loop instead of the top. Therefore, the statements within the do block are always executed at least once.

❖ The for Statement

The for statement provides a compact way to iterate over a range of values. Programmers often refer to it as the "for loop" because of the way in which it repeatedly loops until a particular condition is satisfied. When using this version of the for statement, keep in mind that:

- ✓ The initialization expression initializes the loop; it's executed once, as the loop begins.
- ✓ When the termination expression evaluates to false, the loop terminates.
- ✓ The increment expression is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment or decrement a value.

Syntax:-

```
for (initialization; termination; increment) {
statement(s)
}
```

Ex:-The following example demonstrates the for loop used to print numbers from 1 to 7.

```
public class ForDemo {
        public static void main(String[] args) {
                System.out.println("Printing Numbers from 1 to 7");
                for (int count = 1; count <= 7; count++) {
                        System.out.println(count);
                }
        }}
```

Output:-Printing Numbers from 1 to 7

```
1
2
3
4
5
6
7
```

Note: If the variable that controls a for statement is not needed outside of the loop, it's best to declare the variable in the initialization expression.

- **The for-each Loop**

The basic for loop was extended in Java 5 to make iteration over arrays and other collections more convenient. This newer for statement is called the enhanced for or for-each.The for-each loop is used to access each successive value in a collection of values.

Syntax:-

```
for (type var : arr) { body-of-loop}
```

Ex:-The following program demonstrates the iteration of numbers through array, which hold numbers from 1 through 5

```
public class ForEachLoop {
public static void main(String[] args){
int[] numbers = {1,2,3,4,5};
for (int item : numbers) {
System.out.println("Count is: " + item);
        }
    }
}
```

Output:-

```
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Count is: 5
```

Note: Although the enhanced for loop can make code much clearer, it can't be used in some common situations. Don't use it if you need compatibility with versions before Java 5.

Input & Output

All Java programs automatically import the **java.lang** package. This package defines a class called System, which encapsulates several aspects of the run-time environment. For example, using some of its methods, you can obtain the current time and the settings of various properties associated with the system. System also contains three predefined stream

variables: in, out, and err. These fields are declared as public, static, and final within System. This means that they can be used by any other part of your program and without reference to a specific System object.
System.out refers to the standard output stream. By default, this is the console.
System.in refers to standard input, which is the keyboard by default. System.err refers to the standard error stream, which also is the console by default. However, these streams may be redirected to any compatible I/O device. System.in is an object of type InputStream; System.out and System.err are objects of type PrintStream. These are byte streams, even though they typically are used to read and write characters from and to the console.
There are few ways to read input string from your console/keyboard. The following smaple code shows how to read a string from the console/keyboard by using Java.

Using Buffered Reader:

```
public class ConsoleReadingDemo {
   public static void main(String[] args) {
      // ====
      BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
      System.out.print("Please enter user name : ");
      String username = null;
      try {
         username = reader.readLine();
      } catch (IOException e) {
         e.printStackTrace();
      }
      System.out.println("You entered : " + username);
Using Scanner class:
      // ===== In Java 5, Java.util,Scanner is used for this purpose.
      Scanner in = new Scanner(System.in);
      System.out.print("Please enter user name : ");
      username = in.nextLine();
      System.out.println("You entered : " + username);

 Using Console class;
      // ====== Java 6
      Console console = System.console();
      username = console.readLine("Please enter user name : ");
      System.out.println("You entered : " + username);
   }
}
```

Output

Console output is most easily accomplished with print( ) and println( ), described earlier,which are used in most of the examples in this book. These methods are defined by theclass PrintStream (which is the type of object referenced by System.out). Even thoughSystem.out is a byte stream, using it for simple program output is still acceptable. However,a character-based alternative is described in the next section.Because PrintStream is an output stream derived from OutputStream, it also implementsthe low-level method write( ). Thus, write( ) can be used to write to the console. The simplestform of write( ) defined by PrintStream is shown here:

void write(int*byteval*)

This method writes to the stream the byte specified by *byteval.* Although *byteval*is declaredas an integer, only the low-order eight bits are written. Here is a short example that useswrite( ) to output the character "A" followed by a newline to the screen:

```
// Demonstrate System.out.write().
classWriteDemo {
public static void main(String args[]) {
int b;
b = 'A';
System.out.write(b);
System.out.write('\n');
}
}
```

You will not often use write( ) to perform console output (although doing so might beuseful in some situations), because print( ) and println( ) are substantially easier to use.

