

CC – V – PROGRAMMING IN JAVA 18UCS5

Objective:

- To understand the basic concepts of Object Oriented Programming with Java language
- Enable to write the basic programmes

UNIT I

Fundamentals of Object Oriented Programming – Java Evolution – Overview of Java Language – Data Types , Variables , Arrays – Operators – Control Statements.

UNIT II

Introduction to Classes – Class fundamentals – Declaring Objects – Constructors – Methods – Overloading Methods – Nested and Inner Classes – String Handling.

UNIT III

Inheritance – Method Overriding – Abstract Class – Packages – Interfaces – Exception Handling – Types Of Exception – Try And Catch – Nested Try Statements.

UNIT IV

Multithreaded Programming – Stream I/O and Files: Java I/O Classes and Interfaces – File – Stream Classes – Byte Streams – Character Streams – Using Stream I/O – Serialization – Stream Benefits.

UNIT V

Applets and Graphics: Fundamentals of Applets – Graphics – AWT and Event Handling: AWT Components and Event Handlers – AWT Controls and Event Handling Types and Examples.

Outcome:

- Able to write the JAVA programmes

TEXT BOOK

Programming With Java A Primer 3/E E. Balaguruswamy

UNIT I: Chapter 1 to 7

UNIT II: Chapter 8, 9

UNIT III: Chapter 10, 11, 13

UNIT IV: Chapter 12, 16

UNIT V: Chapter 14, 15

REFERENCE BOOK

Programming With Java – C. Muthu

- <http://www.learnjavaonline.org/>

Part – A Answer all the Questions 10 X 2 = 20 Marks	Part – B Internal Choice Type 5 X 5 = 25 Marks	Part – C Answer any 3 Questions 3 X 10 = 30 Marks
Question 1,2 – I Unit 3,4 – II Unit 5,6 – III Unit 7,8 – IV Unit 9,10 – V Unit	11a (or) 11b – I Unit 12a (or) 12b – II Unit 13a (or) 13b – III Unit 14a (or) 14b – IV Unit 15a (or) 15b – V Unit	16 – I Unit 17 – II Unit 18 – III Unit 19 – IV Unit 20 – V Unit

Fundamentals of Object-Oriented Programming

Definition of Object-Oriented Programming

Object-Oriented Programming is an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand.

Object-Oriented Paradigm

- The major objective of Object-Oriented approach is to eliminate some of the flaws encountered in the Procedural approach.
- OOP allows us to decompose a problem into a number of entities called Objects.
- The combination of data and methods make up an object.

Some of the features of Object-Oriented Paradigm are as follows:

- Emphasis is on data rather than procedure,
- Programs are divided into what are known as Objects.
- Data is hidden and cannot be accessed by external functions.
- Objects may communicate with each other through methods.
- New data and methods can be easily added whenever necessary.
- Data Structures are designed such that they characterize the objects.
- Follows bottom-up approach in program design.
- Methods that operate on the data of an object are tied together in the data structure.

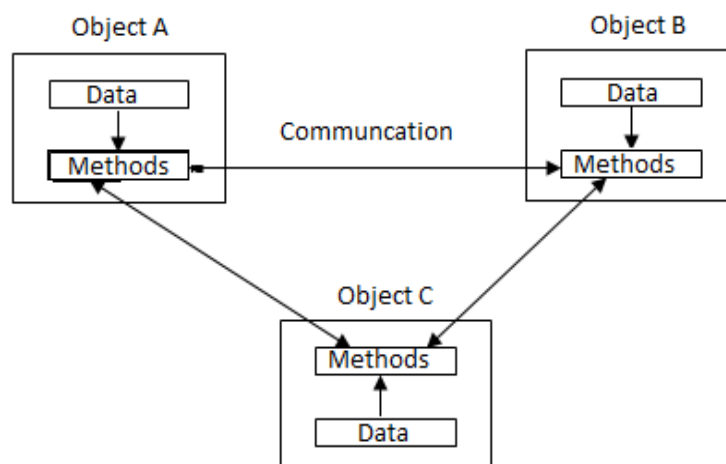


Fig. Organization of data and methods in OOP

Object Oriented Programming Concepts (OR) Object Oriented Programming Principles:

When we represent the data in Object Oriented Programming language we get the security. Examples of Object Oriented Programming Languages are LISP, ADA, ALGOL, SMALLTALK, OBJECT COBAL, OBJECT PASCAL, Cpp, JAVA, DOT NET, etc. In order to say any language is an Object Oriented Programming Language it has to satisfy the following principles of OOPs.

OOP Principles

- Class
- Object
- Data Abstraction and Data Encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding
- Message Passing

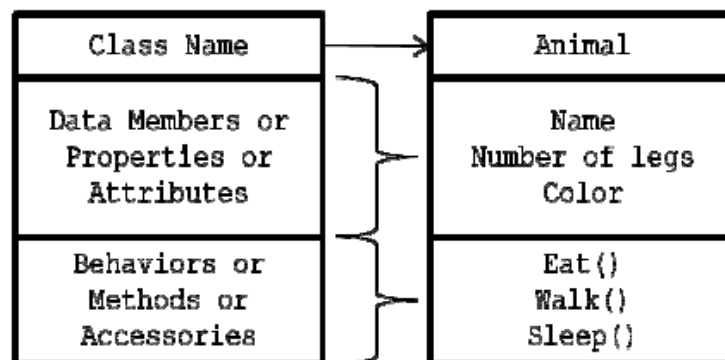
Class:

“A Class is a way of binding the data and associated methods in a single unit”. Any JAVA program if we want to develop then that should be developed with respect to Class only i.e., without Class there is no JAVA program.

The entire set of data and code of an object can be made a user defined data type using the concept of a class. In fact Objects are treated as variables of the type Class. Once a Class has been defined, we can create any number of objects belonging to that Class.

Class Diagram for defining a class:

Example:



Syntax for defining a class:

```

Class <clsname>
{
    Variable Declaration;
    Methods Definition;
}

```

Object:

Objects are the basic runtime entities in an object-oriented system. In order to store the data for the data members of the class, we must create an object.

Definitions of an Object

1. Instance (instance is a mechanism of allocating sufficient amount of memory space for data members of a class) of a class is known as an object.
2. Class variable is known as an object.
3. Blue print of a class is known as an Object.
4. Real world entities are called as Objects.

The following figure is the representation of an object

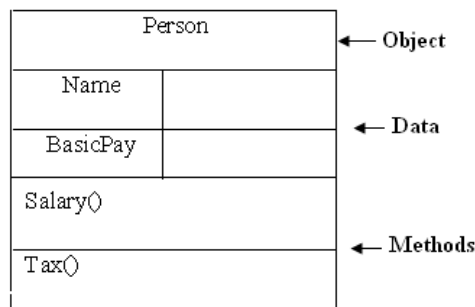


Fig: Representation of an Object

Data Abstraction and Data Encapsulation:**Data Abstraction:**

- Abstraction and Encapsulation in Java are two important Object oriented programming concept and they are completely different to each other.
- Data abstraction is a mechanism of retrieving the essential details without dealing with background details.
- Abstraction represent taking out the behavior from how exactly it's implemented.

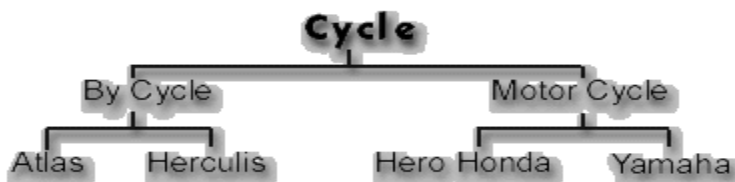
Data Encapsulation:

- Encapsulation means hiding details of implementation from outside world so that when things change nobody gets affected.
- The wrapping up of data and methods into a single unit is known as Encapsulation. The data is not accessible to the outside world and only those methods, which are wrapped in the class, can access it.

Inheritance:

Inheritance is the process of by which objects of one class acquire the properties of objects of another class.

Inheritance supports the concept of hierarchical classification. The concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it. The following is an example for the inheritance concept.



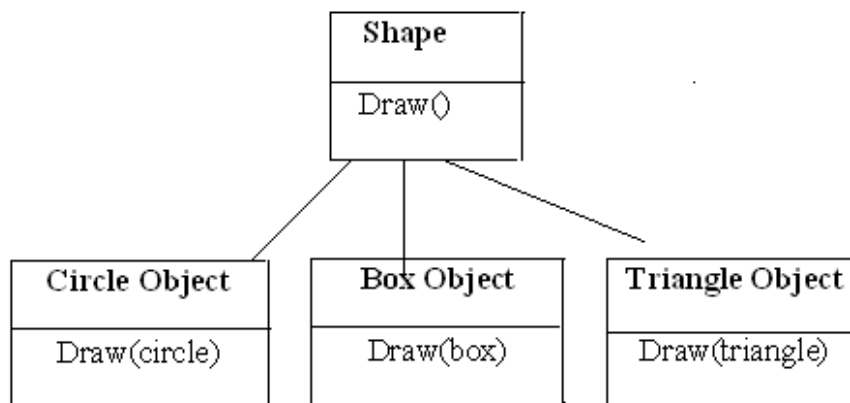
Example 1



Example 2

Polymorphism:

Polymorphism means the ability to take more than one form. For example, an operation may exhibit different behavior in different instances. The behavior depends upon the type of data used in the operation. The following is an example for polymorphism concept.



Dynamic Binding:

Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at runtime.

Message passing:

The Object Oriented Program consists of a set of objects that communicate with each other. Objects communicate with one another by sending and receiving information. It involves the following basic steps.

- Creating classes that define objects and their behavior.
- Creating objects from class definitions.
- Establishing communication among objects.

Exchanging the data between multiple objects is known as Message Passing. Objects communicate with one another by sending and receiving information much the same way as people pass message to one another.

Message passing involves specifying the name of the object, the name of the method (message) and the information to be sent. Example

employee.salary (name);

object _____ ↑ message ↑ _____ information

Objects have a life cycle. They can be created and destroyed. Communication with an object is feasible as long as it is alive.

Benefits of OOP:

OOP offers several benefits to the program designer and the user. The principal advantages are:

- Through inheritance, we can eliminate redundant code and extend the use of existing classes.

- We can build programs from standard working modules that communicate with one another rather than, having to start writing the code from scratch. This leads to saving of development time and higher productivity.
- The principle of data hiding helps the programmers to built secure program.
- It is possible to have multiple objects to coexist without any interference.
- It is easy to partition the work in a project based on objects.
- Object-oriented systems can be easily upgraded from small to large system.
- Message passing technique for communication between objects makes the interface descriptions with external system much simpler.
- Software complexity can be easily managed.

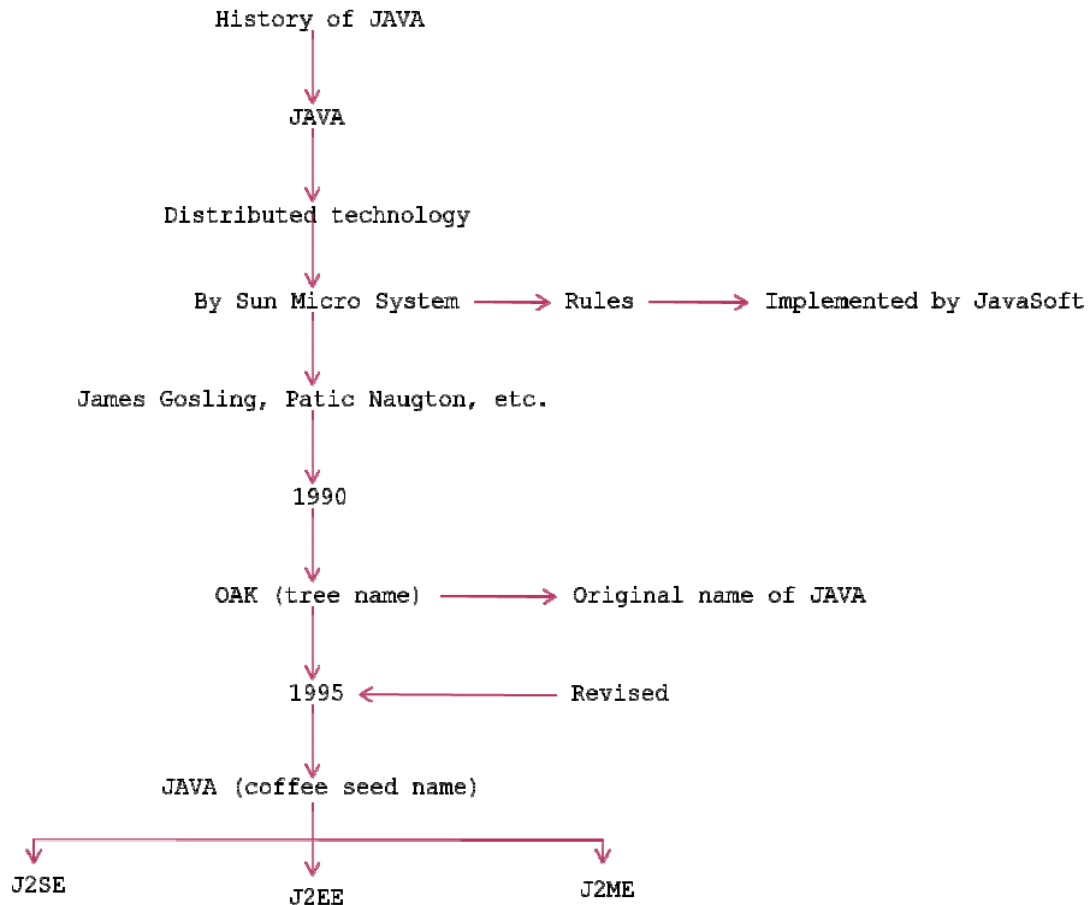
Applications of OOP

The following are some of areas of Object Oriented Programming

- Real-time systems
- Simulation and modeling
- Object oriented databases
- Hypertext, hypermedia and expert text.
- AI and expert systems.
- Neural networks and parallel programming.
- Decision support and office automation systems.

Java Evolution

History of Java



1. Java is a programming language used to develop distributed applications.
2. Java was developed at SUN Micro System INC USA by a person called James Gosling and others in the year of 1990.
3. Originally Sun Micro System is one of the Academic University (Stanford University of Networks) developed “rules” for development of java and those rules are programmatically implemented by Java Soft INC USA. Java Soft is one of the software divisions of Sun Micro System.
4. The original name of java language is OAK (Scientifically it is a tree name).
5. The software OAK was not fulfilling or not solving industry problems or demands.
6. Hence the software OAK was revised in the year of 1995 and released to the industry on the name of Java.

JAVA released to the market in three categories J2SE (JAVA 2 Standard Edition), J2EE (JAVA 2 Enterprise Edition) and J2ME (JAVA 2 Micro/Mobile Edition).

- J2SE is basically used for developing client side applications/programs.
- J2EE is used for developing server side applications/programs.
- J2ME is used for developing mobile or wireless applications.

Why the java is a platform independent language?

The most striking feature of the language is that it is a platform-independent language.

Reason

Microsoft System has developed a technology called DOT NET and Sun Micro System has developed a technology called JAVA. Both these technologies are called distributed technologies.

The technology DOT NET will run only on that operating system's which are provided by Microsoft. Hence DOT NET technology is platform dependent technology. Whereas, the technology called JAVA will run on all operating systems irrespective of their providers. Hence JAVA is called platform independent technology.

Features or Buzzwords of JAVA

The following are the features of the JAVA language.

1. Compiled and Interpreted.
2. Simple and Small.
3. Platform Independent.
4. Architectural Neutral.
5. Portable.
6. Multithreaded.
7. Distributed.
8. Robust
9. Secure.
10. High Performance.
11. Interpreted.
12. Dynamic.
13. Object Oriented and these are explained below.

Compiled and Interpreted

Usually a computer language is either compiled or interpreted. Java combines both these approaches thus Java is a two-stage system.

First, Java compiler translates source code into what is known as byte code instructions. Byte codes are not machine instructions.

In the second stage, the Java interpreter generates machine code that can be directly executed by the system.

Thus Java is both compiled and interpreted language.

Simple and Small

Java is small and simple language. For example, Java does not use pointers, preprocessor header files, goto statement and many others. It also eliminates operator overloading and multiple inheritance.

****Platform Independent**

A program or technology is said to be platform independent if and only if which can run on all available operating systems.

The language JAVA will have a common data types and the common memory spaces on all operating systems. The JAVA software contains the special programs which converts the format of one operating system to another format of other operating system. Hence JAVA language is treated as platform independent language.

****Architectural Neutral**

A language or technology is said to be architectural neutral which can run on any available processors in the real world. The languages like C, C++ are treated as architectural dependent. The language Java can run on any of the processor irrespective of their architecture and vendor.

Portable

A portable language is one which can run on all operating systems and on all processors irrespective of their architectures and providers. The languages like C, C++ are treated as non-portable languages whereas the language Java is called Portable language.

Portability= Platform Independent + Architectural Neutral

Multithreaded

Multithreaded means handling multiple tasks simultaneously. Java supports multithreaded programs. This means that we need not wait for the application to finish one task before beginning another. For example, we can listen to an audio clip while scrolling a page and at the same time download an applet from a remote computer.

Distributed

Java is designed as a distributed language for creating the applications on networks. It has the ability to share both data and programs. Java programs can open and access remote objects on Internet as easily as they can do in a local system.

Robust

Java is one of the Robusted programming language. Since java programming language effectively addresses the run time errors. In Java programming runtime errors are known as Exceptions.

The languages like C, C++ are treated as week programming languages (Not Robust), since they are unable to deal with runtime errors. Runtime errors are occurred when we enter invalid input.

Note:

Compile time errors are occurring when the programmer is not following syntax rules of the programming language.

Secure

Java is one of the most secured programming languages. To provide the security to our java real time applications, we need not to write our own security code. Since java library (API) contains readily available security programs for protecting confidential information from unauthorized users.

Hence java is one of the powerful secured programming language.

High Performance

Java is one of the High Performance programming language because of the following reasons.

- Automatic memory management (Garbage Collector).
- Magic of byte code (Execution of the java application is very faster compared to other programming language applications).
- According to industry experts java programmer performance is high because java programming environment is free from pointers. Hence development of an application takes less time.

Interpreted

In the older versions of java, compilation phase is so fast than the interpretation phase. It was the industry complaint on older versions of java.

In the newer versions of java, to speed up the interpretation phase, sun micro system had developed JIT (Just in Time) compiler and added to JVM.

In the current versions of java interpretation phase is so faster than compilation phase. Hence java is one of the highly interpreted programming language.

Dynamic

In any programming language memory can be allocated in two ways. They are

- Static Memory Allocation
- Dynamic Memory Allocation

Java programming does not follow the static memory allocation but it always follows dynamic memory allocation.

Dynamic memory allocation is one in which memory will be allocated at run time. In java programming to allocate the memory space dynamically we use an operator called “new”. “new” operator is known as dynamic memory allocation operator.

Object Oriented

Any programming language that satisfies all the principles of OOP is called as an Object Oriented Programming Language. The Java language satisfies all the principles of OOP. Hence it is called an Object Oriented Programming Language. Java is a true object oriented language. Almost everything in java is an object.

Differences between Java and C

The major difference between Java and C is that Java is an Object Oriented Programming Language whereas the C is a Procedure Oriented Programming Language.

C	Java
Structures are supported	Structures are not supported
Unions are supported	Unions are not supported
Storage classes such as automatic, register, external are supported	Storage classes are not supported
Type definition is supported	Type definition is not supported
Size of() operator is supported	Size of() operator is not supported
Pre-processor directives such as # define, #include are supported	Pre-processor directives are not supported

Differences between Java and C++

The major difference between the Java and C++ is that, Java is a pure object oriented programming language whereas C++ is a partially object oriented programming language.

C++	Java
C++ is not a purely object-oriented programming language, since it is possible to write C++ programs without using a class or an object	Java is purely object programming language. since it is not possible to write a java program without using atleast one class
Pointers are available in C++	We cannot create and use pointers in java
Allotting memory & deallocating memory is the responsibility of the programmer	Allocation & deallocation of memory will be taken care of by JVM
C++ has goto statement	Java does not have goto statement
Multiple Inheritance feature is available in C++	No multiple Inheritance in java
Operator overloading is available in C++	It is not available in java
#define, typedef and header files are available in C++	#define, typedef and header files are available in C++
There are 3 Access specifiers in C++ private,public & and protected	Java supports 4 access specifiers private,public,protected and default
There are constructors and destructors in C++	Only constructors are there in java, No destructors are available in this language

Java and Internet

- The first application program written in java was Hot Java. Hot Java is a Web Browser used to run applets on Internet.
- Internet users can use Java to create the applet programs and run them locally by using a web browser.
- Internet users can also download the applet from a remote computer as shown in below.

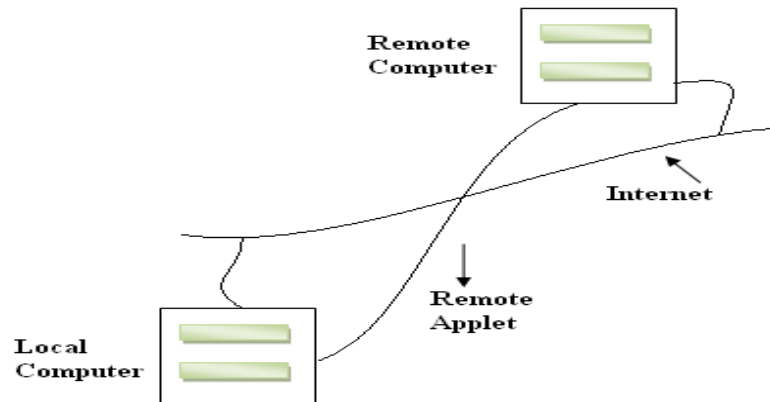


Fig: Downloading of applets via internet

- Due to this, Java is popularly known as Internet Language.

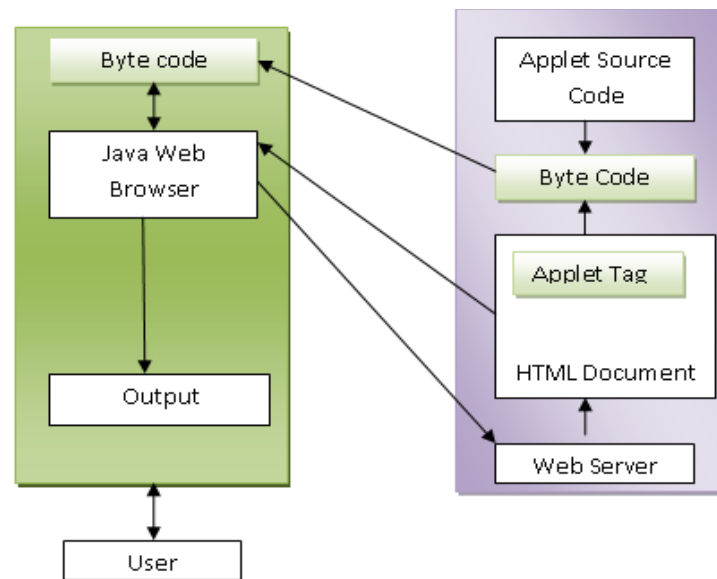
Java and World Wide Web

World Wide Web (WWW) is an open ended information retrieval system designed for internet distributed environment. This system contains the web pages that provide both information and control. We can navigate to a new document in any direction.

Java and Web share the same philosophy, Java could be easily incorporated into the web system. Before Java the WWW was limited to display the images and texts. However, the incorporation of Java into Web pages made it capable of supporting animation, graphics, games, and a wide range of special effects.

Java communicates with a Web page through a special tag called `<APPLET>`. The following are the steps that illustrate the communication steps.

- The user sends a request for an HTML document to the remote computer's Web server. The Web server is a program that accepts a request, processes the request, and sends the requested document.
- The HTML document is returned to the user's browser. The document contains the `APPLET` tag, which identifies the applet.
- The corresponding applet byte code is transferred to the user's computer. This byte code had been previously created by the Java compiler using the Java source code file for that applet.
- The Java enabled browser on the user's computer interprets the byte codes and provide output.
- The user may have further interaction with the applet but with no further downloading from the provider's Web server. This is because the byte code contains all the information necessary to interpret the applet.