Formatted output

Java 5 implements *formatted output* with `printf()`. This concept will be very familiar to C and Fortran programmers.

format(). Amazingly, there was no built-in way to right justify numbers in Java until Java 5. You had to use *if* or *while* to build the padding yoursefl. Java 5 now provides the `format()` method (and in some cases also the equivalent `printf()` method from C)..

The *format()* method's first parameter is a string that specifies how to convert a number. For integers you would typically use a "%" followed by the number of columns you want the integer to be right justified in, followed by a decimal conversion specifier "d". The second parameter would be the number you want to convert. For example,

```
int n = 2;
System.out.format("%3d", n);
```

This would print the number in three columns, that is with two blanks followed by 2.
You can put other non-% characters in front or back of the conversion specification, and they will simply appear literally. For example,

```
int n = 2;
System.out.format("| %3d |", n);
would print

|  2 |
```

Command Line Arguments

Java application can accept any number of arguments directly from the command line. The user can enter command-line arguments when invoking the application. When running the java program from java command, the arguments are provided after the name of the class separated by space. For example,

suppose a program named CmndLineArguments that accept command line arguments as a string array and echo them on standard output device.
C:\>java CmndLineArguments nkxgen zero one two three
CmndLineArguments.java

```
/**
  * How to use command line arguments in java program.
  */
 class CmndLineArguments {
  public static void main(String[] args) {
      int length = args.length;
      if (length <= 0) {
        System.out.println("You need to enter some arguments.");
      }
      for (int i = 0; i < length; i++) {
        System.out.println(args[i]);
      }
   }
  }
```

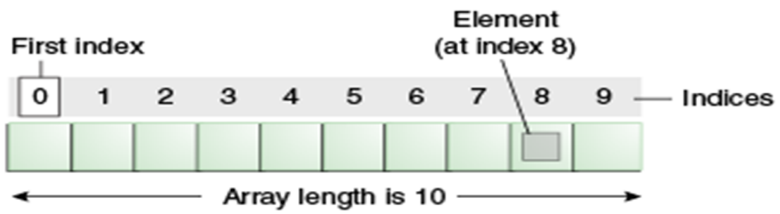
Run program with some command line arguments like:
>java CmndLineArguments nkxgen zero one two three

OUTPUT
Command line arguments were passed :
nkxgen
zero
one
two
three

Arrays

Array: Array is the most important thing in any programming language. By definition, array is the static memory allocation. It allocates the memory for the same data type in sequence. It contains multiple values of same types. It also store the values in memory at the fixed size. Multiple types of arrays are used in any programming language such as: one - dimensional, two - dimensional or can say multi - dimensional.

An array is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed.



An array of ten elements

Each item in an array is called an element, and each element is accessed by its numerical index. As shown in the above illustration, numbering begins with 0. The 9th element, for example, would therefore be accessed at index 8.

1. An array is a group of variables of the same data type and referred to by a common name.
2. An array is contiguous block of memory locations referred by a common name.

For Example:-To store the marks of 200 students, you can declare an array, marks, of size 200 and can store the marks of as many students.

int marks[] = new int[200];

Why Array is needed?

You might come across a situation where you need to store similar type of values for a large number of data items.

1. An array consists of a name and the number of elements of the array.
2. You can refer to a specific array element by the array name and the element number, which is known as the index number.

For Example:-

To store the marks of all the students of a university, you need to declare thousands of variables. In addition, each variable name needs to be unique. To avoid such situations, you can use arrays.

Note:- an array index element always starts with zero.