Web Browsers

Web browsers are used to navigate through the information found on the net. They allow us to retrieve the information spread across the Internet and display it using the hypertext markup language (HTML). Examples for Web browsers are as follows.

- Hot Java
- Netscape Navigator
- Internet Explorer

Hot Java

When the Java language was first developed and ported to the Internet, no browsers were available that could run Java applets.

Hot Java is the web browser from Sun Microsystems that enables the display of interactive content on the Web, using the Java language. Hot Java is entirely written in Java language and demonstrates the capabilities of Java Programming Language.

Netscape Navigator

Netscape Navigator, from Netscape Communications Corporation, is a general-purpose browser that can run Java applets.

Netscape Navigator has many useful features such as visual display about downloading process and indication of the number of bytes downloaded.

Internet Explorer

Internet Explorer is another popular browser developed by Microsoft for Windows 95, NT and XP Workstations. Both the Navigator and Explorer use tool bars,

icons, menus and dialog boxes for easy navigation. Explorer uses a just-in-time (JIT) compiler which greatly increases the speed of execution.

Java Environment:

Java Environment includes a large number of development tools and hundreds of classes and methods. The development tools are part of the system known as Java Development Kit (JDK) and the classes and methods are part of the Java Standard Library (JSL), also known as the Application Programming Interface (API).

Java Development Kit (JDK):

The Java Development Kit comes with a collection of tools that are used for developing and running Java programs. They include

Applet viewer : Enables us to run Java applets.

java (Java Interpreter) : Java Interpreter, which runs applets and applications by reading and interpreting bytecode files.

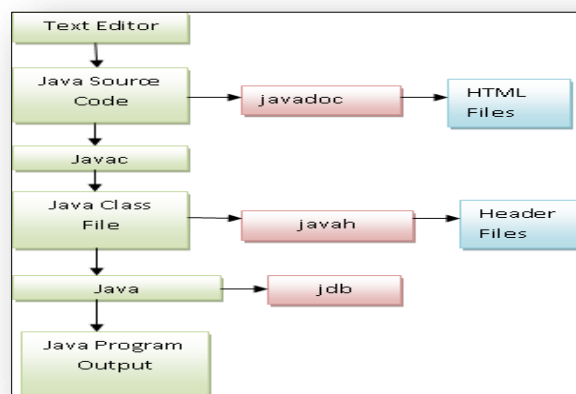
javac (Java Compiler) : The Java Compiler, which translates Java source code to bytecode files that the interpreter can understand.

Javadoc (for creating HTML documents): Creates HTML- format documentation from Java source code files.

javah (for C header files): Produces header files for use with native methods.

javap (Java disassemble): Which enables us to convert bytecode files into a program description.

jdb (Java debugger) : Which helps us to find errors in our programs.



Application Programming Interface:

The Java Standard Library includes hundreds of classes and methods grouped into several functional packages. Most commonly used packages are

- **Language Support Package:** A collection of classes and methods required for implementing basic features of java.
- **Utility Package:** A collection of classes to provide utility functions such as date and time functions.
- **Input/output Package:** A collection of classes required for input/output manipulations.
- **Networking Package:** A collection of classes for communicating with other computers via Internet.
- **AWT Package:** The Abstract Window Tool Kit package contain classes that implements platform independent graphical user interface.
- **Applet Package:** This includes a set of classes that allows us to create Java applets.

Java Runtime Environment:

The Java Runtime Environment (JRE) facilitates the execution of java programs. It primarily comprises of the following:

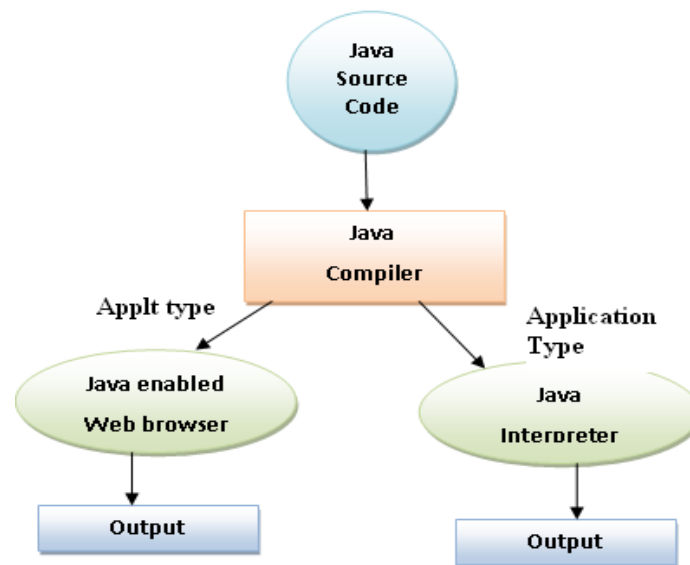
- **Java Virtual Machine (JVM):** It is a program that interprets the intermediate Java byte code and generates the desired output.
- **Runtime class libraries:** These are a set of core class libraries that are required for the execution of java programs.
- **User interface tool kit:** AWT and Swing are example of toolkits that support varied input methods for the users to interact with the application program.
- **Deployment technologies:** JRE comprises the following key deployment technologies.
 - **Java plug-in:** Enables the execution of a Java applet on the browser.
 - **Java Web Start:** Enables remote-deployment of an application.

Chapter-3

Overview of Java Language

Java is a general purpose, Object-Oriented programming language. We can develop two types programs. They are

- Stand- Alone applications
- Web-Applications



Q. Explain the Structure of a java program

To write a java program, we first define classes and then put them together. A Java program may contain one or more sections as shown in below.

Documentation Section	← Suggested
Package Statement	← Optional
Import statement	← Optional
Interface statements	← Optional
Class Definitions	← Optional
Main method class { Main method definition }	← Essential

Documentation Section

The documentation section contains a set of comment lines giving the name of the program, the author and other details.

There are three types of comments in java – single line, multi line, and Java documentation.

➤ Single line Comments

These comments are for making a single line as a comment. These comments start with double slash symbol (`//`) and after this, whatever is written till the end of the line is taken as a comment.

Example: `// This is my comment of one line`

➤ Multi line comments

These comments are used for representing several lines as comments. These comments start with `/*` and end with `*/`. In between `/*` and `*/`, whatever is written is treated as a comment.

Example:

```
/* This is a first line
   This is second line */
```

➤ Java Documentation Comments

These comments start with `/**` and end with `*/`. These comments are used to provide description for every feature in a java program.

Package Statement

The first statement allowed in a java file is a **package** statement. This statement declares a **package** name and informs the compiler that the classes defined here belong to this package. The package statement is optional.

Example

```
package student;
```

Import Statement

The next statement after a package statement may be a number of **import** statements. This is similar to the **#include** statement in C. Using **import** statements; we can access to classes, that are part of other named packages.

Example: import student.test;

This statement instructs the interpreter to **load** the **test** class contained in the package **student**.

Interface Statements

An interface statement is like a class, but includes a group of method declarations. This is also an optional section and is used only when we wish to implement the multiple inheritance features in the program.

Class Definitions

A Java program may contain multiple class definitions. Classes are the primary and essentials of a Java program.

Main Method Class

Since every Java stand-alone program requires a **main** method as its starting point, this class is the essential part of a java program. A simple java program may contain only this part. The **main** method creates objects of various classes and establishes communications between them. On reaching the end of **main**, the program terminates and the control passes back to the operating system.

Q. Explain a Simple java program

The following is a simple java program

```
//Sample.java
Class Sample
{
    Public static void main (String args [])
    {
        System.out.println("Welcome to Java Programming Language.....");
    }
}
```

```
    }  
}
```

The above simple java program contains the following..

Class Declaration

The first line

```
class Sample
```

declares a class, which is an object oriented construct. **class** is a keyword used to declare a new class definition. **Sample** indicates the name of the class.

Opening Brace

Every class definition in java begins with an opening brace “{” and ends with a matching closing brace “}”.

The Main Line

The third line

```
public static void main(String args[])
```

defines a method named **main**. Every Java application program must include the **main ()** method. This is the starting point of the interpreter to begin the execution of the program.

A Java program can have any number of classes but only one of them must include a main method to initiate the execution.

This line indicates a number of keywords, public , static and void

public:	The keyword public is an access specifier that declares the main method is accessible to all other classes.
static	Static is reserved keyword which means that a method is accessible and usable even though no objects of the class exist.
void	This keyword states that the main method does not return any value.

The Output line

The executable statement in this program is,

```
System.out.println(" Welcome to Java Programming Language.....");
```

- System: It is name of Java utility class.
- out: It is an object which belongs to System class.
- Println(): It is utility method name which is used to send any String to console.

Welcome to Java Programming Language.....

The method **println()** always append a new line character to the end of the string.

Q. Explain about the Java Tokens

The smallest individual units in a program are called as tokens. The compiler recognizes them for building up expressions and statements.

In simple terms, a java program is a collection of tokens, comments and white spaces. Java language includes five types of tokens. They are

- Reserved keywords
- Identifiers
- Literals
- Operators
- Separators

Reserved Keywords

Keywords are an essential part of a language definition. They implement the specific features of the language. The Java language has reserved 50 words as keywords as shown in the below table.

Since the keywords have specific meaning in java, we cannot use them as names for variables, classes, methods and so on. All keywords are to be written in lower-case letters.

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

Identifiers

Identifiers are program designed tokens. They are used for naming classes, methods, variables, objects, labels, packages and interfaces in a program. Java identifiers follow the following rules

- They can have alphabets, digits and the underscore (_) and dollar (\$) sign characters.
- They must not begin with a digit.
- Uppercase and lowercase letters are distinct.
- They can be of any length.

Literals

Literals in java are a sequence of characters that represent constant values to be stored in variables. Java language specifies five major types of literals. They are

- Integer literals
- Floating point literals
- Character literals
- String literals
- Boolean literals

Operators

An operator is a symbol that takes one or more arguments and operates on them to produce a result.

Separators

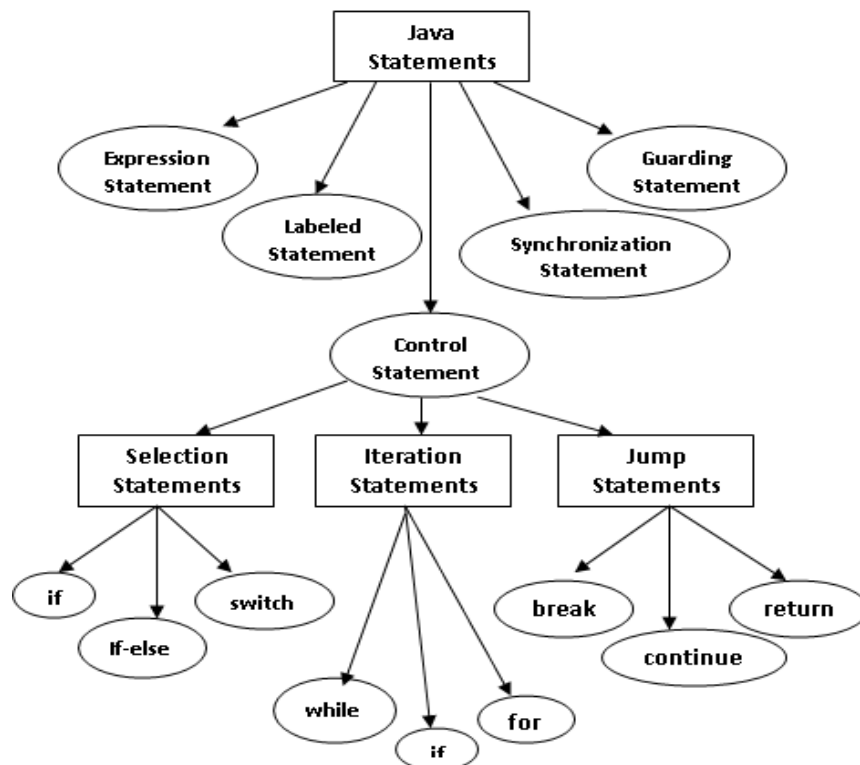
Separators are symbols used to indicate where groups of code are divided and arranged. They basically define the shape and function of our code. The following are the list of separators and their functions.

Name	Its function
Parentheses ()	Used to enclose the parameters in method definition and invocation, also used for defining precedence in expressions.
Braces {}	Used to contain the values of automatically initialized arrays and to define a block of code for classes, methods and local scopes
Brackets []	Used to declare array types and for dereferencing the array values.
Semicolon ;	Used to separate the statements
Comma ,	Used to separate the consecutive elements in a variable declaration
Period .	Used to separate package names from sub-packages and classes, also used to separate a variable method from a reference variable.

Q. Explain about the Java Statements?

The statements in java are like sentences in natural languages. A statement is an executable combination of tokens ending with a semicolon (;) mark. Statements are usually executed in the sequence in the order in which they appear.

The following diagram illustrates the types of java statements.



Expression Statements

Most statements are expression statements. Java has seven types of Expression statements.

- Assignment
- Pre-increment
- Pre-decrement
- Post-increment
- Post-decrement
- Method call
- Allocation expression

Labeled Statement

Any statement may begin with a label. Such labels must not be keywords. In java they are used as the arguments of jump statements.

Control Statements

In java, it is possible to control the flow of execution. Java language provides three types of control structures. Those are discussed below

➤ Selection Statement

These select one of several control flows. There are three types of statements in java. Those are **if, if-else and switch**

➤ Iteration Statement

These specify how and when looping will take place. There are three types of iteration statements in java. Those are **while, do and for**.

➤ Jump Statement

Jump statements pass the control to the beginning or end of the current block, or to a labeled statement. The four types of jump statements are **break, continue, return and throw**.

Synchronization Statement

These are used for handling issues with multithreading.

Guarding Statement: Guarding statements are used for safe handling of code that may cause exceptions. These statements use the keywords **try, catch and finally**.

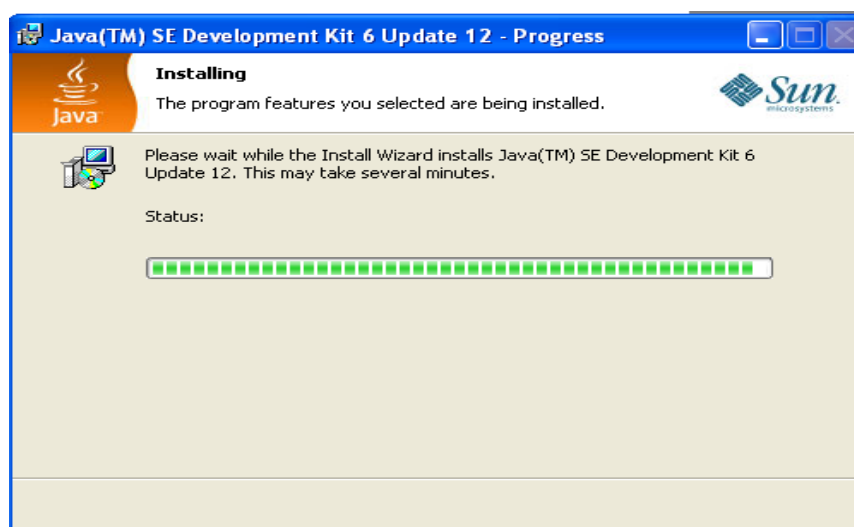
Q. Explain the steps for installing and configuring the java.

The following are the steps to install a java software in our system.

1. Double click the .exe file to initiate the installation procedure. The welcome screen appears as shown in below.



2. The welcome screen shows the licensing terms and conditions. Click the Accept button to begin the Java installation. The progress screen appears as shown in below.



- The progress screen depicts the percentage of installation that has been completed. Once the installation is complete, the complete screen appears as shown in below.

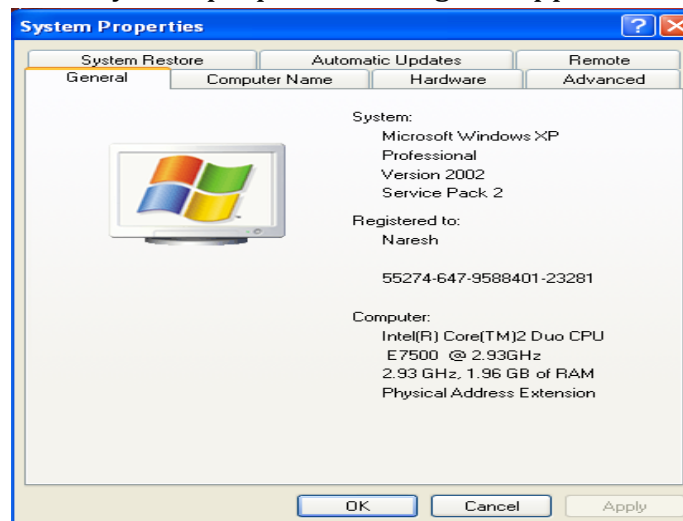


- The complete screen depicts the successful installation of java on the computer system. Click the Finish button to end the installation process.

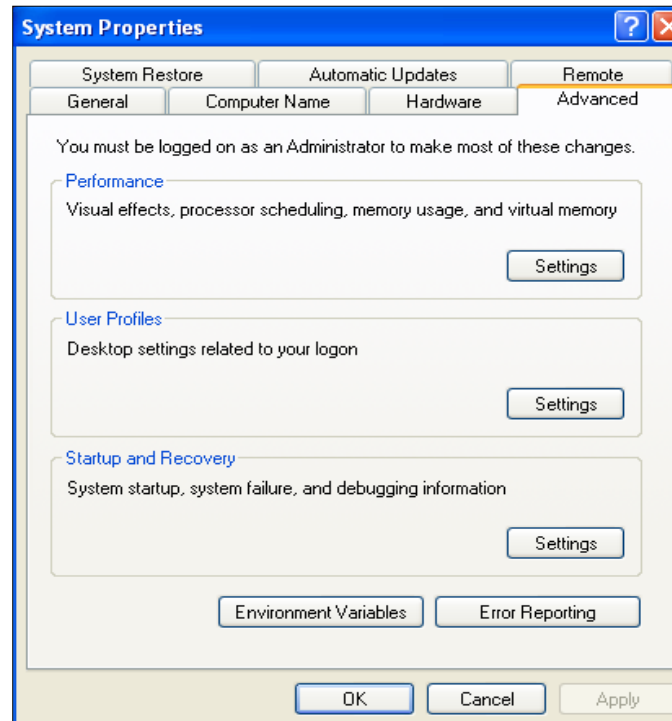
Configuring Java

Once Java is installed, we need to configure it by adding the Java path to environment variable, PATH. The following are the steps to set the PATH variable to the Java directory.

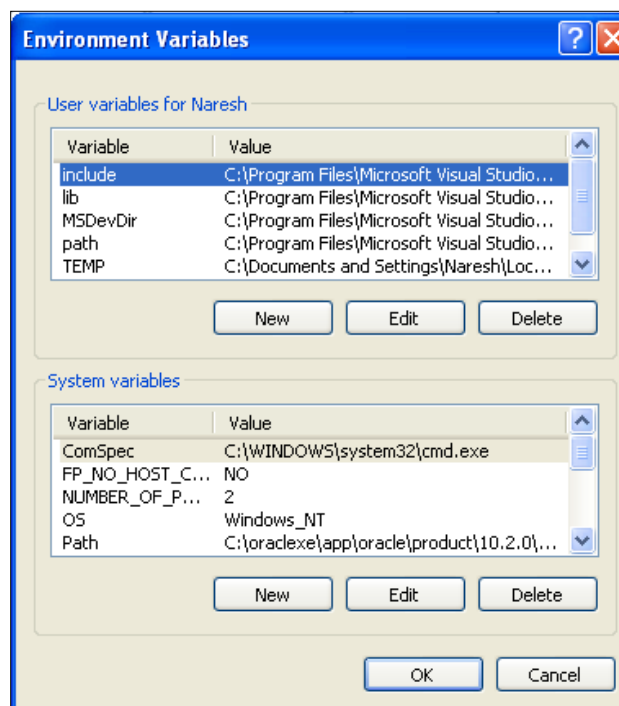
- Right click on the My Computer icon and select the properties option from the drop down menu. The system properties dialog box appears as shown in below.



2. Select the Advanced tab to display the advanced tab page, as shown in below.

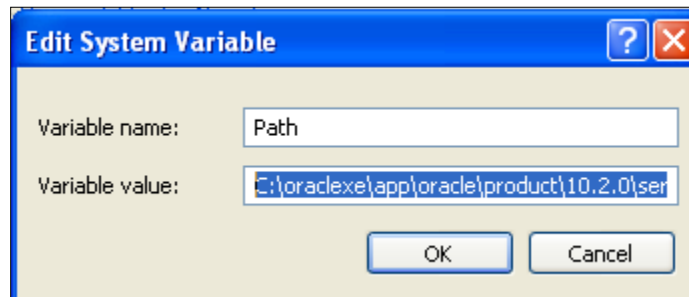


3. Click the Environmental Variables button to display the Environmental variables dialog box as shown in below.



4. The Environmental dialog box is divided into two sections.
- User variables
 - System Variables.

Under the Systems Variable section, select the path option below the variable column and click the Edit button. The Edit System Variable dialog box appears as shown in below.



5. By default the path variable is already set to multiple locations. To set the Java directory path to the Path variable, append the directory path in the variable value text box, separated by a semi-colon, as shown in below.

Q. Explain the procedure for implementing a java program

Implementation of a Java application program involves a series of steps. They include

- Creating the program
- Compiling the program
- Running the program

Creating the program

Consider the following program

```
//Test.java
class Test
{
    public static void main(String args[])
    {
        System.out.println("Hellow !");
    }
}
```

```

        System.out.println("Welcome to the world of Java...");
    }
}

```

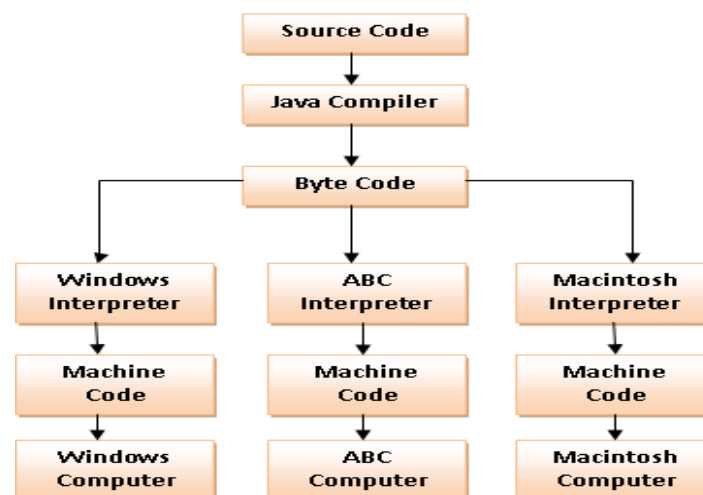
Save this program using a file name Test.java. This file is called the source file. All the java source files will have the extension **.java**.

Compiling the program

To compile the java program, we must use the java compiler **javac** along with the source file name on the command line as shown in below.

```
C:/> javac Test.java
```

If everything is OK, the java compiler creates a **.class** file containing the byte code of the program (Test.class).



Running the program

To run a stand-alone application we need to use the java interpreter on the command prompt as shown in below.

```
C:/> java Test
```

Now, the interpreter looks for the main method in the program and begins execution from there. When the program is executed the above program displays the output...

Hellow !

Welcome to the world of Java...

Q. Explain about the Java virtual machine

Java Virtual Machine (JVM) is the heart of the entire Java program execution process. It is responsible for taking the **.class** and converting each byte code instruction into the machine language instruction that can be executed by the microprocessor.

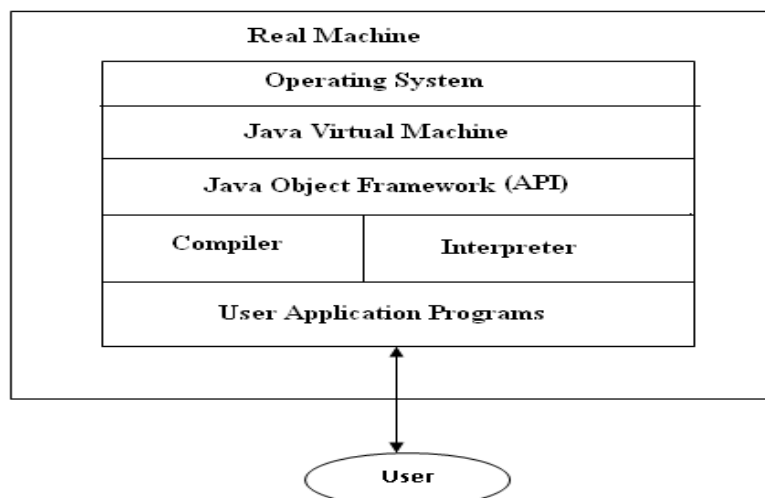
First of all, the **.java** program is converted into a **.class** consisting of byte code instructions by the java compiler for a machine called Java Virtual Machine as shown in below. The JVM exists only inside the computer memory.



The byte code is not a machine specific code (machine code). The machine code is generated by the Java Interpreter by acting as an intermediary between the virtual machine and the real machine as shown in below.



The following figure illustrates how Java works on a typical computer. The Java Object Framework (Java API) acts as the intermediary between the user programs and the virtual machine and which in turn acts as the intermediary between the operating system and the java object framework.



C:\> java CommandLine Simple Object_Oriented Portable Distributed

Output

Number of arguments =4

1 : Java is Simple !

2 : Java is Object_Oriented !

3 : Java is Portable !

4 : Java is Distributed

Q. Why java is called as a freeform language (OR) explains the programming style of a java program

Java is a freeform language, because we need not have to indent any lines to make the program work properly. Java system does not care where on the line we begin typing. The following is an example.

The statement,

```
System.out.println("Java is wonderful ! ");
```

Can be written as

```
System  
.  
out.  
println  
(  
"Java is wonderful !"  
);
```

*******End of Chapter-3*******

Chapter-4

Constants, Variables and Data types

Constants in java

A constant is an identifier which takes a fixed value during the program execution same rules we should follow while declaring the constants of variables.

Types of constants:

We have 4 types of constants .which are given below.

1. Integer constants: A constants which is formed with integers including with zero, which can take positive and negative.

Ex: 123,-12, 0, 22

2. Floating point constants: A constants which is formed with real values including fractional part apart from integers

Ex: 12.5,-12.0, 3.14, 6.1

The real numbers are formed with two parts known as mantissa and exponent where mantissa is either decimal or integer and exponent is an integer either positive (or) negative

Ex: 26.506, 26.506+e-2, 26.506+e3

3) Character constants: A Character constant is a single character i.e., enclosed with a pair of single code

Ex: 'a', '8', '4', '_', 'c'

Every character is having integer values known as ASCII (American standard code for information interchange) values

4) String constants: A group of characters combined with the a pair of double codes is known as string constants.

Ex: "BVRICE", "INDIA"

Variables in java

A Variable is an identifier that takes different values at different times during the program execution. In java language a variable can be declared as follows

Syntax: Data type variable name:

Example:

- int marks, total;
- float average;
- char x;
- long area;
- double volume;

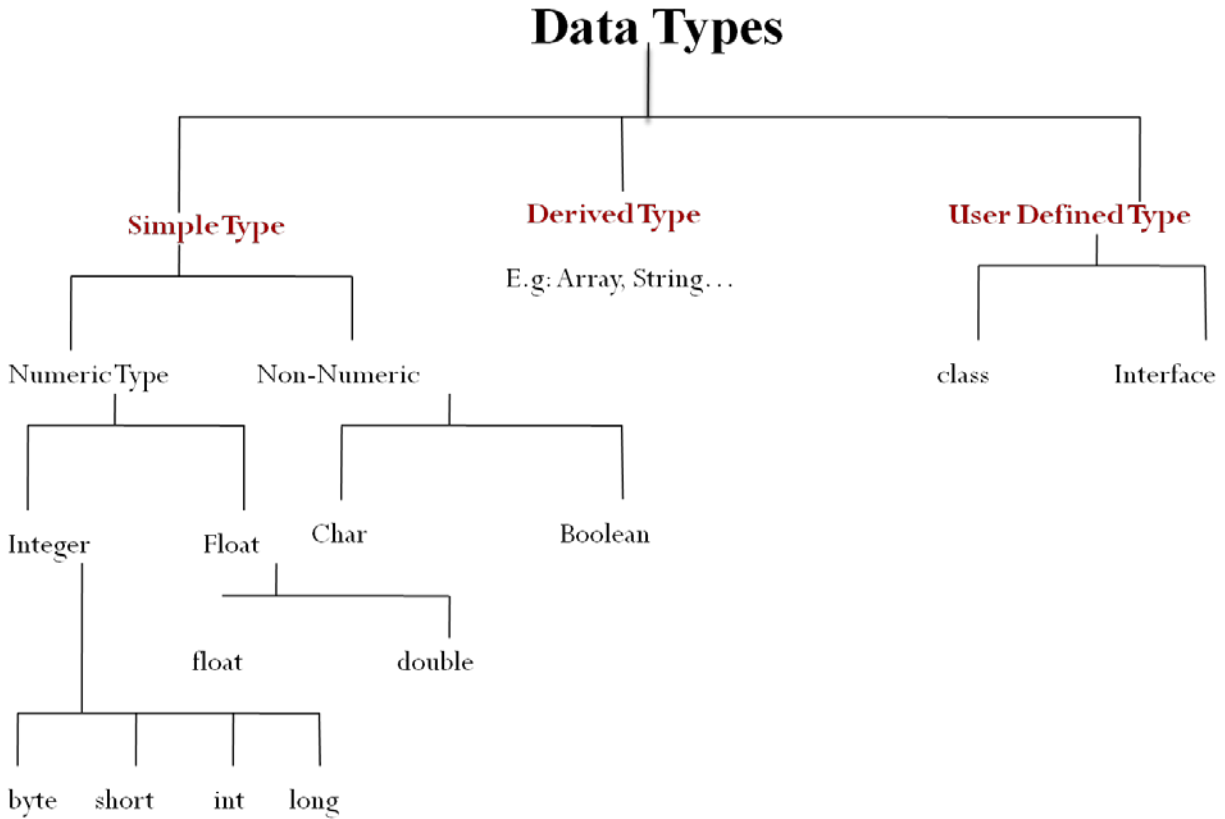
While declaring the variable we should follow some precautions which are given below.

Rules for defining variables

- 1) They must begin with a character including underscore.
- 2) They should not start with digits.
- 3) They can take any of length.
- 4) Keywords are not used in the variable names.
- 5) Upper case and lower case are distinct i.e., sum & SUM are not the same.

Data types in java

“Data types are used for representing the data in main memory of the computer.”



In java we have eight data types which are grouped in four groups. They are **Integer category Data types**, **Floating point data types**, **Character category data types**, **Boolean category data types**.

Integer category data types

They are used to represent integer data. This category of data type contains four data types. Which are given in the following table

Data type	Size	Range
byte	1	+127 to -128
short	2	+32767 to -32768
int	4	+2147483647 to -2147483648
long	8	

Whatever the data type we should not exceed the predefined value

Float category data types

Float category data types are used for representing the data in the form of scale, precision i.e., these category data types are used for representing float values.

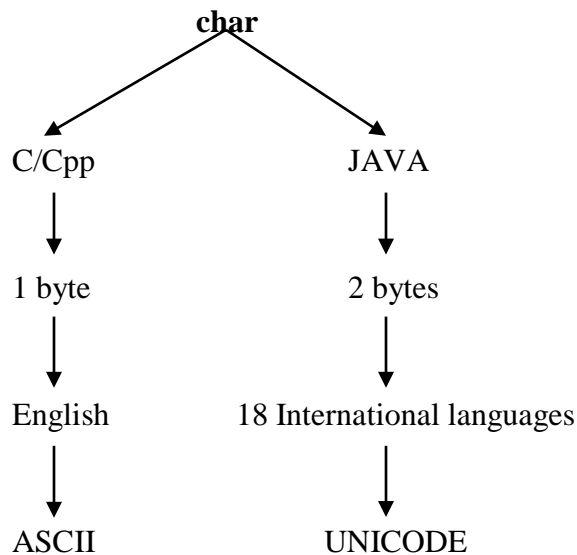
This category contains two data types they are given in the following table.

Data type	size	Range
Float	4	+2147483647 to -2147483648
Double	8	

Whenever we take any decimal constant directly in a Java program it is by default treated as highest data type in float category i.e., double.

Character category data types

- A character is an identifier which is enclosed within single quotes.
- In JAVA to represent character data, we use a data type called char. This data type takes two bytes since it follows UNICODE character set.



Data type	Size	Range
char	2	+32767 to -32767

UNICODE character set is one which contains all the characters which are available in 18 international languages and it contains 65536 characters.

Boolean category data types

- Boolean category data types is used for representing logical values i.e., TRUE or FALSE values.
- To represent logical values we use a keyword called **Boolean**.
- This data type takes 0 bytes of memory space.

NOTE: All keywords in JAVA must be written in **small letters only**.

Type casting in java

The process of converting one data type to another data type is called casting.

Examples:

```
int m=50;
```

```
byte n= (byte)m;
```

Java data type casting comes with 3 types

1. Implicit casting

2. Explicit casting

3. Boolean casting

Implicit casting (widening conversion):

A data type of lower size (occupying less memory) is assigned to a data type of higher size. This is done implicitly by the JVM. The lower size is widened to higher size. This is also named as **automatic type conversion**. Examples:

```
int x=10;           // occupies 4 bytes
```

```
double y=x;        // occupies 8 bytes
```

```
System.out.println(y); // prints 10.0
```

In the above example 4 bytes integer value is assigned to 8 bytes double value.

Explicit casting (narrowing conversion):

A data type of higher size (occupying more memory) cannot be assigned to a data type of lower size. This is not done implicitly by the JVM and requires **explicit casting**, a casting operation to be performed by the programmer. The higher size is converted to lower size.

```
double x=10.5;           // 8 bytes
int y=x;                 // 4 bytes; raises compilation error
```

In the above code, 8 bytes double value is converted to 4 bytes int value. It raises error. Let us explicitly type cast it.

```
double x=10.5;
int y=(int)x;
```

The double **x** is explicitly converted to int **y**. The thumb rule is, on both sides, the same data type should exist.

Boolean casting:

A boolean value cannot be assigned to any other data type. Except boolean, all the remaining 7 data types can be assigned to one another either implicitly or explicitly; but boolean cannot. We say, boolean is **incompatible** for conversion. Maximum we can assign a boolean value to another boolean.

Following raises error.

```
boolean x=true;
int y=x;           // error
boolean x=true;
int y=(int)x;     // error
```

byte -> short -> int -> long -> float -> double

In the above statement, left to right can be assigned implicitly and right to left requires explicit casting. That is, **byte** can be assigned to **short** implicitly but **short** to **byte** requires explicit casting.

Declaration of variables:

In java, the variables are the names of the storage locations. After designing the variable name, we must declare them to the compiler. It tells three things.

1. It tells the compiler what the variable name is
2. It specifies what type of data the variable will hold.
3. Place of declaration statement decides the scope of the variable.

A variable must be declared before it is used in the program variable can be used to store a value of any data type. That is, name has nothing to do with the type. Java allows any properly formed variable to be declared as a type.

The declaration statement defines the type of the variable. The general form of the declaration is

<data type name> <variable-name>

Ex: int count;

Giving values to variables:

A variable must be given a value after it has been declared. This can be achieved in two ways.

1. Assignment statement
2. Read statement

Assignment statement

The syntax for giving a value to the variable is as follows.

<variable-name>=<value>

Ex: int x;

x=10; or int x=10;

Read statement:

We may also give values to variables interactively through the keyboard using the readLine () method as illustrated in program

Example1: Reading data from keyboard

The following java program illustrates the concept of reading the data from the keyboard using the readLine() method.

Aim: To read an integer value from the keyboard.

//Reading.java

```
import java.io.DataInputStream
class Reading
{
    public static void main(String args[]) throws IOException
    {
        DataInputStream in;
        in=new DataInputStream(System.in);
        int i;
        System.out.println("Enter an integer")
        i=Integer.parseInt (in.readLine ());
        System.out.println ("entered integer value is:"+i);

    }
}
```

Example2:

Aim: To read two integer values from the keyboard using readLine() method and to find out the sum of those two integer values.

//IntegerSum.java

```
import java.lang.*;
import java.io.*;

class IntegerSum
{
    public static void main(String args[])throws IOException
    {
        int i1,i2,i3;
        String s1,s2;
        DataInputStream in=new DataInputStream(System.in);
        System.out.println("Enter the first integer");
        s1=in.readLine();
        i1=Integer.parseInt(s1);
```

```
        System.out.println("Enter the second integer");
        s2=in.readLine();
        i2=Integer.parseInt(s2);
        i3=i1+i2;
        System.out.println("The intger sum=" + i3);
    }
}
```

Example3:

Aim: To find the sum of two float values by reading them from the keyboard using readLine() method.

//FloatSum.java

```
import java.lang.*;
```

```
import java.io.*;
```

```
class FloatSum
```

```
{
    public static void main(String args[])throws IOException
    {
        float f1,f2,f3;
        DataInputStream in=new DataInputStream(System.in);
        System.out.println("Enter the first integer");
        f1=Float.parseFloat(in.readLine());
        System.out.println("Enter the second integer");
        f2=Float.parseFloat(in.readLine());
        f3=f1+f2;
        System.out.println("The intger sum=" + f3);
    }
}
```

Scope of variables

These are of three types

- **Instance variables:** These are declared and created when the objects are instantiated. They take different values for each object.
- **Class variables:** These are present within the class.
- **Local variables:** are the variables that are present within the method.

Symbolic constants:

A Numeric value is not always clear, to the sense that it means different things at different places. For example 50 may mean no of students at one place, and the pass marks at the other place.

Assignment of a symbolic name to such constants frees us from these problems. For example we may use the name strength to denote the no of students. “pass-mark “to denote the pass marks required in a subject.

A constant is declared as follows.

Syntax: final <data types> <symbolic-name>=value

Ex: final int strength=50;

1. Symbolic names take the same form as variable names. But they are generally written in capitals to distinguish from the normal variable names.
2. After declaration they should not be assigned any value.

Ex: STRENGTH=200; is illegal

3. They should not be declared inside a method. They should be used only as class data members in the beginning of the class.

Chapter-5

Operators and Expressions

Q. Explain about the Operators in Java

Java supports a rich set of operators. An Operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables.

Java Operators are classified into a number of related categories as shown in below.

- Arithmetic Operators
- Relational Operators
- Logical Operators.
- Assignment Operators
- Increment and Decrement Operators.
- Conditional Operators.
- Bitwise Operators
- Special Operators

Arithmetic Operators

Java provides all the basic arithmetic operators. They are listed below. These can be used to construct the mathematical expressions. The operators work the same way as they do in any other languages. These can operate on any built-in numeric data type of java.

Operator	Meaning
+	Addition or unary plus
-	Subtraction or Unary minus
*	Multiplication
/	Division
%	Modulo Division

Arithmetic Operators are used as follows

$$a+b, a-b, a*b, a/b, a\%b$$

where a and b are may be variables or constants and are known as operands.

We have three types of arithmetic possible names.

- Integer Arithmetic.
- Real Arithmetic
- Mixed-Mode Arithmetic.

Integer Arithmetic

When both the operands in a single arithmetic expression such as $a+b$ are integers, the expression is called an integer expression, and the operation is called integer arithmetic. Integer arithmetic always yields an integer value.

Real Arithmetic

An arithmetic operation involving only real operands is called real arithmetic. A real operand may assume values either in decimal or exponential notation. Since floating point values are rounded to the number of significant digits permissible, the final value is an approximation of the correct result.

Mixed-Mode Arithmetic

When one of the operands is real and the other is an integer, the expression is called a mixed mode arithmetic expression. If either operand is of the real type, then the other operand is converted to real and the real arithmetic is performed. The result will be a real.

Relational Operators

We often compare two quantities, and depending on their relation, take certain decisions. These comparisons can be done with the help of relational operators. The list of operators as shown in below

Operator	Meaning
<	Is less than
<=	Is less than or equal to
>	Is greater than
>=	Is greater than or equal to
==	Is equal to
!=	Is not equal to

A simple relational expression contains only one relational operator and is of the following form.

exp1 reoperator exp2

here exp1 and exp2 are any two arithmetic expressions, which may be simple constants, variables or combination of them. When arithmetic expressions are used on either side of a relational operator, the arithmetic expressions will be evaluated first and then the results compared. That is, arithmetic operators have a higher priority over relational operators.

Logical Operators

In addition to relational operators, java has three logical operators listed below in the table.

Operator	Meaning
&&	Logical AND
	Logical OR
!	Logical NOT

The logical operators && and || are used when we want form compound conditions by combining two or more relations.

Example: (a>b && b<c)

An expression of this kind which combines two or more relational expressions is termed as a logical expression or a compound relational expression. Like the simple relational expressions, a logical expression also yields a value of true or false, according to the truth table of AND and OR.

exp1	exp2	exp1 && exp2	exp1 exp2
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Assignment Operators

Assignment Operators are used to assign the value of an expression to a variable. We have seen the usual assignment operator, '='. In addition to that java has a set of short hand assignment operators which are used in the form

var op=exp

Where var is a variable, exp is an expression and op is a java binary operator. The operator 'op=' is known as shorthand assignment operator.

Example:

x+=y+4

This actually means,

x=x+(y+4).

There are three advantages using the shorthand operators, they are

- What appears on the left hand side need not be repeated and therefore it becomes easier to write.
- The statement is more concise and easier to read.
- Use of shorthand operator's result in a more efficient code.

Increment and Decrement Operators

Java has two very useful operators called the increment (++) and decrement (--) operators. The operator ++ adds one to the operand while the operator - subtracts one. Both are unary operators.

Example: ++a, a++, --a, a--.

++a is called as pre increment and a++ is called post increment.

Though ++a and a++ mean the same thing when they form statements independently, they behave differently when they are used in expressions on the right hand side of an assignment statement.

Example:

a=5;

b=++a;

the result is a=6, b=6;

```
a=5;
```

```
b=a++;
```

the result is a=6, b=5;

A prefix operator first adds one to the operand and then the result is assigned to the variable on left. On the other hand, a post fix operator first assigns the value to the variable on left and then increments the operand.

Conditional Operator

The character pair **?:** is a ternary operator available in java. This operator is used to construct conditional expressions of the form

```
exp1 ? exp2 : exp3
```

where exp1, exp2 and exp3 are all expressions.

Here first the expression exp1 was evaluated, if it is true, exp2 is evaluated. If it is false, exp3 is evaluated. However, it has to be clearly understood that among both exp2 and exp3, only one of it is evaluated at any time.

Example:

```
a=10;
```

```
b=15;
```

```
x=(a>b) ? a : b;
```

In this example, x will be assigned the value b.

Bitwise operators:

These operators act on individual bits(0 and 1) of the operands. They act on only integer datatypes i.e. byte, short, int, long. There are different types of operators, they are:

1. Bitwise Complement operator(~)
2. Bitwise and operator (&)
3. Bitwise or operator (|)
4. Bitwise xor operator (^)
5. Bitwise left shift operator (<<)
6. Bitwise right shift operator (>>)

Bitwise Complement operator (~)

This operator gives the complement form of a given number. This operator symbol is ~ which is pronounced as tilde.

Complement form of a positive number can be obtained by changing 0's as 1's and visa versa.

Bitwise and operator (&):

This operator performs *and* operations on the individual bits of the number. The symbol for the operator is (&) which is called ampersand.

If int X=10, Y=11 then find the value of X&Y.

X=10=0000 1010	X	Y	X&Y
Y=11=0000 1011	1	1	1
-----	1	0	0
X&Y=00001010	0	1	0
	0	0	0

From the above table, by multiply the bits, we get X&Y=0000 1010.

Bitwise or operator (|)

This operator performs *or* operations on the individual bits of the number. The symbol for the operator is (|) which is called pipe.

If int X=10, Y=11 then find the value of X|Y.

X=10=0000 1010	X	Y	X Y
Y=11=0000 1011	1	1	1
-----	1	0	1
X Y=00001011	0	1	1
	0	0	0

From the above table, by adding the bits, we get X|Y=0000 1011.

Bitwise xor operator (^)

This operator performs (exclusive or) *xor* operations on the individual bits of the number. The symbol for the operator is (^) which is called cap,carat.

If int X=10, Y=11 then find the value of X^Y.

X=10=0000 1010

Y=11=0000 1011

X^Y=00000001

X	Y	X^Y
1	1	0
1	0	1
0	1	1
0	0	0

From the above table, when odd numbers of 1's are there, we can get a 1 in the output.

Bitwise left shift operator (<<)

The left shift operator will shift the bits towards left for the given number of times.

int a=2<<1;

Let's take the binary representation of 2 assuming int is 1 byte for simplicity.

Position 7 6 5 4 3 2 1 0

Bits 0 0 0 0 0 0 1 0

Now shifting the bits towards left for 1 time, will give the following result

Position 7 6 5 4 3 2 1 0

Bits 0 0 0 0 0 1 0 0

Now the result in decimal is 4. You can also note that, 0 is added as padding the in the position 0.

Bitwise right shift operator (>>)

The right shift operator will shift the bits towards right for the given number of times.

int a=8>>1;

Let's take the binary representation of 8 assuming int is 1 byte for simplicity.

Position 7 6 5 4 3 2 1 0

Bits 0 0 0 0 1 0 0 0

Now shifting the bits towards right for 1 time, will give the following result

Position 7 6 5 4 3 2 1 0

Bits 0 0 0 0 0 1 0 0

Now the result in decimal is 4. Right shifting 1 time, is equivalent to dividing the value by 2.

Write a program using bitwise operators:

```
class bits
{
    public static void main(String[] args)
    {
        byte x,y;
        x=10;
        y=11;
        System.out.println("~x="     +(~x));
        System.out.println("x&y="    +(x&y));
        System.out.println("x|y="    +(x|y));
        System.out.println("x^y="    +(x^y));
        System.out.println("x<<2="   +(x<<2));
        System.out.println("~x>>2"   +(x>>2));
    }
}
```

OUTPUT:

```
D:\ravindra>javac bits.java
D:\ravindra>java bits
^x=-11
x&y=10
x!y=11
x^y=1
x<<2=40
^x>>22
```

Special Operators:

- a. Instanceof Operator
- b. Dot(.) Operator

Instanceof Operator

In **java instanceof operator** is used to test whether the object is an instance of the specified type (class or subclass or interface).

This operator returns “true” if an object given at the left hand side is an instance of the class given at right hand side. Otherwise, it returns “false”.

```
class person
{
    String name;
    int age;
    void talk()
    {
        System.out.println("my name is"+ name);
        System.out.println("my age is"+ age);
    }
}

class Demo1
{
    public static void main(String[] args)
    {
        person raju=new person();
        raju.name="ravindra";
        raju.age=29;
        raju.talk();
    }
}
```

```
boolean k=raju instanceof person;  
System.out.println(k);  
    }  
}
```

Dot operator

This operator is used to access the variables and methods of a class.

Example 1

Syntax:

Object name. variable name

raju.name

Here we are accessing the variable “name” of the “raju” object

Example 2

Syntax:

Object name. method name

raju.talk()

Here we are accessing the method “talk()” of the “raju” object.

This operator also can be used to access classes, packages etc.

Arithmetic Expression

An Arithmetic expression is a combination of variables and operators arranged as per the syntax of the language. Java can handle any complex mathematical expression.

Expressions are evaluated using an assignment statement of the form

variable=expression;

variable is any valid java variable name. when the statement is encountered, the expression is evaluated first and the result then replaces the previous value of the variable on the left hand side.

Example:

$x=a*b-c;$

Precedence of Arithmetic Operators

An arithmetic expression without any parentheses will be evaluated from left to right using the rules of precedence of operators. There are two distinct priority levels of arithmetic operators in java.