The Array class implicitly extends java.lang.Object so an array is an instance of Object.

Advantages of Java Array:

- An array can hold primitive types data.
- An array has its size that is known as array length.
- An array knows only its type that it contains. Array type is checked at the compile-time.

Disadvantages of Java Array:

- An array has fixed size.
- An array holds only one type of data (including primitive types).

Creating Arrays

1. The length of an array is fixed at the time of its creation.
2. An array represents related entities having the same data type in contiguous or adjacent memory locations.
3. The related data having data items form a group and are referred to by the same name.
4. A specific value in an array is accessed by placing the index value of the desired element in a square bracket.

For Example:-

student[85];

Here, the student is the name of the array and of size 85. The complete set of values is known as an array and the individual entities are called as elements of the array.

The various types of arrays in java are:

- One-dimensional arrays

- two-dimensional arrays

One-dimensional Arrays

One-dimensional array is a list of variables of the same data type.

Syntax to declare a one-dimensional array

typearray_name []; //type is the datatype of the array.

For Example:-

String students []; // students is name of the array.

Allocating Memory to Arrays

The new operator is used to allocate memory to an array.

Syntax to allocate memory

array_name = new type[size];

For Example:-

studentss = new String[10]; //size of the array is 10.

Two-dimensional Arrays

In additions to one-dimensional arrays, you can create two-dimensional arrays. To declare two-dimensional arrays, you need to specify multiple square brackets after the array name.

Syntax to declare a two dimensional array

typearray_name = new type[rows][cols];

For Example:-

intmultidim[] = new int[3][];

In a two-dimensional array,

- You need to allocate memory for only the first dimension.
- You can allocate the remaining dimensions separately.
- When you allocate memory to the second dimension, you can also allocate different number to each dimension.

For Example:-

```
intmultidim[] = new int[3][];
multidim[0] = new int[1];
multidim[1] = new int[4];
```

Accessing Arrays

You need to access various elements of an array to assign, retrieve, and manipulate the values stored in the array.

Assigning values to the Elements of an Array

To access a specific array,

1. You need to specify the name of the array and the index number of the element.
2. The index position of the first element in the array is 0.

For Example:-

```
String designations[];
designations = new String[2];
designations[0] = “Asst Vice President”;
designations[1]=”Vice President”;
```

- ✓ You can declare and allocate memory to a user-defined array in a single statement.

Syntax
typearr [] = new type[size];

For Example:-
int employees[] = new int[10];

- ✓ You can also declare and initialize arrays in the same statement.

For Example:-
String designations[] = {"Asst Vice President", "Vice President"};
Accessing values from various Elements of an Array - You can access values from elements in the array by referring to the element by its index number.

For Example:-
String designations[];
designations = new String[3];
designations[1] = "Asst Vice President";
designations[2]= "Vice President";
designations[0]=designations[2];
In the above example, the value of the third element of the array is assigned to the first element of the array.

Declaration of an array:
intnum[]; or intnum = new int[2];

Sometimes user declares an array and its size simultaneously. You may or may not be define the size in the declaration time. such as:

intnum[] = {50,20,45,82,25,63};

In this program we will see how to declare and implementation. This program illustrates that the array working way. This program takes the numbers present in the num[] array in unordered list and prints numbers in ascending order. In this program the sort() function of the java.util.*; package is using to sort all the numbers present in the num[] array. The Arrays.sort() automatically sorts the list of number in ascending order by default. This function held the argument which is the array name num.

Here is the code of the program:-

```
public class  array{
  public static void main(String[] args){
       int num[] = {50,20,45,82,25,63};
       int l = num.length;
       int i;
       System.out.print("Given number : ");
      for (i = 0;i < l;i++ ){
          System.out.print("  " + num[i]);
      }
   }
}
```

String array

```
class StringCharacter
{
  static String[] names={"nitai","gour","gopi","alar","hari"};
  public static void main(String args[]){
  for(int i=0;i<5;i++){
  System.out.println(names[i]);
  }
}
}
```