High Priority * / %

Low Priority + -

The basic evaluation procedure includes two left to right passes through the expression. During the first pass, the high priority operators are applied and during the second pass, the low priority operators are applied. Consider the following evaluation statement

$x= a - b / 3 + c * 2 -1$

when $a=9$, $b=12$ and $c=3$, the statement becomes

$x= 9 - 12 / 3 + 3 * 2 -1$

and is evaluated as follows.

First pass

Step1: $x=9-4+3*2-1$ (12/3 evaluated)

Step2: $x=9-4+6-1$ (3*2 evaluated)

Second pass:

Step3: $x=5+6-1$ (9-4 evaluated)

Step4: $x=11-1$ (5+6 evaluated)

Step5: $x=10$ (11-1 evaluated)

Q. Explain about the type conversions in Expressions

Automatic Type Conversion

Java permits mixing of constants and variables of different types in an expression, but during evaluation, it adheres to very strict rules of type conversion. If the two operands of an expression are of different type, the lower type is automatically converted into the higher type before the operation proceeds. The result is of the higher type.

The final result of an expression is converted to the type of variable on the left of the assignment sign before assigning the value to it. However, the following changes are introduced during the final assignment.

- Float to int causes truncation of fractional part.
- Double to float causes rounding of digits.
- Long to int causes dropping of the excess higher order bits.

The following table provides a reference for type conversion

	char	byte	short	int	long	float	Double
char	int	int	int	int	long	float	double
byte	int	int	int	int	long	float	double
short	int	int	int	int	long	float	double
int	int	int	int	int	long	float	double
long	long	long	long	long	long	float	double
float	float	float	float	float	float	float	double
double	double	double	double	double	double	double	double

Casting a Value

There are instance which need type casting than type conversion, i.e., there are instance which are different from type conversion. The process of this conversion is known as casting a value. The general form a cast is:

(type-name) expression.

Example:

Consider the ratio of females to males in a town

```
ratio= female_number/ male_number;
```

Since **female_number** and **male_number** are declared as integers in the program, the decimal part of the result of the division would be lost and ratio would not represent a correct result. This problem can be solved by converting locally one of the variables to the floating point as shown in below.

```
ratio= (float)female_number/ male_number;
```

The operator **(float)** converts the female_number to floating point for the purpose of evaluation of the expression.

Q. Explain the operator precedence and Associativity in java

Each operator in java has a precedence associated with it. The precedence is used to determine how an expression involving more than one operator is evaluated. The operators at the higher level of precedence are evaluated first.

The operators of the same precedence are evaluated either from left to right or from right to left, depending on the level. This is known as the associativity property of an operator.

The following table provides complex lists of operators, their precedence levels, and their rules of association.

Operator	Description	Associativity	Rank
· () []	Member selection Function call Array element reference	Left to Right	1
- ++ -- ! ~ (type)	Unary minus Increment Decrement Logical negation Ones complement Casting	Right to left	2
* / %	Multiplication Division Modulus	Left to right	3
+ -	Addition Subtraction	Left to right	4
<< >> >>>	Left shift Right shift Right shift with zero fill	Left to right	5
< <= > >= instanceof	Less than Less than or equal to Greater than Greater than or equal to Type comparison	Left to right	6
== !=	Equality In equality	Left to right	7
& ^ && ?: = op=	Bitwise AND Bitwise XOR Bitwise OR Logical AND Logical OR Conditional Operator Assignment operator Shorthand assignment	Left to right Left to right Left to right Left to right Left to right Right to left Right to left	8 9 10 11 12 13 14

Consider the following conditional statement

```
if(x==10 + 15 && y<10)
```

the precedence rules says that the addition operator has high priority than the logical operator (&&) and the relational operators (== and <). Therefore, the addition of 10 and 15 is executed first. This is equivalent to

```
if (x==25 && y<10)
```

Assume that x=20 and y=5, then

X==25 is false and y< 10 is true

Finally we get,

```
If(False && True)
```

Because one of the conditions is false, the compound condition is false.

Q. Explain the Mathematical Functions in Java

Java supports the basic mathematical functions through Math class defined in java.lang package. The following table lists the mathematical functions defined in the Math class. These functions are used as follows

```
Math.function_name();
```

Ex:

```
Double y=Math.sqrt(x);
```

Function	Action
sin(x)	Returns the sine of the angle x in radians
cos(x)	Returns the cosine of the angle x in radians
tan(x)	Returns the tangent of the angle x in radians
asin(y)	Returns the angle whose sine is y
acos(y)	Returns the angle whose cosine is y
atan(y)	Returns the angle whose tangent is y
atan2(x,y)	Returns the angle whose tangent is x/y
pow(x,y)	Returns x raised to y (x^y)
exp(x)	Returns e raised to x (e^x)
log(x)	Returns the natural logarithm of x
sqrt(x)	Returns the square root of x
round(x)	Returns the integer closest to the argument

abs(a)	Returns the absolute value of a
max(a,b)	Returns the maximum of a and b
min(a,b)	Returns the minimum of a and b

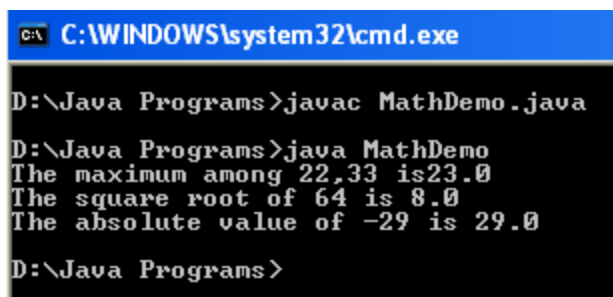
The following program illustrates the math functions supported by the java language.

Aim: To illustrates the use of mathematical functions supported by the java language.

Program:

```
class MathDemo
{
    public static void main(String args[])
    {
        double x;
        x=Math.max(22,23);
        System.out.println("The maximum among 22,33 is" + x);
        x=Math.sqrt(64);
        System.out.println("The square root of 64 is " + x);
        x=Math.abs(-29);
        System.out.println("The absolute value of -29 is " + x);
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\Java Programs>javac MathDemo.java
D:\Java Programs>java MathDemo
The maximum among 22,33 is23.0
The square root of 64 is 8.0
The absolute value of -29 is 29.0
D:\Java Programs>
```

Chapter-6

Decision Making and Branching

Introduction:

Generally a program executes its statements from beginning to end. But not many programs execute all their statements in strict order from beginning to end. Most programs decide what to do in response to changing circumstances. In a program, Statements may be executed sequentially, selectively or iteratively. Every programming language provides constructs to support sequence, selection or iteration.

When a program breaks the sequential flow and jumps to another part of the code, it is called selection or branching.

When the branching is based on a particular condition, it is known as conditional branching.

If branching takes place without any condition, it is known as unconditional branching.

Decision Making with If Statement:

An if statement tests a particular condition. If the condition is true then statement or set of statements is executed. Otherwise (if the condition evaluates to false) then it will come out of loop.

The if statement may be implemented in different forms depending on the complexity of conditions to be tested.

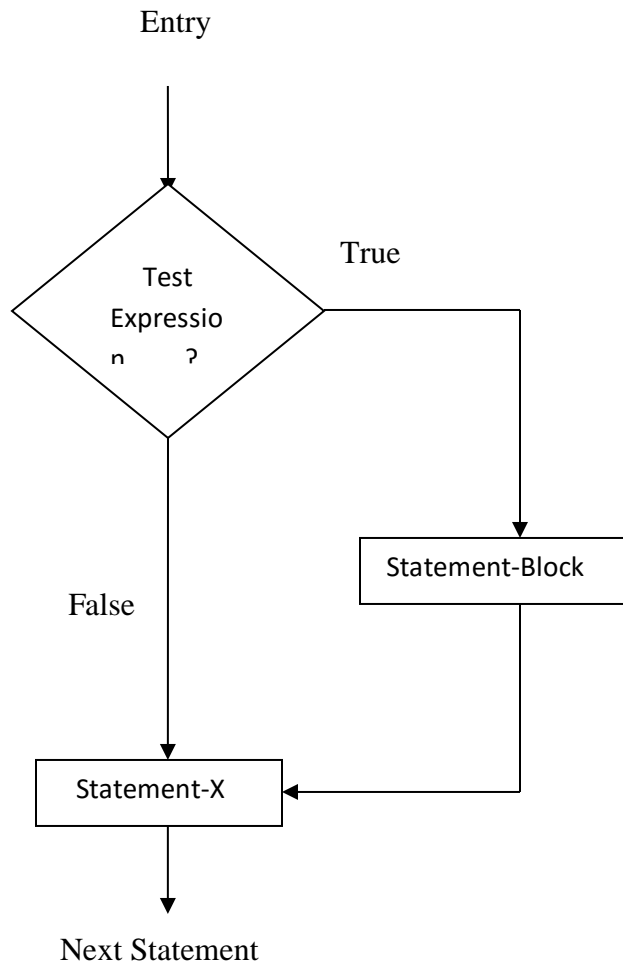
- Simple if statement
- If-else statement
- Nested if-else statement
- else if ladder

Simple If Statement: The syntax of the if statement is shown below

Syntax: if(expression)

```
{  
    Statements;  
}  
  
Statement-x;
```

Where a statement may consist of a single statement, a compound statement, or nothing. The expression must be enclosed in parenthesis. If the expression evaluates to true, the statement is executed, otherwise ignored.



Example

We can take a look on a program which will display a message “Success” if a particular value is greater than 5. It then displays a message “Executed successfully” and complete its execution.


Program

```
/*
 * FileName : SimpleIfStatementDemo1.java
 */
public class SimpleIfStatementDemo1
{
    public static void main(String args[])
    {
        //Declaring a variable "test" and initializing it with a value 10
        int test=10;

        //Checking if "test" is greater than 5
        if(test>5)
        {
            //This block will be executed only if "test" is greater than 5
            System.out.println("Success");
        }

        //The if block ends.
        System.out.println("Executed successfully");
    }
}
```

Output



```
C:\WINDOWS\system32\cmd.exe
D:\>java SimpleIfStatementDemo1
Success
Executed successfully
D:\>
```

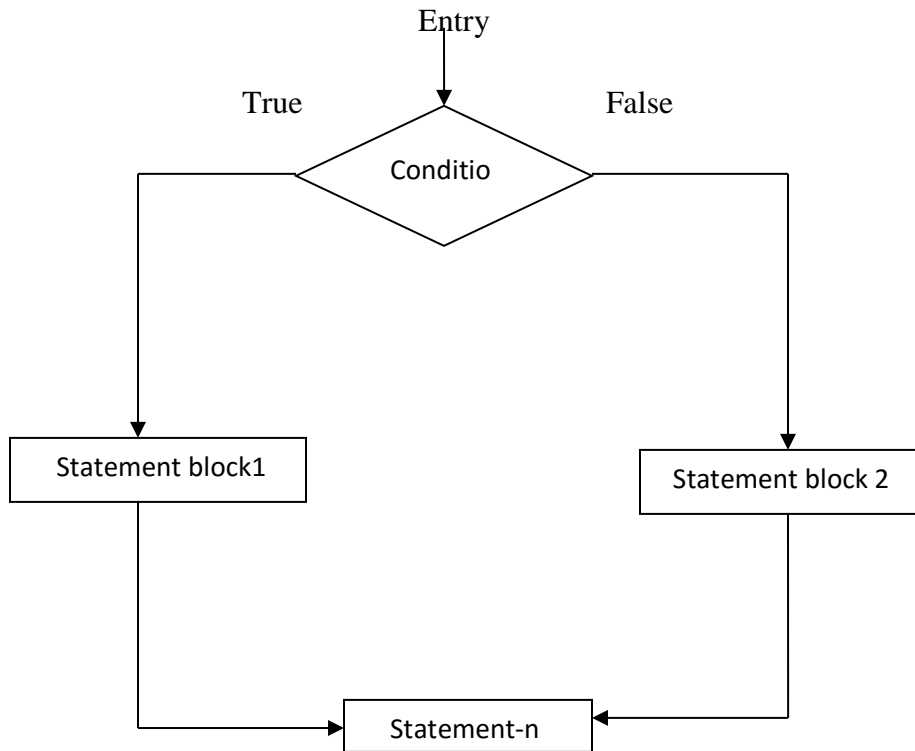
This program outputs the message “Success” because the declared variable “test” has a value 10 which is greater than 5.

The if-else statement: The syntax of the if-else statement is the following:

Syntax:

```
if(condition)
{
    Statement block 1;
}
else
{
    Statement block2;
}
Statement-n
```

If the condition is true ,then the statement block1 and then statement-n and so on executed in order. If the condition is false, statement block2 and then statement-n and so no are executed in order. In either case, only one of the statement statement block1 or statement block2 is executed but not both.



Write a program to illustrate if else statement:

```
public class si
{
    public static void main(String args[]){
        int a = 10;
        int b = 20;
        if(a>b)
        {
            System.out.println("a is greater than b");
        }
        else
        {
            System.out.println("b is less than a");
        }
    }
}
```

Output:

```
D:\ravindra>javac si.java
D:\ravindra>java si
b is less than a
```

Nesting if-else statements:

When a series of decisions are involved, we may have to use more than one if statements in nested as follows.

Syntax:

```
if(condition)
{
    if(condition)
    {
        Statement block1;
    }
else
    {
        Statement block 2;
    }
}
else
{
    if(condition)
    {
        Statement block 3;
    }
else
    {
        Statement block 4;
```



```
}
```

Statement-n;

Flow chart:

Eg:

Class ifelsenesting

```
{
```

```
    Public static void main(String args[])
```

```
    {
```

```
        int a=325,b=712,c=478;
```

```
        System.out.println("largest value is:");
```

```
        if(a>b)
```

```
        {
```

```
            if(a>c)
```

```
            {
```

```
                System.out.println(a);
```

```
            }
```

```
        else
```

```
        {
```

```
            System.out.println(a);
```

```
        }
```

```
    }
```

```
    else
```

```
    {
```

```
        if(b>c)
```

```
        {
```

```
            System.out.println(b);
```

```
    }
    else
    {
        System.out.println(c);
    }
}
```

Output: Largest value is :712

else if ladder:

A common programming construct in the java is the if-else-if ladder , which is often also called the if-else-if staircase because of it's appearance.

Syntax:

```
if (condition1)
    statement1;
else if (condition2)
    statement2;
else if(condition3)
    statement3;
.....
.....
else if(condition n)
    statement n;
else
    default-statement;
statement-x;
```

This construct is known as the else if ladder. The conditions are evaluated from the top of the ladder. As soon as the true condition is found, the statement associated with it is executed

and the control is transferred to the statement-x. When all the n conditions becomes false, then the final else containing the default-statement will be executed.

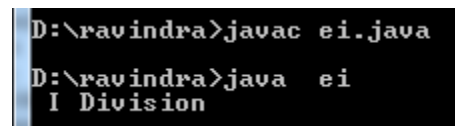
Flow chart:

Eg:

```
class ei
{
    public static void main(String args[])
    {
        int marks=70;

        if(marks>79)
        {
            System.out.println("honours");
        }
        else if(marks>59)
        {
            System.out.println(" I Division");
        }
        else if(marks>79)
        {
            System.out.println("II Division");
        }
        else
        {
            System.out.println(" Fail");
        }
    }
}
```

Output:



```
D:\ravindra>javac ei.java
D:\ravindra>java ei
I Division
```

The Switch Statement:

Java provides a multiple branch selection statement known as switch. This selection statement successively tests the value of an expression against a list of integer or character constants. When a match is found, the statements associated with that constant are executed.

Syntax:

```
switch(expression)
```

```
{
```

```
case value1:
```

```
    block-1
```

```
    break;
```

```
case value2:
```

```
    block-2
```

```
    break;
```

```
    .....
```

```
    .....
```

```
default:
```

```
    default-block
```

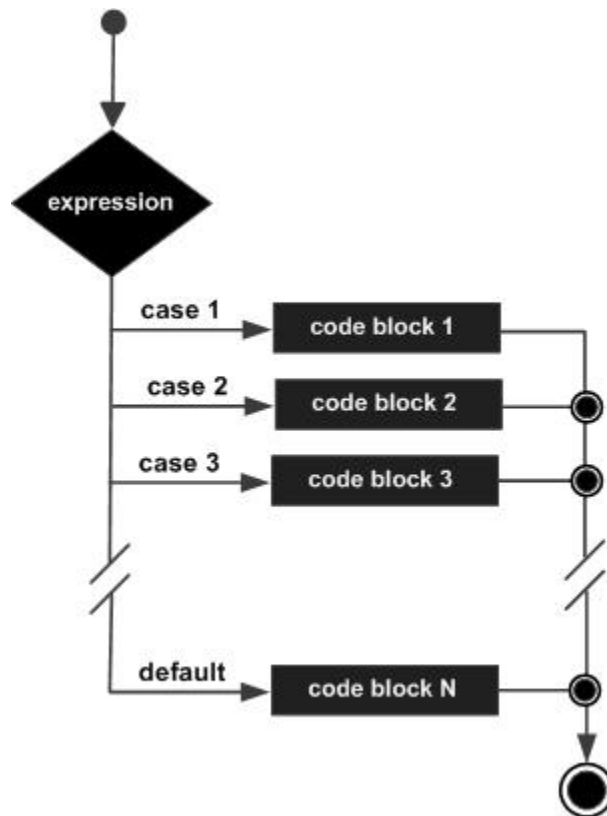
```
    break;
```

```
}
```

```
Statement-x;
```

A switch statement is used for multiple ways selection that will branch to different code segments based on the value of a variable or an expression. The optional default label is used to specify the code segment to be executed. when the value of the variable or expression does not match with any of the case values. if there is no break statement as the last statement in the code segment for a certain case, the execution will continue on into the code segment for the next case clause without checking the case value.

Flowchart:



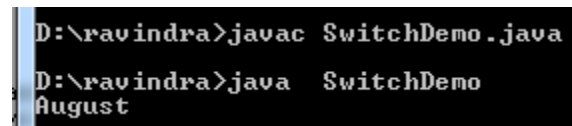
Example:

```
class SwitchDemo
{
    public static void main(String[] args)
    {

        int month = 8;
        switch (month)
        {
```

```
case 1: System.out.println("January"); break;
case 2: System.out.println("February"); break;
case 3: System.out.println("March"); break;
case 4: System.out.println("April"); break;
case 5: System.out.println("May"); break;
case 6: System.out.println("June"); break;
case 7: System.out.println("July"); break;
case 8: System.out.println("August"); break;
case 9: System.out.println("September"); break;
case 10: System.out.println("October"); break;
case 11: System.out.println("November"); break;
case 12: System.out.println("December"); break;
default: System.out.println("Invalid month.");break;
    }
}
}
```

Output:



```
D:\ravindra>javac SwitchDemo.java
D:\ravindra>java SwitchDemo
August
```

The ?:Operator (or)conditional operator (or)ternary operator:

This operator is called ternary operator because it acts on 3 variables. The other name for this operator is conditional operator, since it represents a conditional statement. Two symbols are used for this operator? and :

Syntax: Variable=expression1? expression2: expression 3

First the compiler evaluates the expression1 if it is true expression 2 will be executed by ignoring the expression 3 i.e, expression 2 will become the resultant value for the entire expression. If the expression 1 is false expression 3 will be executed by ignoring expression 2 i.e, expression 3 will become the resultant value of the entire expression.

Chapter-7

Decision Making and Looping

The process of repeatedly executing a block of statements is known as looping. The statements in the block may be executed any number of times, from zero to infinite number of times. If a loop continues forever, it is called an infinite loop.

The Java Language provides three constructs for performing loop operations. They are

- **while construct**
- **do construct**
- **for construct**

While statement

The basic format of the while statement is as follows

```
Initialization:
while (test condition)
{
    Body of the loop;
}
```

The while is an entry controlled loop statement. The test condition is evaluated and if the condition is true, then the body of the loop is executed. After execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again. This process is repeated until the test condition returns a false value.

Consider the following code segment

```
.....
.....
sum=0;
n=1;
while(n<=10)
{
    sum=sum+n*n;
    n=n+1;
}
System.out.println("Sum=" + sum);
.....
```

.....

The body of the loop executed 10 times from n=1,2,...10 each time adding the square of the value of n, which is incremented inside the loop.

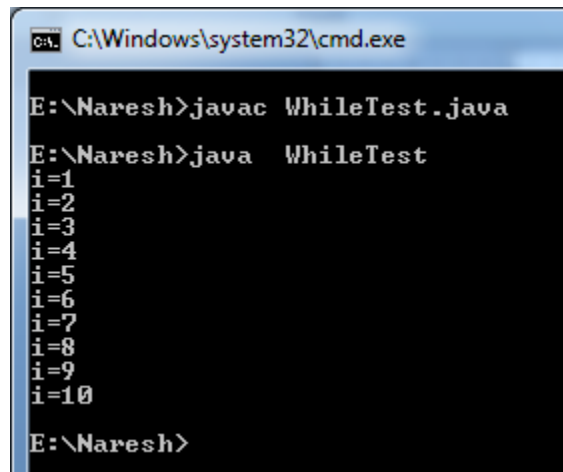
The following program illustrates the use of while loop in java.

Aim: To read a string from the keyboard character by character and to display it on the screen.

Program:

```
import java.io.*;
class WhileTest
{
    public static void main(String[] args) throws IOException
    {
        int i=1;
        while(i<=10)
        {
            System.out.println("i=" + i);
            i++;
        }
    }
}
```

Output:



The do..while Statement

The following is the syntax for the do statement

```
Initialization;  
do  
{  
    Body of the loop;  
} while (test condition);
```

On reaching the do statement, the program proceeds to execute the body of the loop first. At the end of the loop, the test condition in the while statement is evaluated. If the condition is true, the program continues to evaluate the body of the loop once again. This process continues as long as the condition is true. When the condition is false, the loop will be terminated.

Since the test condition is evaluated at the bottom of the loop, the do...while constructs provides an exit-controlled loop and therefore the body of the loop is always executed at least once.

Consider an example

```
.....  
.....  
i=1;  
sum=0;  
do  
{  
    sum=sum+I;  
    i=i+2;  
}while (sum<40 || i<10);
```

The loop will be executed as long as one of the two relations is true.

The following program illustrates the do...while construct

Aim: To display the multiplication table.

Program:

```
class dow {  
  
    public static void main(String args[]){  
  
        int i=0;
```

```

do{
    System.out.println(i);
    i++;
}while(i<10);
}
}

```

Output:

```

D:\ravindra>javac dow.java
D:\ravindra>java dow
0
1
2
3
4
5
6
7
8
9

```

Difference between while and do...while constructs

while	Do...while
It is a looping construct that will execute only if the test condition is true.	It is a looping construct that will execute at least once even if the test condition is false.
It is entry controlled loop.	It is an exit controlled loop.
It is generally used for implementing common looping situations.	It is typically used for implementing menu based programs where the menu is required to be printed at least once.

The for statement

The for loop is another entry controlled loop that provides a more concise loop control structure. The general form of the loop is as follows.

```

for(initialization ; test condition ; increment/decrement)
{

```

```
        Body of the loop;  
    }
```

The execution of the for statement is as follows

1. initialization of the control variable is done first, using assignment statements such as `i=1` and `count=0`. The variables `i` and `count` are known as loop-control variables.
2. The value of the control variable is tested using the test condition. If the test condition is true, the body of the loop is executed; otherwise the loop is terminated.
3. When the body of the loop is executed, the control is transfer back to the for statement after evaluating the last statement in the loop. Now, the control variable is incremented or decrement.

Consider the following segment of a program

```
for(x=0; x<=9; x++)  
{  
    System.out.println(x);  
}
```

This for loop is executed 10 times and prints the digits 0 to 9.

Additional Features of a for Loop

The for loop has several capabilities that are not found in other loop constructs. They are

1. More than one variable can be initialized at a time in the for statement. The statements
`p=1;`
`for(n=0;n<10;n++)`
Can be written as
`for(p=1,n=0;n<10;n++)`
2. Like the initialization section, the increament section may also have more than one part. For example
`for(n=1,m=50;n<=m;n++,m--)`
3. The test condition may have any compound relation and the testing need not be limited only to the control variable. Consider the following

```
sum=0;
```

```

for(i=1;i<20&&sum<100;i++)
{
    .....
    .....
}

```

4. Another unique aspect of for loop is that one or more sections can be omitted, if necessary. Consider the following example

```

.....
.....
m=5;
for( ; m!=100 ; )
{
    System.out.println(m);
    m=m+5;
}
.....
.....

```

Nesting of for loops

Nesting of loops means one for statement within another for statement, is allowed in java. The for loops can be nested as follows

Aim: Program to display a right angled triangle of @

Program:

```

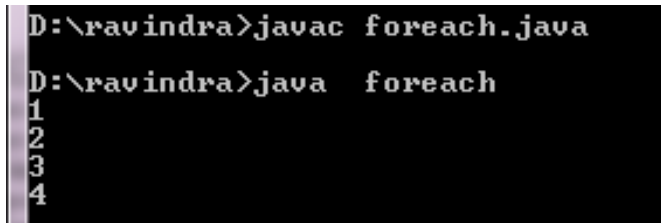
class NestedLoop
{
    public static void main(String[] args)
    {
        int i,j;
        System.out.println("The right angle triangle of @ is shown in below:\n");
        for(i=1;i<=9;i++)
        {
            for(j=1;j<=i;j++)
            {
                System.out.print("@");
            }
            System.out.println(" ");
        }
    }
}

```



```
        {  
            System.out.println(i);  
        }  
    }  
}
```

Output:



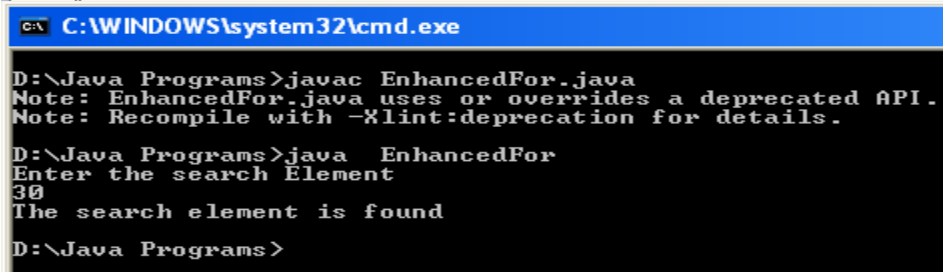
```
D:\ravindra>javac foreach.java  
D:\ravindra>java foreach  
1  
2  
3  
4
```

Aim: To find an element in the array of elements.

Program:

```
import java.io.*;  
class EnhancedFor  
{  
    int a[]={10,20,30,40,50};  
    int n;  
    boolean flag=false;  
  
    void getData(int x)  
    {  
        n=x;  
    }  
    void search()  
    {  
        for(int i:a)  
        {  
            if(i==n)  
            {  
                flag=true;  
                break;  
            }  
        }  
    }  
}
```

```
        else
        {
            flag=false;
        }
    }//for
    if(flag)
    {
        System.out.println("The search element is found");
    }
    else
    {
        System.out.println("The search element is not found");
    }
} //search()
public static void main(String[] args) throws IOException
{
    int n;
    DataInputStream in=new DataInputStream(System.in);
    System.out.println("Enter the search Element");
    n=Integer.parseInt(in.readLine());
    EnhancedFor o1=new EnhancedFor();
    o1.getData(n);
    o1.search();
} //main()
} //class
```

Output:

```
C:\WINDOWS\system32\cmd.exe
D:\Java Programs>javac EnhancedFor.java
Note: EnhancedFor.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
D:\Java Programs>java EnhancedFor
Enter the search Element
30
The search element is found
D:\Java Programs>
```

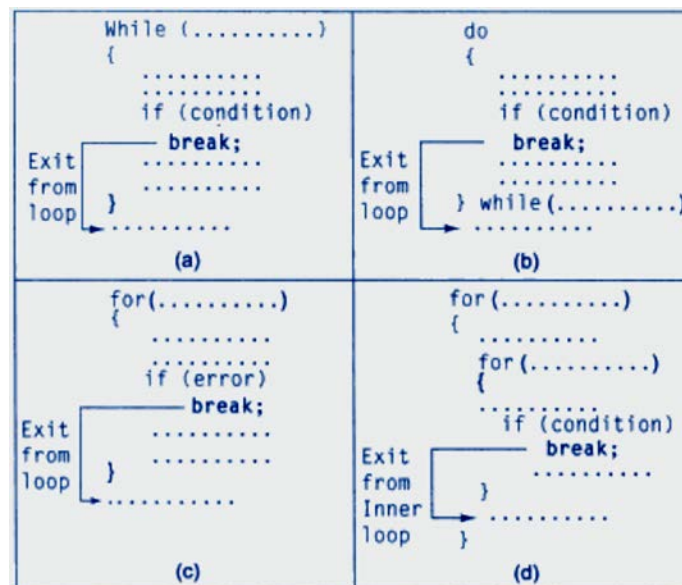
Jumps in Loops

Java permits a jump from one statement to the end or beginning of a loop as well as a jump out of a loop.

Jumping Out of loop (break statement)

When the break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop. When the loops are nested, the break would only exit from the loop containing it. The break will exit only a single loop.

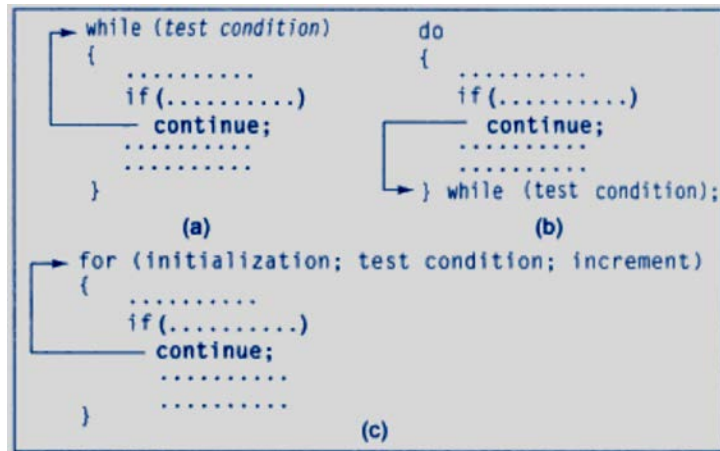
This statement can also be used within while, do or for loops as illustrated below.



Skipping a part of a Loop (continue statement)

Like the break statement java supports another similar statement called continue statement. Whenever the continue statement is encountered within the loop, that causes the loop to be continued with the next iteration after skipping any statements in between. The continue statement tells the compiler “ SKIP THE FOLLOWING STATEMENTS AND CONTINUE WITH THE NEXT ITERATION”.

The use of the continue statement in loops is illustrated below.



Labeled Loops

In java, we can give a label to a block of statements. A label is any valid java variable name. To give a label to a loop, place it before the loop with a colon at the end. Example

```
Loop1: for(.....)
{
    .....
    .....
}
.....
```

A simple break statement causes the control to jump outside the nearest loop and a simple continue statement restarts the current loop. If we want to jump outside a nested loops or to continue a loop that is outside the current one, then we may have to use the labeled break and labeled continue statements.

Example:

```
outer: for (int m = 1; m<11; m++)
{
    for (int n = 1; n<11; n++)
    {
        System.out.print (" " + m*n);
        if (n == m)
            continue outer;
    }
}
```

Here, the continue statement terminates the inner loop when n=m and continues with the next iteration of the outer loop.

Chapter-8

Classes, Objects and Methods

Defining a Class

Definition

The process of binding the data members and member functions into a single unit is called as a class. In other words a class is a user defined data type. Once the class type has been defined, we can create “variables” of that type. In java these variables are termed as instances of classes, which are the actual objects.

The basic form of a class definition is as follows

```
class clsname [ extends superclsname]
{
    [fields declaration];
    [methods declaration];
}
```

Here, clsname and superclsname are any valid java identifiers. The keyword extends indicates that the properties of the superclsname class are extended to the clsname class. This concept is called as inheritance.

Everything inside the square brackets is optional. This means that the following would be a valid class definition.

```
class clsname
{
}
```

Here, the body of the class is empty, this does not contain any properties and therefore cannot do anything.

Fields Declaration

Data is encapsulated in a class by placing data fields inside the body of the class definition. These variables are called instance variables because they are created whenever an

object of the class is instantiated. Instance variables are also known as member variables. We can declare the instance variables exactly the same way as we declare the local variables.

Example:

```
class Rectangle
{
    int length;
    int width;
}
```

The class Rectangle contains two integer type instance variables. Here, these variables are only declared and therefore no storage space has been created in the memory.

Methods Declaration

A class with only data fields has no life. The objects created by such a class cannot respond to any messages.

Methods are declared inside the body of the class but immediately after the declaration of instance variables. The general form of a method declaration is as follows.

```
type methodname ( parameter_list)
{
    Method-body;
}
```

Method declaration have four basic parts:

- The name of the method (methodname).
- The type of the value the method returns (type).
- A list of parameters (parameter_list)
- The body of the method.

Let us consider a Rectangle class example as follows

```
class Rectangle
{
    int length;
    int width;
    void getData( int x, int y)    // Method declaration
    {
        length=x;
        width=y;
    }
}
```

```
    }  
}
```

Creating Objects

An instance of a class is called as an object. An object in java is essentially a block of memory that contains space to store all the member variables. Crating an object is also referred to as instantiating an object.

Objects in java are created using the new operator. The new operator creates an object of the specified class and returns a reference to that object.

Here is an example of creating an object of type Rectangle.

```
Rectangle o1;           //declaring the object  
  
o1=new Rectangle();    // instantiate the object
```

The first statement declares a variable to hold the object reference and the second one actually assigns the object reference to the variable. The variable o1 is now an object of the Rectangle class.

Both statements can be combined into one as shown below.

```
Rectangle o1=new Rectangle();
```

The method Rectangle () is the default constructor of the class. We can create any number of objects of Rectangle. For example

```
Rectangle o1= new Rectangle();  
  
Rectangle o2= new Rectangle();
```

It is important to understand that each object has its own copy of the instance variable of its class. This means that any changes to the variables of one object can have no effect on the variables of another.

It is also possible to create two or more references to the same object as shown in below.

```
Rectangle R1=new Rectangle ();  
  
Rectangle R2=R1;
```

Accessing class members

To access the instance variables and the methods of a class we must use the concerned object and the dot (.) operator as shown in below.

```
objname.variablename=value;
```

```
objname.methodname(parameter_list);
```

Here, objname is the name of the object and variablename is the name of the instance variable inside the object that we wish to access.

The following is the program that illustrates the concepts discussed so far.

Aim: To find the area of a rectangle.

Program

//RectArea.java

```
class Rectangle
{
    int len,wid;        // Declaration of variables
    void getData(int x, int y)    //Declaration of method
    {
        len=x;
        wid=y;
    }
    int rectArea()        //Definition of another method
    {
        int area=len*wid;
        return (area);
    }
}
class RectArea        // class with main method
{
    public static void main(String args[])
    {
        int area1, area2;
        Rectangle r1=new Rectangle();    // Creating objects
        Rectangle r2=new Rectangle();

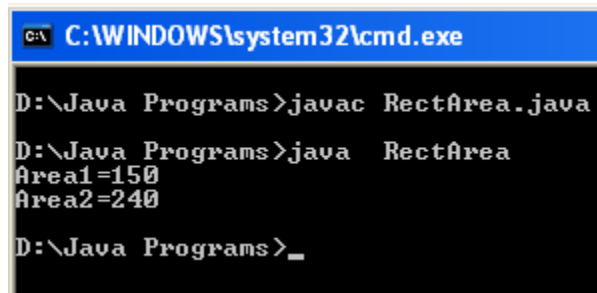
        r1.len=15;        //Accessing Variables
        r1.wid=10;
```

```
        area1=r1.len * r1.wid;

        r2.getData(20,12);    //Accessing methods
        area2=r2.rectArea();

        System.out.println("Area1=" + area1);
        System.out.println("Area2=" + area2);
    }
}
```

Output



```
C:\WINDOWS\system32\cmd.exe
D:\Java Programs>javac RectArea.java
D:\Java Programs>java RectArea
Area1=150
Area2=240
D:\Java Programs>_
```

Q. Explain about the Constructors in Java.

Constructor:

Constructors are special methods having the same name as the class itself. They do not have a return type, not even void.

A Constructor initializes an object automatically when the object is created. Constructors can be overloaded like methods. The constructor is automatically called when an object is created.

Java does three operations when new is used to create an instance of a class.

- Allocates the memory for the object.
- Initializes that objects instance variables, either to initial values or to a default.
- Calls the constructor of the class.

Constructors look a lot like regular methods with two basic differences.

- Constructors always have the same name as that of class.
- Constructor does not have a return type (even void)

The following program illustrates the concept of constructors in java.

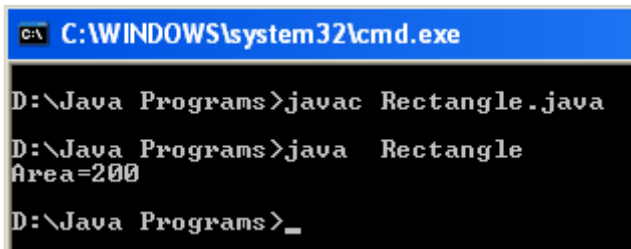
Aim: To find the area of a rectangle.

Program

//Rectangle.java

```
class Rectangle
{
    int len;
    int wid;
    Rectangle(int x, int y)
    {
        len=x;
        wid=y;
    }
    int area()
    {
        return (len*wid);
    }
    public static void main(String[] args)
    {
        //Constructor is implicitly called when we create an object
        Rectangle r1=new Rectangle(10,20); //calling constructor
        int x=r1.area(); //calling the method
        System.out.println("Area=" + x);
    }
}
```

Output



```
C:\WINDOWS\system32\cmd.exe
D:\Java Programs>javac Rectangle.java
D:\Java Programs>java Rectangle
Area=200
D:\Java Programs>_
```


Q. Explain about the method overloading in java.

In java, it is possible to create methods that have the same name, but different parameter list and different definitions. This is called method overloading.

Method overloading is used when objects are required to perform similar tasks but using different input parameters. When we call a method in an object, java matches up the method name first and then number and type of parameters to decide which one of the definitions to execute. The methods return type does not play any role in this.

The following program illustrates the concept of method overloading in java.

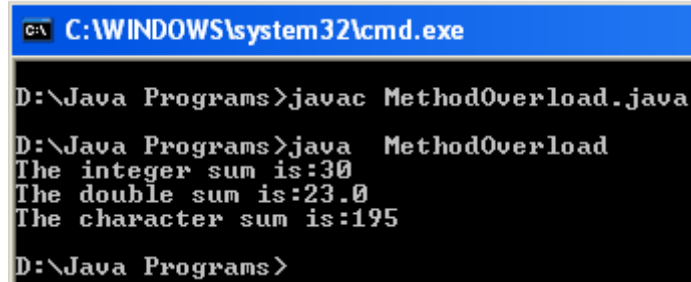
Aim: To find the sum of two integers, two float values, two double values and two characters

Program

//MethodOverload.java

```
class MethodOverload
{
    int sum(int x,int y)
    {
        int z;
        z=x+y;
        return z;
    }
    double sum(double x, double y)
    {
        double z;
        z=x+y;
        return z;
    }
    int sum(char x, char y)
    {
        int z;
        z=x+y;
        return z;
    }
    public static void main(String[] args)
    {
        MethodOverload o1=new MethodOverload();
        System.out.println("The integer sum is:" + o1.sum(10,20));
        System.out.println("The double sum is:" + o1.sum(10.5,12.5));
        System.out.println("The character sum is:" +o1.sum('a','b'));
```

```
    }  
}  
Output
```



```
C:\WINDOWS\system32\cmd.exe  
D:\Java Programs>javac MethodOverload.java  
D:\Java Programs>java MethodOverload  
The integer sum is:30  
The double sum is:23.0  
The character sum is:195  
D:\Java Programs>
```

Q. Explain about the Constructor overloading in java.

Like methods, constructors can also be overloaded. Appropriate constructor is called as per the parameters we pass at the time of object creation.

In the following program, the constructor Demo() is overloaded 3 times.

Aim: To illustrate the concept of constructor overloading.

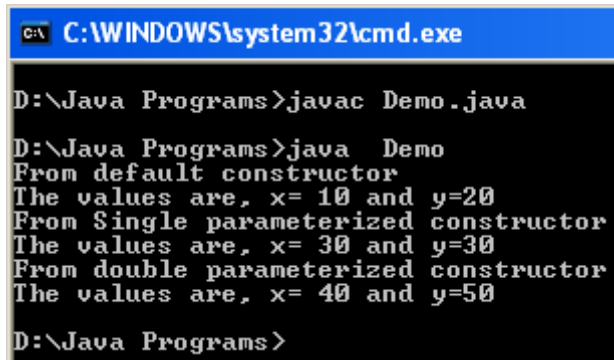
Program

//Demo.java

```
class Demo  
{  
    int x,y;  
    Demo()    // Default constructor  
    {  
        x=10;  
        y=20;  
        System.out.println("From default constructor");  
    }  
    Demo(int a) //Single parameterized constructor  
    {  
        x=a;  
        y=a;  
        System.out.println("From Single parameterized constructor");  
    }  
    Demo(int a, int b) //Double parameterized constructor  
    {
```

```
        x=a;
        y=b;
        System.out.println("From double parameterized constructor");
    }
    void display()
    {
        System.out.println("The values are, x= " + x + " and y=" + y);
    }
    public static void main(String[] args)
    {
        Demo d1=new Demo(); //calling default constructor
        d1.display();
        Demo d2=new Demo(30); // calling Single parameterized constructor
        d2.display();
        Demo d3=new Demo(40,50); //calling double parameterized constructor
        d3.display();
    }
}
```

Output



```
C:\WINDOWS\system32\cmd.exe

D:\Java Programs>javac Demo.java

D:\Java Programs>java Demo
From default constructor
The values are, x= 10 and y=20
From Single parameterized constructor
The values are, x= 30 and y=30
From double parameterized constructor
The values are, x= 40 and y=50

D:\Java Programs>
```

Q. Explain about the static members in java.

static variable

- It is a variable which **belongs to the class** and **not** to **object**(instance)
- Static variables are **initialized only once**, at the start of the execution. These variables will be initialized first, before the initialization of any instance variables
- A **single copy** to be shared by all instances of the class
- A static variable can be **accessed directly** by the **class name** and doesn't need any object

The following is the syntax for accessing a static variable

Syntax

<class-name>.<variable-name>

static method

- It is a method which **belongs to the class** and **not** to the **object**(instance)
- A static method **can access only static data**. It cannot access non-static data (instance variables)
- A static method **can call only** other **static methods** and cannot call a non-static method from it.
- A static method can be **accessed directly** by the **class name** and doesn't need any object

The following is the syntax for accessing a static variable

Syntax : **<class-name>.<method-name>**

Side Note

main method is static , since it must be be accessible for an application to run , before any instantiation takes place.

The following program illustrates the concept of static members in java.

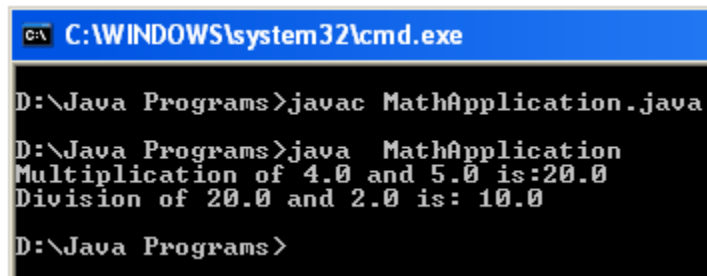
Aim: To perform the mathematical operations

Program:

```
//MathApplication.java  
  
class MathOperation  
{  
    static float mul(float x, float y)  
    {
```

```
        return x*y;
    }
    static float divide(float x, float y)
    {
        return x/y;
    }
}
class MathApplication
{
    public static void main(String[] args)
    {
        float a=MathOperation.mul(4.0f,5.0f);
        float b=MathOperation.divide(a,2.0f);
        System.out.println("Multiplication of 4.0 and 5.0 is:"+ a);
        System.out.println("Division of " + a + " and 2.0 is: " + b);
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\Java Programs>javac MathApplication.java
D:\Java Programs>java MathApplication
Multiplication of 4.0 and 5.0 is:20.0
Division of 20.0 and 2.0 is: 10.0
D:\Java Programs>
```

Q. Explain about the Nesting of methods in java.

A method can be called by using only its name by another method of the same class. This is known as nesting of methods.

The following program illustrates the concept of Nesting of methods in java.

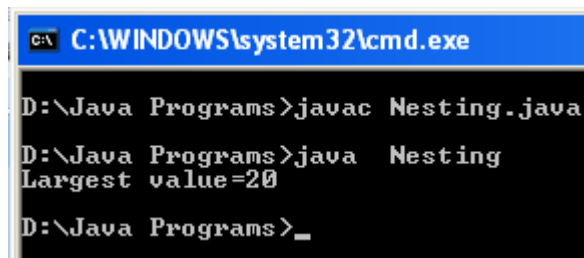
Aim: To find the largest of two numbers.

Program:

```
//Nesting.java
class Nesting
{
    int m,n;
    Nesting(int x, int y)
    {
        m=x;
        n=y;
    }
}
```

```
int largest()
{
    if (m>n)
        return m;
    else
        return n;
}
void display()
{
    int large=largest();
    System.out.println("Largest value=" + large);
}
public static void main(String[] args)
{
    Nesting o1=new Nesting(10,20);
    o1.display();
}
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\Java Programs>javac Nesting.java
D:\Java Programs>java Nesting
Largest value=20
D:\Java Programs>_
```

Q. Explain the Inheritance in java.

Definition:

The process of obtaining the data members and member functions from one class to another class is known as Inheritance.

The class which is giving data members and member functions to some other class is known as base/ super / parent class.

The class which is retrieving or obtaining the data members and member functions is known as derived/sub/child class.

A Derived class contains some of features of its own plus some of the data members from base class.

Advantages of Inheritance

- Application development time is less.
- Application amount of memory space is less.
- Redundancy (Repetition) of the code is reduced.

Syntax for INHERITING the features from base class to derived class:

```
class <clsname-2> extends <clsname-1>
{
    Variable declaration;
    Method definition;
}
```

Extends is a keyword which is used for inheriting the data members and methods from base class to the derived class and it also improves functionality of derived class.

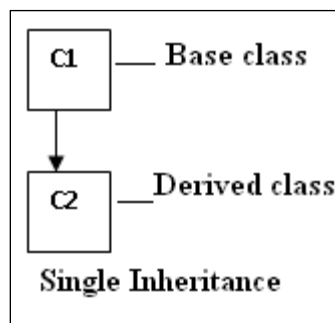
Types of Inheritances (Reusable Techniques)

Based on getting the features from one class to some other class, in java we have the following types of reusable techniques.

- Single Inheritance (only one super class)
- Multilevel Inheritance (several super classes)
- Multiple Inheritance (several super class, many sub classes)
- Hierarchical Inheritance (one super class, many sub classes)

Single Inheritance

It is one it contains a single base class and a single derived class. It is shown in below.



The following program illustrates the concept of Simple Inheritance

Aim: Program to find the area of the room and the volume of a bed room.

Program

//InheritTest.java

```
class Room
{
    float len, bre;
    Room(float x, float y)
    {
        len=x;
        bre=y;
    }
    float roomArea()
    {
        return (len*bre);
    }
}
class BedRoom extends Room
{
    float hei;
    BedRoom(float x, float y, float z)
    {
        super(x,y);
        hei=z;
    }
    float volume()
    {
        return (len*bre*hei);
    }
}
class InheritTest
{
    public static void main(String args[])
    {
        BedRoom r1=new BedRoom(12.2f,15.2f,10.3f);
        float area=r1.roomArea();
        float vol=r1.volume();
    }
}
```



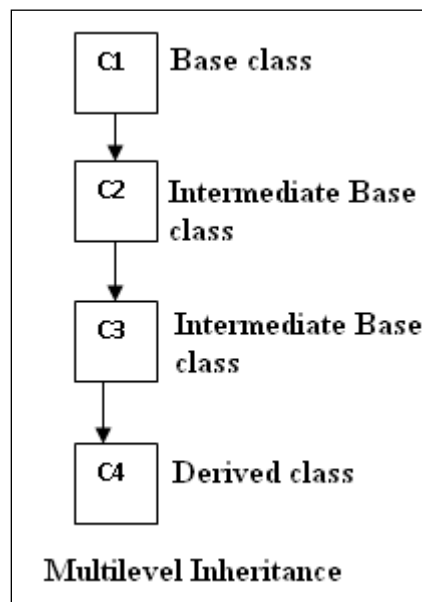
```
System.out.println("The area of the room is: " + area);  
System.out.println("The volume of the bed room is: " + vol);  
    }  
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe  
D:\Java Programs>javac InheritTest.java  
D:\Java Programs>java InheritTest  
The area of the room is: 185.43999  
The volume of the bed room is: 1910.0319  
D:\Java Programs>_
```

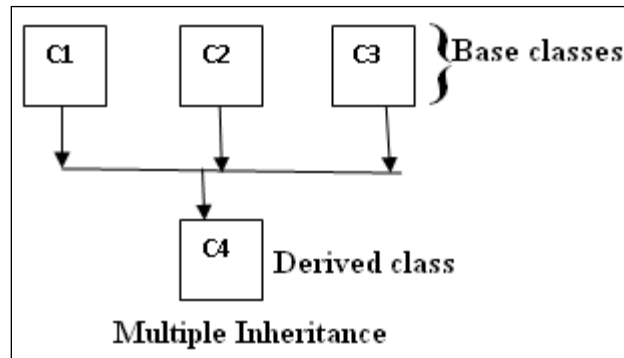
Multilevel Inheritance

It is one in which it contains one base class and one derived class and multiple intermediate base classes.(An intermediate base class is one, in one context it acts as a derived class, in another context the same class acts as a base class). It is shown in below.



Multiple Inheritance

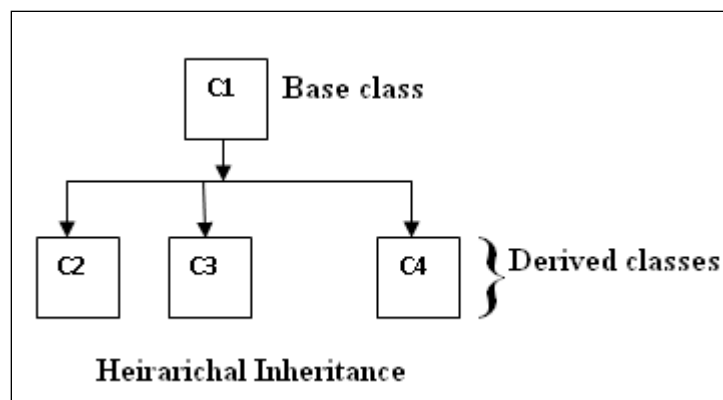
It is one in which it contains multiple base classes and a single derived class. It is shown in below.



The concept of multiple inheritance is not supported in java directly but it can be supported through the concept of interfaces.

Hierarchical Inheritance

It is one in which there exists a single base class and multiple derived classes. It is shown in below.



Q. Explain about the super keyword in java.

Subclass Constructor ('super' keyword)

A Subclass constructor is used to construct the instance variables of both the subclass and the super class. The subclass constructor uses the keyword super to invoke the constructor method of the super class. The super keyword is used subject to the following conditions.

- **super** may only be used within a subclass constructor method.
- The call to super class constructor must appear as the first statement within the subclass constructor.
- The parameters in the **super** call must match the order and type of the instance variables declared in the super class.

Q. Explain about the method overriding in java.

The process of redefining the same method for many number of implementations is called as method overriding.

The method overriding is possible by defining a method in the sub class that has the same name, same arguments and same return type as a method in the super class.

Method Overriding = Method Heading is same + Method body is different.

The following program illustrates the concept of method overriding

Aim: To illustrate the concept of method overriding in java.

Program:

//OverrideTest.java

```
class Super
{
    int x;
    Super(int x)
    {
        this.x=x;
    }
    void display() // method defined
    {
        System.out.println("The value of x at super is: " + x);
    }
}
class Sub extends Super
{
    int y;
```

```

Sub(int x, int y)
{
    super(x);
    this.y=y;
}
void display() //method defined again
{
    System.out.println("The value of x at sub class is: " + x);
    System.out.println("The value of y at sub class is: " + y);
}
}
class OverrideTest
{
    public static void main(String args[])
    {
        Sub s1=new Sub(100,200);
        s1.display();
    }
}

```

Output:

```

C:\WINDOWS\system32\cmd.exe
D:\Java Programs>javac OverrideTest.java
D:\Java Programs>java OverrideTest
The value of x at sub class is: 100
The value of y at sub class is: 200
D:\Java Programs>_

```

Q. Explain about the final variables, final methods and final classes in java.

Final variables and methods

All methods and variables can be overridden by default in subclasses. If we wish to prevent the subclasses from overriding the members of the super class, we can declare them as final using the keyword final as a modifier. For example

```

final int size=100;

final void showStatus()
{
    -----
    -----
}

```

Making a method as final ensures that the functionality in this method will never be altered in any way. Similarly, the value of a final variable can never be changed.

Final class

If we don't want to give features of base class to derived class then the definition of the base class must be made as final.

Syntax: final class <clsname>

```
{  
    -----  
}
```

Example:

```
final class Personal  
{  
    int idno;  
    int pinno;  
    String pwd;  
    -----  
}  
class Others extends Personal // error  
{  
    -----  
}
```

Once the class is final, it never participates in inheritance process. In other words final classes cannot be extended or inheritable.

Note:

- Final variable values cannot be modified.
- Final methods cannot be overridden.
- Final classes cannot be inheritable.

Q. Explain the abstract methods and classes in java.

Abstract Method or Undefined method

An abstract method is one which does not contain any definition/body but it contains only prototype/declaration.

In order to make undefined methods as abstract methods, we need to write abstract keyword before the method prototype.

Syntax for abstract method

```
abstract return_type method_name ( list_args);
```

When we are declaring a method as an abstract method, that means, we can indicate that a method must always be redefined in a subclass, thus making overriding compulsory.

Example:

```
abstract void sum();  
abstract void sum(int x, int y);
```

abstract class

If a class is containing one or more abstract methods then that class should also be declared as an abstract class.

Syntax for abstract class

```
abstract class <clsname>  
{  
    -----  
    abstract return_type method_name ( list_args);  
    -----  
}
```

Example

```
abstract class OPerationsDemo  
{  
    abstract void sum();  
    abstract void mul();  
}
```

Here, the class OperationsDemo is an abstract class whose object is cannot be created directly but we can create indirectly.

Q. How can we pass variable number of arguments for a method in java.

(OR)

Explain methods with varargs in java.

Varargs represents variable length arguments in methods, which is one of the feature introduced by J2SE 5.0.

The varargs allows us to declare a method with the unspecified number of parameters for a given argument. The varargs is identified by the type of an argument followed by the ellipsis(...) and the name of a variable.

Varargs takes the following form

```
<access specifier> <static> void method_name (type...arguments)
{
}

```

In the above syntax, the method contains an argument called varargs in which type is the return type of the argument, ellipses(...) is the key to varargs and arguments is the name of the variable.

The following program illustrates the use of varargs in java.

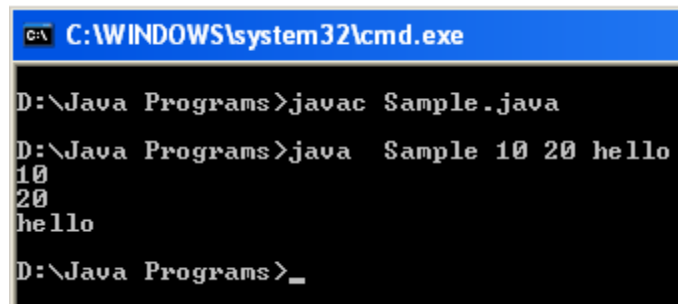
Aim: To illustrate the use of varargs to print the String value passed as an argument to a method.

Program:

```
//Sample.java
class Sample
{
    Sample(String...a)
    {
        for(int i=0;i<=a.length-1;i++)
        {
            System.out.println(a[i]);
        }
    }
    public static void main(String args[])
    {
        Sample s1=new Sample(args);
    }
}

```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\Java Programs>javac Sample.java
D:\Java Programs>java Sample 10 20 hello
10
20
hello
D:\Java Programs>_

```

Java Chapter-9

Arrays, Strings and Vectors

Arrays

Definition of an Array

- An array is a collection of same type variables referenced by a common name.
- All the elements in the array are of the same data type.
- Elements of an array are accessed individually using the index.
- The index value in every array starts with 0 (zero).
- An array can have one or more dimensions

One Dimensional Array

A list of items can be given one variable name using only one subscript and such a variable is called a single-dimensional variable or a one-dimensional array.

Creating an array

Creation of arrays is a two step process:

- 1) Declaring an array variable.
- 2) Allocating memory for the array.

Declaration of arrays

Syntax for declaring an array variable is:

```
type array-name[];
```

Here, type – Data type of the array and array-name is any valid identifier.

We can also declare an array in java as follows,

```
Type[] array-name;
```

Examples:

```
int number[];
```

```
float average[];
```

```
int[] counter;
```

Creation of arrays (Allocating the memory for an array)

After declaring an array, we need to create it in memory. Java allows us to create arrays using new operator only.

Syntax for allocating memory for an array:

```
array-name = new type[size];
```

Here, new – keyword used to allocate memory for the array, and

size – No of elements that the array will hold.

We can combine both the steps into a single one as shown below:

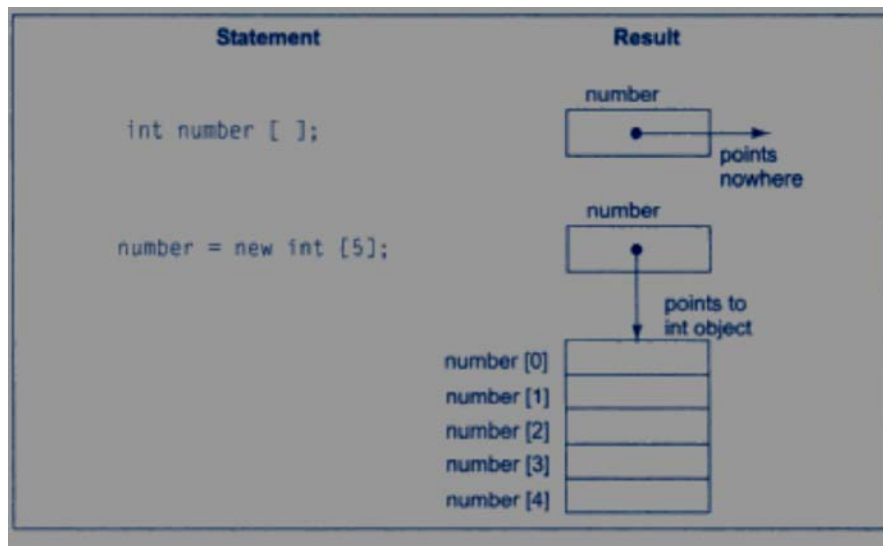
```
type array-name[] = new type[size];
```

Examples:

```
number=new int[5];
```

```
average=new float[10];
```

The following figure illustrates the creation of an array in memory.



Initialization of Arrays

- Every array element is accessed using its index value.
- Syntax for initializing the array elements is:

```
array-name[index] = value;
```

Example:

```
int a[] = new int[3];
```

```
a[0] = 10;
```

```
a[1] = 20;
```

```
a[2] = 30;
```

- There is another way for initializing array elements, right at the time of declaring the array.

Example:

```
int a[] = {1,2,3,4,5};
```

In the above example, 'a' is an integer type array with size automatically set to 5. All the 5 elements are initialized with the values 1,2,3,4,5.

a[0] is 1, a[1] is 2, a[2] is 3, a[3] is 4 and a[4] is 5.

Array Length

In java, all arrays store the allocated size in a variable named length. We can obtain the length of the array using the length.

Example:

```
int number[]={10,20,30,40,50};  
int n=number.length;
```

The following java program illustrates the concept of single dimensional arrays.

Aim: To sort an array of elements

Program:

//ArrayDemo.java

```
import java.io.*;  
class ArrayDemo  
{  
    public static void main(String[] args) throws IOException  
    {  
        int a[];  
        int n,temp;  
        DataInputStream in=new DataInputStream(System.in);  
        System.out.println("Enter the array size.....");  
        n=Integer.parseInt(in.readLine());  
        a=new int[n];  
        System.out.println("Enter the elements into the array");  
        for(int i=0;i<=n-1;i++)
```

```
{
    a[i]=Integer.parseInt(in.readLine());
}
System.out.println("The array elements before Sorting are....");
for(int i=0;i<=n-1;i++)
{
    System.out.println(a[i]);
}
System.out.println("The array elements after Sorting are.....");
{
    for(int i=0;i<=n-1;i++)
    {
        for(int j=i+1;j<=n-1;j++)
        {
            if(a[i]>=a[j])
            {
                temp=a[i];
                a[i]=a[j];
                a[j]=temp;
            }
        }
    }
}
for(int i=0;i<=n-1;i++)
{
    System.out.println(a[i]);
}
}
```

Output

```
C:\WINDOWS\system32\cmd.exe
D:\Java Programs>javac ArrayDemo.java
Note: ArrayDemo.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
D:\Java Programs>java ArrayDemo
Enter the array size.....
3
Enter the elements into the array
5
2
10
The array elements before Sorting are....
5
2
10
The array elements after Sorting are.....
2
5
10
D:\Java Programs>_
```

Two-Dimensional Arrays

- Until now we have only seen one-dimensional array. But, an array in general can have more than one dimension.
- Let's see about two-dimensional arrays. For each dimension, we must specify an extra set of [].

A two dimensional array is declared as shown below:

```
int a[][] = new int[3][4];
```

'a' is a two dimensional array which can hold 12 elements in 3 rows and 4 columns.

- Initializing two dimensional array elements is done using the index values:

```
int a[][] = new int[2][2];
```

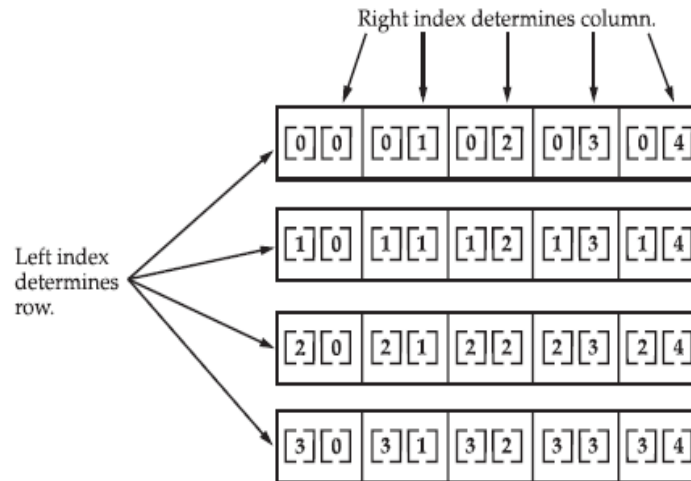
```
a[0][0] = 1;
```

```
a[0][1] = 2;
```

```
a[1][0] = 3;
```

```
a[1][1] = 4;
```

The following is a representation of a two-dimensional array in memory.



- We can directly initialize the elements of a two dimensional array as shown below:
`int a[][] = { {1,2},{3,4}};`
- Alternative way of declaring a two dimensional array is:
`int[][] a = {{1,2},{3,4}};`

The following program illustrates the concept of two-dimensional arrays in java.

Aim: To read and print a matrix.

Program:

//ArrayDemo1.java

```
import java.io.*;
class ArrayDemo1
{
    public static void main(String[] args) throws IOException
    {
        int a[][];
        int m,n;
        DataInputStream in=new DataInputStream(System.in);
        System.out.println("Enter the number of rows of a matrix.....");
        m=Integer.parseInt(in.readLine());
        System.out.println("Enter the number of Columns of a matrix.....");
```

```
n=Integer.parseInt(in.readLine());
a=new int[m][n];
System.out.println("Enter the elements into the matrix....");
for(int i=0;i<m;i++)
{
    for(int j=0;j<n;j++)
    {
        a[i][j]=Integer.parseInt(in.readLine());
    }
}
System.out.println("The matrix elements are....");
for(int i=0;i<=n-1;i++)
{
    for(int j=0;j<n;j++)
    {
        System.out.print(" " + a[i][j]);
    }
    System.out.println("\n");
}
} //main()
} //ArrayDemo1
```

Output

```
C:\ C:\WINDOWS\system32\cmd.exe
D:\Java Programs>javac ArrayDemo1.java
Note: ArrayDemo1.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
D:\Java Programs>java ArrayDemo1
Enter the number of rows of a matrix.....
2
Enter the number of Columns of a matrix.....
2
Enter the elements into the matrix....
1
2
3
4
The matrix elements are....
 1 2
 3 4
D:\Java Programs>
```

Variable Size Arrays

Java treats multidimensional arrays as “arrays of arrays”. It is possible to declare a two-dimensional array as follows.

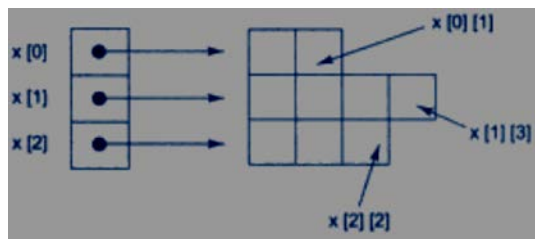
```
int x[][]=new int[3][];
```

```
x[0]=new int[2];
```

```
x[1]=new int[4];
```

```
x[2]=new int[3];
```

These statements create a two-dimensional array as having different lengths for each row as shown in below.



Strings

- A string is a sequence/group of characters.
- Strings in java are implemented as objects of the class “String” unlike other languages which implement strings using character arrays.
- Once a string (object) is created, we cannot change the characters in that string. Rather, we can create a new string object.
- All kind of string operations like concatenation, copying, case inversion and substring and many others can be performed on string objects.
- In java, **Strings are class objects** and implemented using two classes, namely, **String** and **StringBuffer**.
- A java String is not a character array and is not NULL terminated.

Strings may be declared and created as follows:

```
String stringname= new String(“string”);
```

Example:

```
String name=new String(“kanth”); //is same as
```

```
String name=”kanth”;
```

Like arrays, it is possible to get the length of string using the length method of the String class.

```
int len = name.length();
```

Java string can be concatenated using the + operator.

```
E.g.: String firstname = “sri”;
```

```
String lastname = “kanth”;
```

```
String name = firstname+lastname;
```

```
( or ) String name = “sri”+”kanth”;
```

String Arrays:

We can also create and use arrays that contain strings. The statement,

```
String names[]=new String[3];
```

will create an **names** array of size 3 to hold three string constants

String Methods: (Methods of String class)

The String class defines a number of methods that allow us to accomplish a variety of string manipulation tasks.

Method	Task
<code>s2=s1.toLowerCase()</code>	converts the String s1 to all lowercase
<code>s2=s1.toUpperCase()</code>	converts the String s1 to all Uppercase
<code>s2=s1.replace(_x','y');</code>	Replace all appearances of x with y
<code>s2=s1.trim();</code>	Remove white spaces at the beginning and end of String s1
<code>s1.equals(s2);</code>	Returns <code>__true'</code> if s1 is equal to s2
<code>s1.equalsIgnoreCase(s2)</code>	Returns <code>__true'</code> if s1=s2, ignoring the case of characters
<code>s1.length()</code>	Gives the length of s1
<code>s1.CharAt(n)</code>	Gives nth character of s1
<code>s1.compareTo(s2)</code>	Returns <code>-ve</code> if <code>s1<s2</code> , <code>+ve</code> if <code>s1>s2</code> , and <code>0</code> if s1 is equal s2
<code>s1.concat(s2)</code>	Concatenates s1 and s2
<code>s1.indexOf(_x')</code>	Gives the position of the first occurrence of <code>__x'</code> in string s1
<code>s1.indexOf(_x',n)</code>	Gives the position of <code>__x'</code> that occurs after nth position in the string s1

The following program illustrates the concept of String class in java

Aim: To sort the string in Alphabetical order

Program:

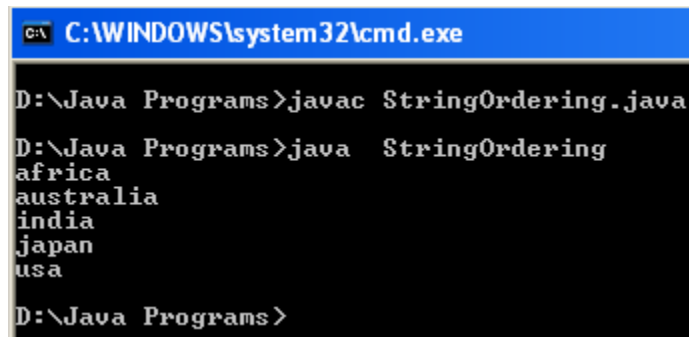
```
//StringOrdering.java
```

```
//Alphabetical ordering of strings
```

```
class StringOrdering
{
    public static void main(String args[])
    {
        String names[]={ "india","usa","australia","africa","japan"};
        int size=names.length;
        String temp;
        for(int i=0;i<size;i++)
```

```
    {
        for(int j=i+1;j<size;j++)
        {
            if(names[j].compareTo(names[i])<0)
            {
                temp=names[i];
                names[i]=names[j];
                names[j]=temp;
            }
        }
    }
    for(int i=0;i<size;i++)
    System.out.println(names[i]);
}
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\Java Programs>javac StringOrdering.java
D:\Java Programs>java StringOrdering
africa
australia
india
japan
usa
D:\Java Programs>
```

StringBuffer Class :

StringBuffer is a peer class of String. While **String** creates string of fixed length, **StringBuffer** creates strings of flexible length that can be modified in terms of both length and content. We can insert characters and substrings in the middle of a string, or append another string to the end. Below, there are some of methods that are frequently used in string manipulations.

Method	Task
s1.setCharAt(n, 'x')	Modifies the nth character to x
s1.append(s2)	Appends the string s2 to s1 at the end
s1.insert(n,s2)	Inserts the string s2 at the position n of the string s1
s1.setLength(n)	Sets the length of the string s1 to n. If n<s1.length() s1 is truncated .if n>s1.length() zeros are added to s1

The following program illustrates the concept of StringBuffer class in java.

Program:

```
class StringBufferConst
{
    public static void main(String args[])
    {
        StringBuffer buff1 = new StringBuffer();
        System.out.println("Length is: "+buff1.length());
        System.out.println("Capacity is: "+buff1.capacity());

        StringBuffer buff2 = new StringBuffer(20);
        System.out.println("Length is: "+buff2.length());
        System.out.println("Capacity is: "+buff2.capacity());

        StringBuffer buff3 = new StringBuffer("hello");
        System.out.println("Length is: "+buff3.length());
        System.out.println("Capacity is: "+buff3.capacity());

        CharSequence seq = "Hai";
        StringBuffer buff4 = new StringBuffer(seq);
        System.out.println("Length is: "+buff4.length());
        System.out.println("Capacity is: "+buff4.capacity());
    }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
D:\Java Programs>javac StringBufferConst.java
D:\Java Programs>java StringBufferConst
Length is: 0
Capacity is: 16
Length is: 0
Capacity is: 20
Length is: 5
Capacity is: 21
Length is: 3
Capacity is: 19
D:\Java Programs>
```

Q. Explain about the Vectors in java?

- **Vector** is a class contained in **java.util** package.
- It is used to create a generic dynamic array known as vector that can hold objects of any type and any number.
- The objects do not have to be homogeneous.

The vectors are created as follows

```
Vector vect=new Vector(); // declaring without size.
```

```
Vector list=new Vector(3);// declaring with size 3.
```

- A Vector without size can accommodate an unknown number of items.
- When a size is specified, this can be overlooked and a different number of items may be put into the vector.

Advantages of Vectors over Arrays

1. It is convenient to use vectors to store objects.
2. A vector can be used to store a list of objects that may vary in size.
3. We can add and delete objects from the list as and when required.

A major constraint in using vectors is that we cannot directly store simple data type in a vector, we only store objects. Therefore, we need to convert simple types to objects. This can be done using the wrapper classes.

The vector class supports a number of methods that can be used to manipulate the vectors created. The following table contains the important methods of a vector class.

Method call	Task Performed
List.addElement(item)	Adds the item specified to the list at the end.
List.elementAt(10)	Gives the name of the 10 th object.
List.size()	Gives the number of objects present.
List.removeElement(item)	Removes the specified item from the list.
List.removeElementAt(n)	Removes the item stored in the n th position of the list.
List.removeAllElements()	Removes all the elements in the list.
List.copyInto(array)	Copies all the elements from list to array.
List.insertElementAt(item,n)	Insert the item at the n th position.

The following program illustrates the concept of Vectors in java.

Aim: To work with the Vectors and arrays in java

Program

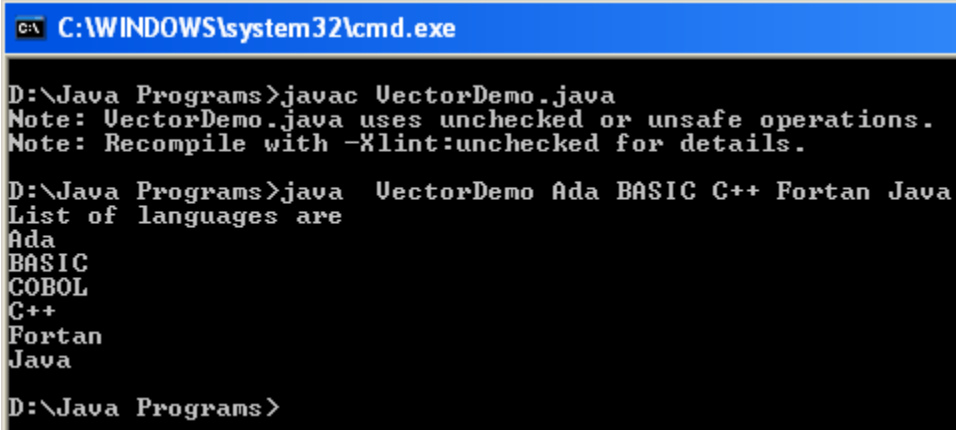
//Program to work with vectors and arrays

```
import java.util.*;
class VectorDemo
{
    public static void main(String[] args)
    {
        Vector v=new Vector();
        int len=args.length;
        for(int i=0;i<len;i++)
        {
```



```
        v.addElement(args[i]);
    }
    v.insertElementAt("COBOL",2);
    int size=v.size();
    String s[]=new String[size];
    v.copyInto(s);
    System.out.println("List of languages are");
    for(int i=0;i<size;i++)
    {
        System.out.println(s[i]);
    }
}
}
```

Output



```
C:\WINDOWS\system32\cmd.exe
D:\Java Programs>javac VectorDemo.java
Note: VectorDemo.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
D:\Java Programs>java VectorDemo Ada BASIC C++ Fortan Java
List of languages are
Ada
BASIC
COBOL
C++
Fortan
Java
D:\Java Programs>
```

Wrapper classes

The Vectors cannot handle the primitive data types like int, float, long , char and double. Primitive data types may be converted into object types by using the wrapper classes contained in the java.lang package. The following table shows the simple data types and their corresponding wrapper classes.

Simple Type	Wrapper Class
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long

The wrapper classes have a number of unique methods for handling primitive data types and objects. They are listed in the following table.

Constructor calling	Conversion Action
Integer intval=new Integer(i);	Primitive integer to Integer Object
Float floatval=new Float(f);	Primitive float to Float Object
Double dobval=new Double(d);	Primitive double to Double Object
Long longval=new Long(l);	Primitive long to Long Object

The following table shows the methods to convert Object numbers to Primitive Numbers using typeValue() method

Method Calling	Conversion Action
int i=intval.intValue();	Object to Primitive integer
float f=floatval.floatValue();	Object to primitive float
long l=longval.longValue();	Object to primitive long
double d=dobval.doubleValue();	Object to primitive double

The following table shows the methods to convert numbers to strings using toString() method

Method Calling	Conversion Action
str=Integer.toString(i);	Primitive integer to string
str=Float.toString(f);	Primitive float to string
str=Double.toString(d);	Primitive double to string
str=Long.toString(l);	Primitive long to string

The following table shows the methods to convert String objects to numeric Objects using the static method ValueOf().

Method Calling	Conversion Action
dobval=Double.ValueOf(str);	Converts strings to Double object
floatval=Float.ValueOf(str);	Converts strings to Float object
intval=Integer.ValueOf(str);	Converts strings to Integer object
longval=Long.ValueOf(str);	Converts strings to Long object

The following table shows the methods to convert Numeric Strings to Primitive Numbers using Parsing Methods.

Method Calling	Conversion Action
int i=Integer.parseInt(str);	Converts string to primitive integer
long l=Long.parseLong(str);	Converts string to primitive long

Chapter-10

Interfaces: Multiple Inheritance

An interface is a special case of **abstract class**, which contains all the final variables and abstract methods (methods without their implementation). An interface specifies what a class must do, but not how to do.

Using the keyword *interface*, we can fully abstract a class interface from its implementation. Interfaces are syntactically similar to classes, but they lack instance variable, and their methods are declared without anybody. Once it is defined, any number of classes can implement an interface. Also, one class can implement any number of interfaces.

Defining interface

An interface is defined much like a class. This is the general form of an interface:

```
access interface interface_name
{
    type final_varname1=value;
    type final_varname2=value;
    ....
    Returntype method-name1(parameter_list);
    returntype method-name2(parameter_list);
    ....
}
```

Here, *access* is either *public* or not used. *Interface_name* can be any valid identifier. Methods which are declared have no bodies. They end with a semicolon after the parameter list. They are explicitly abstract methods. Variables are implicitly final and static, meaning they cannot be changed by the implementing class. They must be initialized with a constant value. All the methods and variable are implicitly public if the interface, itself is declared as public.

Example:

```
interface Item
{
    static final int code=1001;
    static final String name="Fan";
    void display(); //abstract method
}
```

Note that the code for the method is not included in the interface and the method declaration simply ends with a semicolon. The class that implements this interface must define the code for the method.

Q. Explain the differences between class and interface

class	Interface
The members of a class can be constant or variables.	The members of an interface are always declared as constant, i.e., their values are final.
The class definition can contain the code for each of its methods. That is, the methods can be abstract or non-abstract.	The methods in an interface are abstract in nature,i.e., there is no code associated with them. It is later defined by the class that implements the interface.
It can be instantiated by declaring objects.	It cannot be used to declare objects. It can only implemented by a class.
It can use various access specifiers like public, private, or protected.	It can only use the public access specifier.

Extending Interfaces

Like classes, interfaces can also be extended. That is, an interface can be sub interfaced from other interfaces. The new sub interface will inherit all the members of the super interface in the manner similar to sub classes. This is achieved using the keyword extends as shown in below.

```
interface name2 extends name1
{
    Body of name2;
}
```

Consider the following example,

```
interface ItemConstants
{
    int code=1001;
    String name="Fan";
}
interface Item extends ItemConstants
{
    void display();
}
```

```
class Display implements Item
{
    public void display()
    {
        System.out.println(code + " " + name);
    }
}
```

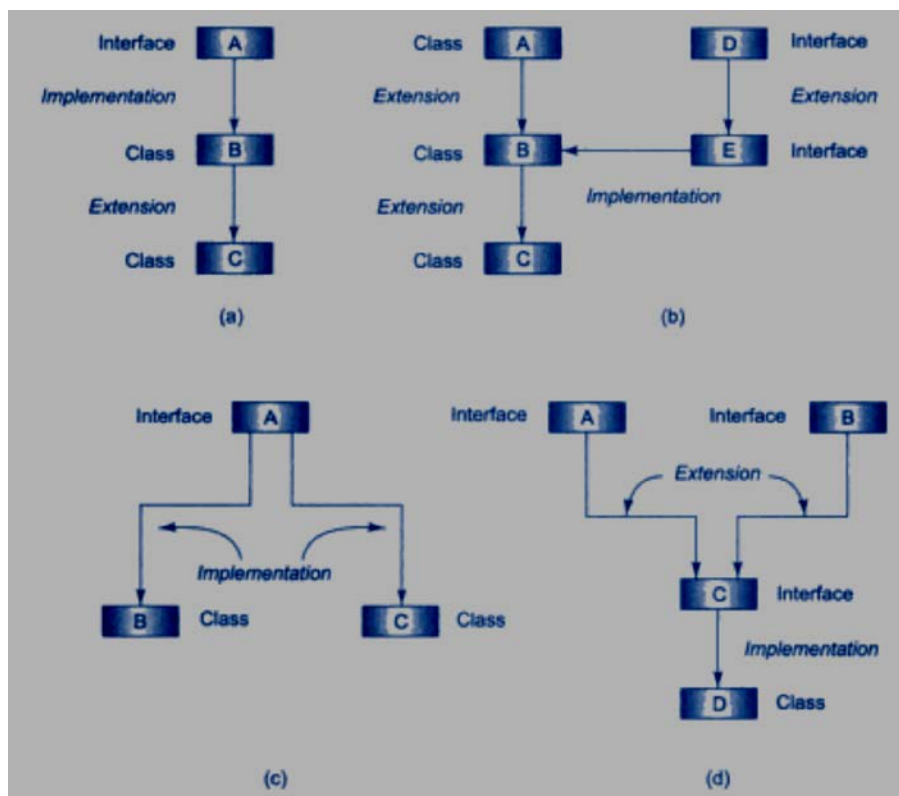
Implementing Interfaces

Once an interface has been defined, one or more classes can implement that interface. To implement an interface, include the **implements** clause in a class definition, and then create the methods defined by the interface. The general form of a class that includes the implements clause looks like this:

```
class class_name implements interface_name
{
    Body of class name
}
```

Here the class *class_name* “implements” the interface *interface_name*.

When a class implements more than one interface, they are separated by a comma. The implementation of interfaces can take various forms as illustrated below.



The following program illustrates the concept of interfaces in java.

//InterfaceTest.java

```
interface Area //Interface defined
{
    final static float pi=3.14f;
    float compute(float x,float y);
}
class Rectangle implements Area
{
    public float compute(float x, float y)
    {
        return (x*y);
    }
}
class Circle implements Area
{
    public float compute(float x,float y)
    {
        return (pi*x*y);
    }
}
class InterfaceTest
{
    public static void main(String args[])
    {
        Rectangle rect=new Rectangle();
        Circle cir=new Circle();
        Area area;
        area=rect;
        System.out.println("Area of Rectangle=" + area.compute(10,20));
        area=cir;
        System.out.println("area of circle=" + area.compute(10,10));
    }
}
```

Output:

```
C:\WINDOWS\system32\cmd.exe
D:\Java Programs>javac InterfaceTest.java
D:\Java Programs>java InterfaceTest
Area of Rectangle=200.0
area of circle=314.0
D:\Java Programs>
```

Accessing Interface Variables

Interfaces can also be used to declare a set of constants that can be used in different classes. The constant values will be available to any class that implements the interface. The values can be used in any method, as part of any variable declaration, or anywhere where we can use a final value.

Example

```
interface A
{
int m=10;
int n=50;
}
class B implements A
{
void method(int size)
{
-----
-----
if(size<n)
-----
}
}
```

Multiple Inheritance

It is one in which it contains multiple base classes and a single derived class. It is not possible through classes in java but it is possible through interfaces.

The following program illustrates the concept of implementing multiple inheritance in java.

Aim: Implementing the Multiple Inheritance in java

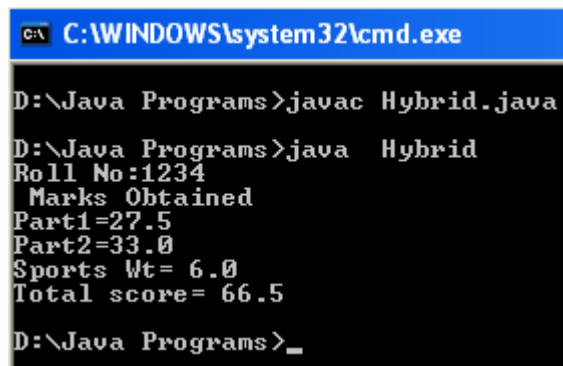
//Hybrid.java

//Program to illustrate the concept of Implementing the Multiple inheritance in java.

```
class Student
{
    int rno;
    void getNumber(int n)
    {
        rno=n;
    }
    void putNumber()
    {
        System.out.println("Roll No:" + rno);
    }
}
class Test extends Student
{
    float part1,part2;
    void getMarks(float m1,float m2)
    {
        part1=m1;
        part2=m2;
    }
    void putMarks()
    {
        System.out.println(" Marks Obtained");
        System.out.println("Part1=" + part1);
        System.out.println("Part2=" + part2);
    }
}
interface Sports
{
    float sportWt=6.0f;
    void putWt();
}
class Results extends Test implements Sports
{
```

```
float total;
public void putWt()
{
    System.out.println("Sports Wt= " + sportWt);
}
void display()
{
    total=part1+part2+sportWt;
    putNumber();
    putMarks();
    putWt();
    System.out.println("Total score= " + total);
}
}
class Hybrid
{
    public static void main(String args[])
    {
        Results std1=new Results();
        std1.getNumber(1234);
        std1.getMarks(27.5f,33.0f);
        std1.display();
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\Java Programs>javac Hybrid.java
D:\Java Programs>java Hybrid
Roll No:1234
Marks Obtained
Part1=27.5
Part2=33.0
Sports Wt= 6.0
Total score= 66.5
D:\Java Programs>_
```

Java Chapter-11

Packages: Putting Classes Together

A Package is a collection of classes, interfaces and sub-packages. A Sub package in turns divides into classes, interfaces and sub-sub-packages, etc.

Learning about JAVA is nothing but learning about various packages. By default one predefined package is imported for each and every java program and whose name is java.lang.*.

In java the packages are classified into two categories. They are

- Java API packages (Predefined Packages)
- User defined Packages.

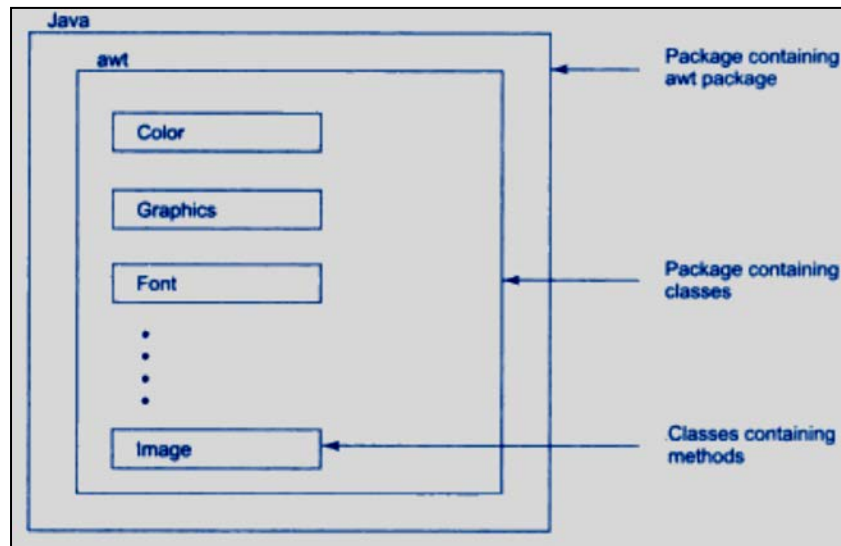
Java API packages

Java API provides a large number of classes grouped into different packages according to their functionalities. They are shown in below.

Package Name	Package Description
Java.lang.*	This package is used for achieving the language functionalities such as conversion of data from string to fundamental data, displaying the results on to the console, obtaining the garbage collector. This is the package which is by default imported for each and every java program.
Java.io.*	This package is used for developing file handling applications, such as , opening the file in read or write mode, reading or writing the data, etc.
Java.awt.* (abstract window toolkit)	This package is used for developing GUI(Graphical User Interface) components such as buttons, check boxes, scroll boxes, etc.
Java.applet.*	This package is used for developing browser oriented applications. In other words this package is used for developing distributed applications. An applet is a java program which runs in the context of www or browser.
Java.util.*	This package is used for developing quality or reliable applications in java. This package contains various classes and interfaces which improves the performance of J2ME applications. This package is also known as collection framework.
Java.net.*	This package is used for developing client server applications.

Using System Packages

The packages are organized in a hierarchical structure as shown in below.



There are two ways of accessing the classes stored in a package. The first approach is to use the fully qualified class name of the class that we want to use. This is done in the following way.

Example: `import java.awt.Colour`

Note that **awt** is a package within the package **java** and the hierarchy is representing by separating levels with dot.

The second approach is importing all the classes in a package at a time. This is done in the following way.

Example: `import java.awt.*;`

The statements,

```
import packagename.classname;
Or
import java.packagename.*;
```

Are known as import statements and must appear at the top of the file, before any class declarations, import is a keyword.

Note: Whenever we create user defined package statement as a part of java program, we must use package statement as a first executable statement.

Defining & creating package:

Step1:

Simply include a **package command** has the first statement in java source file. Any class you declare within that file will belong to the specified package.

Syntax: package packagename;

E.g.: package mypack;

Step 2:

Next **define the class** that is to be put in the package and declare it as public.

Step 3:

Now store the **classname.java file** in the directory having the name same as package name.

Step 4:

File is to be compiled as follows.

C:\> javac -d . classname.java

which creates **.class** file in the directory. Java also supports the package hierarchy, which allows grouping related classes into a package and then grouping related packages into a larger package. We can achieve this by specifying multiple names in a package statement, separated by dot.

i.e., package firstpackage.secondpackage;

Accessing Package

A java system package can be accessed either by using a fully qualified class name or by using import statement. We generally use import statement.

Syntax:

```
import pack1[.pack2][.pack3].classname;
```

OR

```
import pack1[.pack2][.pack3].*;
```

Here pack1 is the top level package, pack2 is the package which is inside in pack1 and so on. In this way we can have several packages in a package hierarchy. We should specify explicit class name finally. Multiple import statements are valid. * indicates that the compiler should search this entire package hierarchy when it encounters a class name.

Importing Packages

Java includes import statement to bring certain classes or entire package into visibility. In a java source file import statement occurs immediately following package statement and before any class definitions.

Syntax:

```
import pack1[.pack2].(classname/*);
```

Here pack1 is the name of the top level package. Pack2 is the name of the subordinate package separated by (.). finally classname / * indicates whether the java compiler should import the entire package or a part of it.

EX:

```
import java.util.Date.
```

Here java is main package; util is subordinate package, Date is the class belongs to util package.

Example Program (User defined Package)

Step1:

PackEg.Java

```
package p1;
public class PackEg
{
    public void display()
    {
        System.out.println("Welcome to java packages");
    }
}
```

Step2:

Compile the above program by using the following syntax.

```
C:\>javac -d . PackEg.java
```

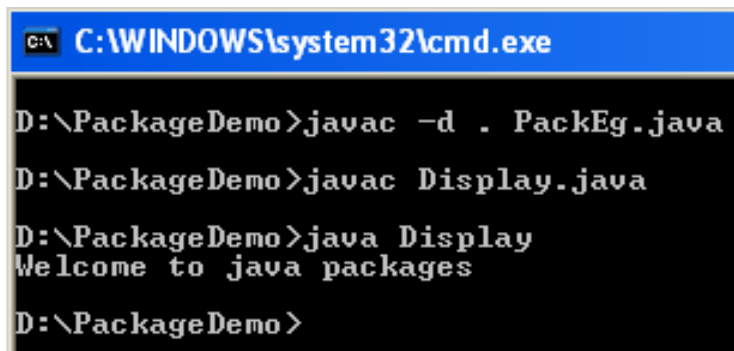
Then it creates a directory having the name of the package (PackEg) and it contains the .class file for our program.

Step3:

PackDemo.java

```
import p1.PackEg  
class Display  
{  
    public static void main(String args[])  
    {  
        PackEg p=new PackEg();  
        p.display();  
    }  
}
```

Now we will get the output as “Welcome to Java Packages” as follows.



```
C:\WINDOWS\system32\cmd.exe  
D:\PackageDemo>javac -d . PackEg.java  
D:\PackageDemo>javac Display.java  
D:\PackageDemo>java Display  
Welcome to java packages  
D:\PackageDemo>
```

Static Import

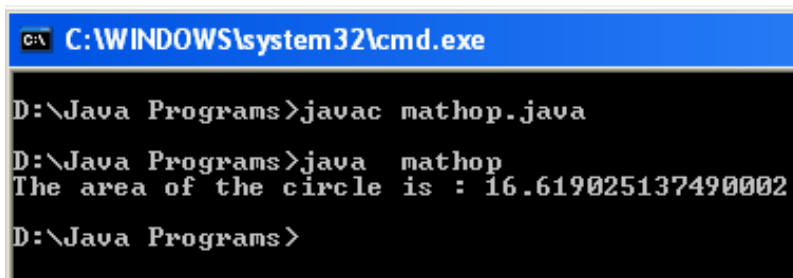
This feature eliminates the need of qualifying a static member with the class name. the static import declaration is similar that of import. We can use the import statement to import classes from packages and use them without qualifying the package. The syntax for using the static import feature is as follows.

```
import static package-name.subpackage-name.class-name.*;
```

The following program illustrates the concept of using the static import statement is as follows.

```
import static java.lang.Math.*;
public class mathop
{
    public void circle(double r)
    {
        double area=PI * r*r;
        System.out.println("The area of the circle is : " + area);
    }
    public static void main(String args[])
    {
        mathop obj=new mathop();
        obj.circle(2.3);
    }
}
```

Output



```
C:\WINDOWS\system32\cmd.exe
D:\Java Programs>javac mathop.java
D:\Java Programs>java mathop
The area of the circle is : 16.619025137490002
D:\Java Programs>
```

Chapter-12

Multithreaded Programming

Multithreading:

It means handling multiple tasks simultaneously. Java supports multithreaded programs. This means that we need not wait for the application to finish one task before another. For e.g. we can listen to an audio clip while scrolling a page and at the same time download an applet from a distant computer. This feature greatly improves the interactive performance of graphical applications.

Thread:

Thread is a sequence of instructions that is executed to define a unique flow of control. It is the smallest unit of code.

Differences between Multithreading and Multitasking

Multithreading	Multitasking
It is a programming concept in which a program or process is divided into two or more subprograms or threads that are executed at the same time in parallel.	It is an operating system concept in which multiple tasks are performed simultaneously.
It supports execution of multiple parts of a single program simultaneously.	It supports execution of multiple programs simultaneously.
The processor has to switch between different parts or threads of a program.	The processor has to switch between different programs or processes.
It is highly efficient.	It is less efficient in comparison to multithreading.
A thread is the smallest unit in multithreading.	A Program or process is the smallest unit in a multitasking environment.
It helps in developing efficient programs.	It helps in developing efficient operating systems.
It is cost-effective in case of context switching.	It is expensive in case of context switching.

Creating Threads:

Threads are implemented in the form of objects that contain a method called **run()**. The **run()** method is the heart and soul of any thread. A typical run() would appear as follows.

```
public void run()
{
    .....
    (statements for implementing thread)
    .....
```

}

The run () method should be invoked by an object of the concerned thread. This can be achieved by creating the thread and initiating it with the help of another thread method called **start ()**. A new thread can be created in two ways:

1. By **extending Thread** class: Define a class that extends **Thread** class and override its run () method with the code required by the thread.
2. By **implementing Runnable** interface: Define a class that implements Runnable interface. The Runnable interface has only one method, run (), that is to be defined in the method with the code to be executed by the thread.

Extending Thread Class

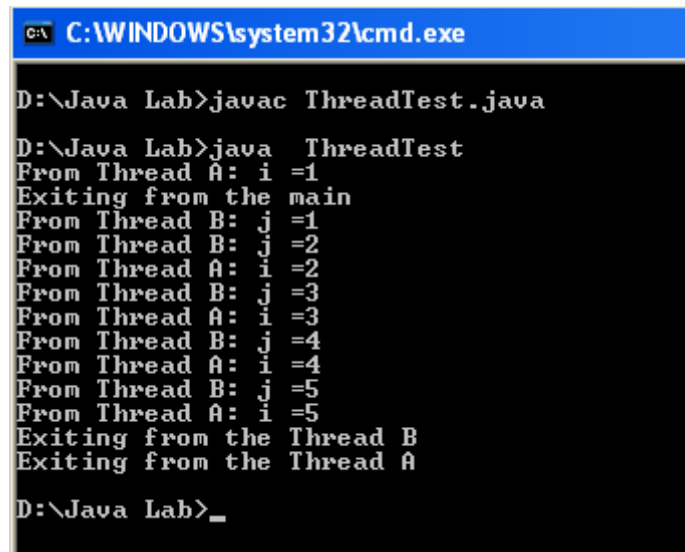
We can make our class runnable as thread by extending the class java.lang.Thread. This gives us access to all the thread methods directly. It includes the following steps:

1. Declare the class as extending the Thread class
2. Implement the run () method that is responsible for executing the sequence of code that the thread will execute.
3. Create a thread object and call the start () method to initiate the thread execution.

The following program illustrates the concept of extending the thread class.

```
class A extends Thread
{
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println("From Thread A: i =" + i);
        }
        System.out.println("Exiting from the Thread A");
    }
}
class B extends Thread
{
    public void run()
    {
        for(int j=1;j<=5;j++)
        {
            System.out.println("From Thread B: j =" + j);
        }
    }
}
```

```
        System.out.println("Exiting from the Thread B");
    }
}
class ThreadTest
{
    public static void main(String args[])
    {
        A a=new A();
        B b=new B();
        a.start();
        b.start();
        System.out.println("Exiting from the main");
    }
}
```

Output:

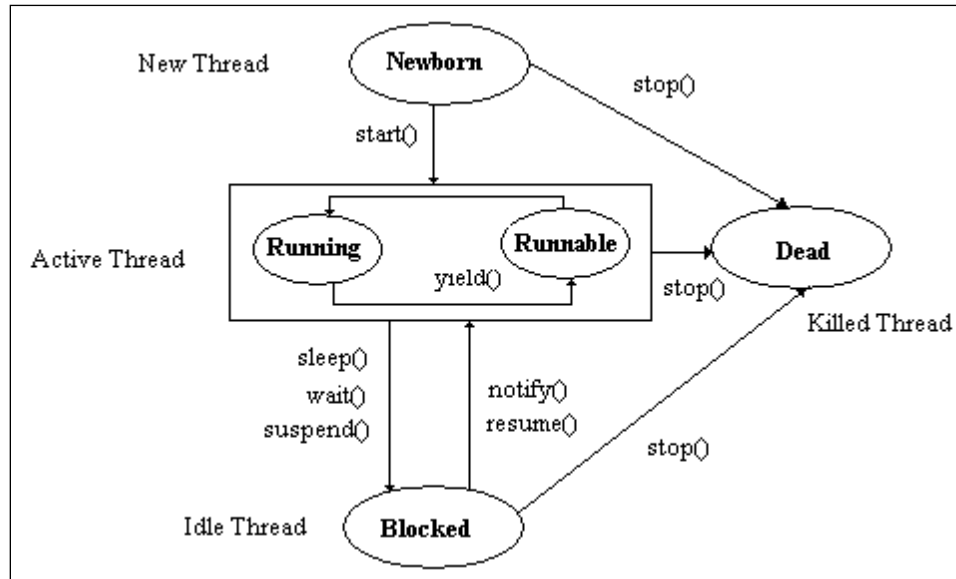
```
C:\WINDOWS\system32\cmd.exe
D:\Java Lab>javac ThreadTest.java
D:\Java Lab>java ThreadTest
From Thread A: i =1
Exiting from the main
From Thread B: j =1
From Thread B: j =2
From Thread A: i =2
From Thread B: j =3
From Thread A: i =3
From Thread B: j =4
From Thread A: i =4
From Thread B: j =5
From Thread A: i =5
Exiting from the Thread B
Exiting from the Thread A
D:\Java Lab>_
```

Lifecycle of a Thread

During the lifetime of a thread, there are many states it can enter. They include:

1. NewBorn State
2. Runnable State
3. Running State
4. Blocked State
5. Dead State

A thread is always in one of these five states. It can move from one state to another via a variety of ways as shown in below figure.



NewBorn State

When we create a thread object, the thread is born and is said to be in *newborn* state. The thread is not yet scheduled for running. At this state, we can do only one of the following things with it:

- Schedule it for running using `start()` method
- Kill it using `stop()` method

Runnable State

The runnable state means that the thread is ready for execution and is waiting for the availability of the processor. If we want a thread to relinquish control to another thread to equal priority before its turn comes, we can do so by using the **`yield()`**

Running State

Running means that the processor has given its time to the thread for its execution. The thread runs until it relinquishes control on its own or it is preempted by a higher priority thread.

Blocked State

A thread is said to be blocked when it is prevented from entering into the runnable state and subsequently the running state. This happens when the thread is suspended, sleeping, or

waiting in order to satisfy certain requirements. A blocked thread is considered —not runnable|| but not dead and therefore fully qualified to run again.

Dead State

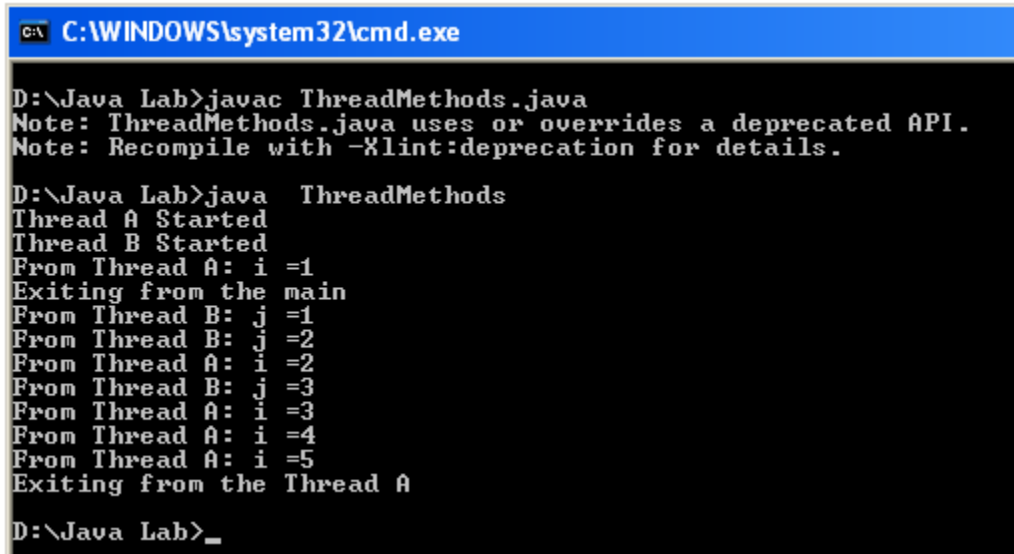
Every thread has a lifecycle. A running thread ends its life when it has completed executing its run() method. It is natural death. However, we can kill it by sending the stop message to it at any state thus causing a premature death to it. It is done by stop() method.

The following program illustrates the concept of using the thread methods.

```
class A extends Thread
{
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            if(i==1)
                yield();
            System.out.println("From Thread A: i = " + i);
        }
        System.out.println("Exiting from the Thread A");
    }
}
class B extends Thread
{
    public void run()
    {
        for(int j=1;j<=5;j++)
        {
            System.out.println("From Thread B: j = " + j);
            if(j==3)
                stop();
        }
        System.out.println("Exiting from the Thread B");
    }
}
class ThreadMethods
{
    public static void main(String args[])
    {
        A a=new A();
        B b=new B();
        System.out.println("Thread A Started");
        a.start();
        System.out.println("Thread B Started");
    }
}
```

```
        b.start();
        System.out.println("Exiting from the main");
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\Java Lab>javac ThreadMethods.java
Note: ThreadMethods.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
D:\Java Lab>java ThreadMethods
Thread A Started
Thread B Started
From Thread A: i =1
Exiting from the main
From Thread B: j =1
From Thread B: j =2
From Thread A: i =2
From Thread B: j =3
From Thread A: i =3
From Thread A: i =4
From Thread A: i =5
Exiting from the Thread A
D:\Java Lab>_
```

Thread Priority

In java, each thread is assigned a priority, which effects the order in which it is scheduled for running. The Threads of the same priority are given equal treatment by the java scheduler and, therefore, they share the processor on a First-Come, First-Serve basis.

Java permits us to set the priority of a thread using the `setPriority()` method as follows.

```
ThreadName.setPriority(Number);
```

The Number is an integer value to which the threads priority is set. The Thread class defines several priority constants.

```
MIN_PRIORITY=1
```

```
NORM_PRIORITY=5
```

```
MAX_PRIORITY=10
```

Whenever multiple Threads are ready for execution, the Java system chooses the highest priority thread and executes it. For a thread of lower priority to gain control, one of the following things should happen.

1. It stops running at the end of run().
2. It is made to sleep using sleep().
3. It is told to wait using wait().

The following program illustrates the concept of thread priorities in java.

//Program to illustrates the concept of Thread Priority

//ThreadPriority.java

```
class A extends Thread
{
    public void run()
    {
        System.out.println("Thread A Started....");
        for(int i=1;i<=5;i++)
        {
            System.out.println("\t From Thread A : i= " + i);
        }
        System.out.println("Exit from Thread A");
    }
}
class B extends Thread
{
    public void run()
    {
        System.out.println("Thread B Started....");
        for(int j=1;j<=5;j++)
        {
            System.out.println("\t From Thread B : j= " + j);
        }
        System.out.println("Exit from Thread B");
    }
}
class C extends Thread
{
    public void run()
    {
        System.out.println("Thread C Started....");
        for(int k=1;k<=5;k++)
        {
            System.out.println("\t From Thread C : k= " + k);
        }
        System.out.println("Exit from Thread C");
    }
}
```

```
class ThreadPriorityDemo
{
    public static void main(String args[])
    {
        A threadA=new A();
        B threadB=new B();
        C threadC=new C();

        threadC.setPriority(Thread.MAX_PRIORITY);
        threadB.setPriority(threadA.getPriority()+1);
        threadA.setPriority(Thread.MIN_PRIORITY);

        System.out.println("Start Thread A");
        threadA.start();
        System.out.println("Start Thread B");
        threadB.start();
        System.out.println("Start Thread C");
        threadC.start();

        System.out.println("Exit from main()");
    }
}
```

Output

```
C:\ C:\WINDOWS\system32\cmd.exe
D:\Java Lab>javac ThreadPriorityDemo.java
D:\Java Lab>java ThreadPriorityDemo
Start Thread A
Start Thread B
Thread A Started....
Start Thread C
    From Thread A : i= 1
Exit from main()
Thread C Started....
    From Thread C : k= 1
Thread B Started....
    From Thread A : i= 2
    From Thread B : j= 1
    From Thread C : k= 2
    From Thread B : j= 2
    From Thread B : j= 3
    From Thread B : j= 4
    From Thread B : j= 5
Exit from Thread B
    From Thread A : i= 3
    From Thread C : k= 3
    From Thread A : i= 4
    From Thread C : k= 4
    From Thread A : i= 5
    From Thread C : k= 5
Exit from Thread A
Exit from Thread C
D:\Java Lab>
```

Synchronizing Threads:

Synchronization of threads ensures that if two or more threads need to access a shared resource then that resource is used by only one thread at a time. You can synchronize your code using the **synchronized** keyword. You can invoke only one synchronized method for an object at any given time.

When a thread is within a synchronized method, all the other threads that try to call it on the same instance have to wait. During the execution of a synchronized method, the object is locked so that no other synchronized method can be invoked. The monitor is automatically released when the method completes its execution. The monitor can also be released when the synchronized method executes the `wait ()` method. When a thread calls the `wait ()` method, it temporarily releases the locks that it holds.

The Synchronized Statement

Synchronization among threads is achieved by using synchronized statements. The synchronized statement is used where the synchronization methods are not used in a class and you do not have access to the source code. You can synchronize the access to an object of this class by placing the calls to the methods defined by it inside a synchronized block.

Syntax:

```
synchronized(obj)
{
    // statements;
}
```

Implementing Runnable Interface

The Runnable interface declares the run () method that is required for implementing threads in our programs. To do this, we must perform the steps listed below:

- 1) Declare the class as implementing the Runnable interface.
- 2) Implement the run() method
- 3) call the thread's start() method to run the thread.

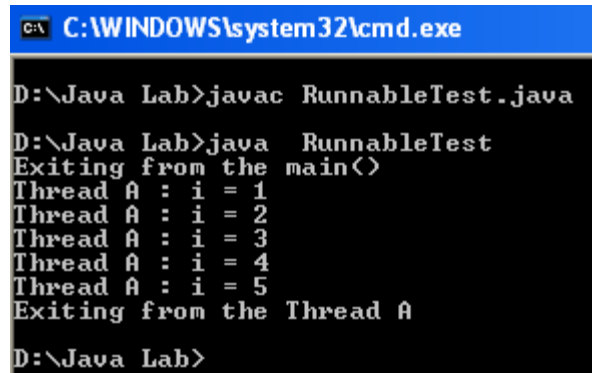
The following program illustrates the concept of implementing the Runnable Interface in java.

```
class A implements Runnable
{
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println("Thread A : i = " + i);
        }
        System.out.println("Exiting from the Thread A");
    }
}
class RunnableTest
{
    public static void main(String args[])
    {
        A ob=new A();

        Thread t=new Thread(ob);
        t.start();
    }
}
```

```
        System.out.println("Exiting from the main()");  
    }  
}
```

Output



```
C:\WINDOWS\system32\cmd.exe  
D:\Java Lab>javac RunnableTest.java  
D:\Java Lab>java RunnableTest  
Exiting from the main()  
Thread A : i = 1  
Thread A : i = 2  
Thread A : i = 3  
Thread A : i = 4  
Thread A : i = 5  
Exiting from the Thread A  
D:\Java Lab>
```

Inter-thread communication

Java supports inter-thread communication using `wait()`, `notify()`, `notifyAll()` methods. These methods are implemented as final methods in **Object**. So, all classes have them. All three methods can be called only from within a synchronized context.

- wait()** tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls **notify()**
- notify()** wakes up the first thread that called **wait()** on the same object.
- notifyAll()** wakes up all the threads that called **wait()** on the same object. The highest priority thread will run first.

Chapter-13

Managing Errors and Exceptions

An Exception is defined as —”**an abnormal error condition that arises during our program execution**”. When an Exception occurs in a program, the java interpreter creates an exception object and throws it out as java exceptions, which are implemented as objects of exception class. This class is defined in *java.lang* package. An Exception object contains data members that will store the exact information about the runtime error (Exception) that has occurred.

Types of Errors

Errors may broadly classified into two categories.

- Compile-time errors
- Run-time errors.

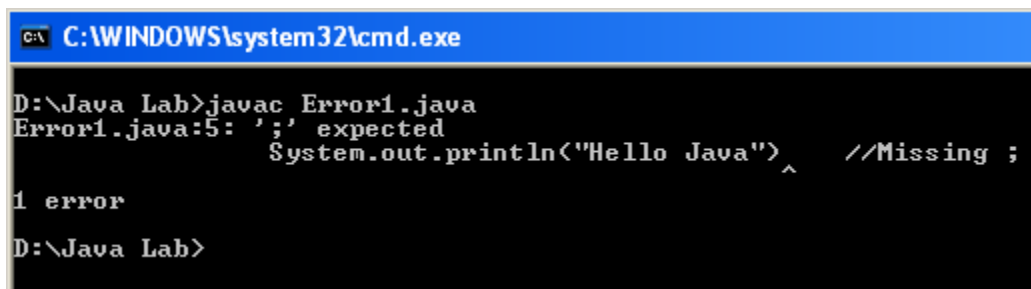
Compile-Time Errors

All syntax errors will be detected and displayed by the java compiler and therefore these errors are known as compile-time errors. Whenever the compiler displays an error, it will not create the .class file.

The following program is the illustration of Compile-time errors

```
class Error1
{
    public static void main(String args[])
    {
        System.out.println("Hello Java") //Missing ;
    }
}
```

When we compile the above program the java compiler displays the following error.

A screenshot of a Windows command prompt window. The title bar reads "C:\WINDOWS\system32\cmd.exe". The command prompt shows the following text:

```
D:\Java Lab>javac Error1.java
Error1.java:5: ';' expected
    System.out.println("Hello Java") ^ //Missing ;
1 error
D:\Java Lab>
```

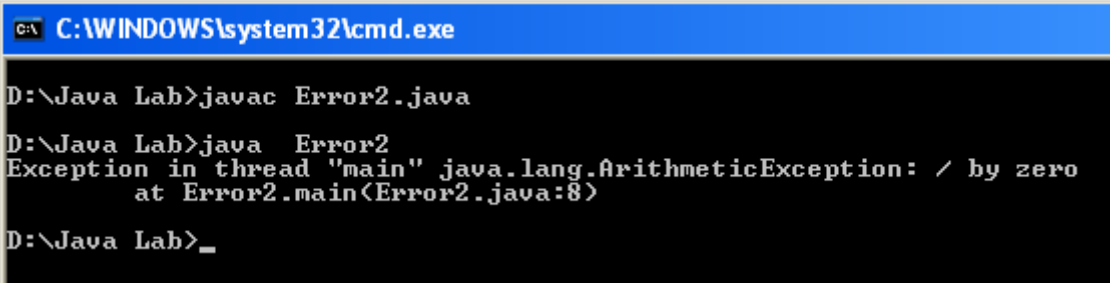
Runtime Errors

Sometimes, a program may compile successfully creating the .class file but may not run properly. Such programs may produce wrong results due to wrong logic or may terminate due to errors such as stack overflow.

The following program is the illustration of Run-time errors .

```
class Error2
{
    public static void main(String[] args)
    {
        int a=10;
        int b=5;
        int c=5;
        int x=a/(b-c); //Division by zero
        System.out.println("x=" + x);
        int y=a/(b+c);
        System.out.println("y=" + y);
    }
}
```

Output



```
C:\WINDOWS\system32\cmd.exe
D:\Java Lab>javac Error2.java
D:\Java Lab>java Error2
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Error2.main(Error2.java:8)
D:\Java Lab>_
```

Exception Handling

An exception is a condition that is caused by a run-time error in the program. When the java interpreter encounters an error such as dividing an integer by zero, it creates an exception object and throws it (informs us that an error occurs).

If the exception object is not caught and handled properly, the interpreter will display an error message as shown in the above output and will terminate the program.

If we want the program to continue with the execution of the remaining code, then we should try to catch the object thrown by the error condition and then display an appropriate message for taking corrective actions. This task is known as **Exception Handling**.

The following table shows the some common errors that are occurred in the java programs.

Exception Type	Cause of Exception
ArithmeticException	Caused by the math errors such as division by zero.
ArrayIndexOutOfBoundsException	Caused by bad array indexes.
FileNotFoundException	Caused by an attempt to access a non existing file.
NumberFormatException	Caused when a conversion between strings and numbers fails.
NullPointerException	Caused by referencing a null object.

Exceptions in java can be categorized into two types.

➤ **Checked Exceptions:**

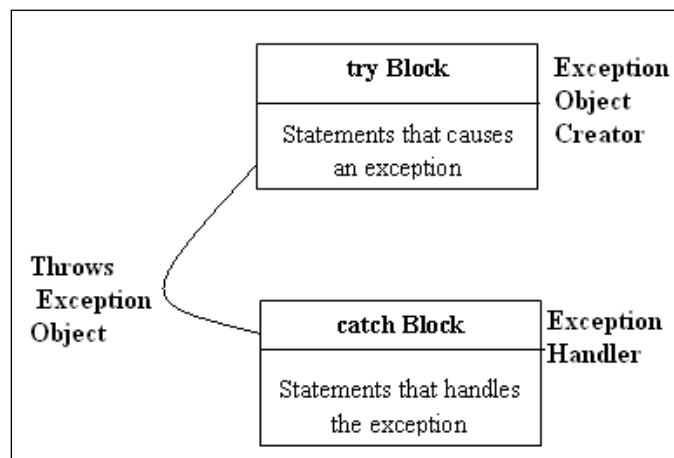
These exceptions are explicitly handled in the code itself with the help of try-catch blocks. Checked exceptions are extended from the java.lang.Exception class.

➤ **Unchecked Exceptions:**

These exceptions are not essentially handled in the program code, instead the JVM handles such exceptions. Unchecked exceptions are extended from the class java.lang.RuntimeException.

Syntax of Exception Handling Code

The basic concepts of exception handling are throwing an exception and catching it. This illustrates in the following figure.



Java uses the keywords try and catch to handles the exceptions in the java programs.

try block:

The statements that produces exception are identified in the program and the statements are placed in *try block*.

Syntax:

```
try
{
    //Statements that causes Exception
}
```

catch block:

The catch block is used to process the exception raised. The catch block is placed immediately after the try block.

Syntax:

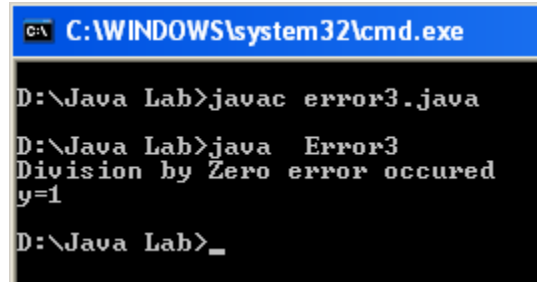
```
catch(ExceptionType ex_ob)
{
    //Statements that handle Exception
}
```

The following program illustrates the use of using the try and catch blocks in the java programs.

```
class Error3
{
    public static void main(String args[])
    {
        int a=10;
        int b=5;
        int c=5;
        int x,y;
        try
        {
            x=a/(b-c); //Division by zero
            System.out.println("x=" + x);
        }
        catch(ArithmeticException e)
        {
            System.out.println("Division by Zero error occured");
        }
    }
}
```

```
    }  
    y=a/(b+c);  
    System.out.println("y=" + y);  
  }  
}
```

Output



```
C:\WINDOWS\system32\cmd.exe  
D:\Java Lab>javac error3.java  
D:\Java Lab>java Error3  
Division by Zero error occured  
y=1  
D:\Java Lab>_
```

Note that the program did not stop at the point of exception condition. It catches the error condition, prints the error message.

The following is the program that catches the invalid command line arguments.

```
class Error4  
{  
    public static void main(String args[])  
    {  
        int number,invalid=0, valid=0;  
        for(int i=0;i<args.length;i++)  
        {  
            try  
            {  
                number=Integer.parseInt(args[i]);  
            }  
            catch(NumberFormatException e)  
            {  
                invalid=invalid+1;  
                System.out.println("Invalid number : " + args[i]);  
                continue;  
            }  
            valid=valid+1;  
        }  
        System.out.println("Valid numbers = " + valid);  
        System.out.println("Invalid numbers: " + invalid);  
    }  
}
```

}

Output

```

C:\WINDOWS\system32\cmd.exe
D:\Java Lab>javac Error4.java
D:\Java Lab>java Error4 10 abc 10.5 20
Invalid number : abc
Invalid number : 10.5
Valid numbers = 2
Invalid numbers: 2
D:\Java Lab>_

```

Nested Try:

A try block is placed inside the block of another try block is termed as Nested try block statements. If any error statement is in outer try block, it goes to the corresponding outer catch block. If any error statement is in inner try block first go to the inner catch block. If it is not the corresponding exception next goes to the outer catch, which is also not corresponding exception then terminated.

The following is the example program

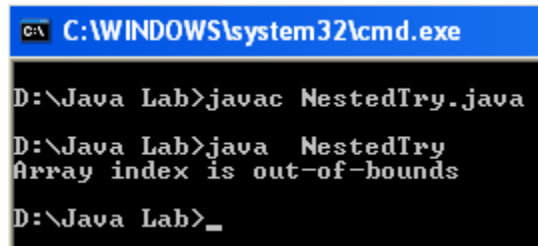
```

class NestedTry
{
    public static void main(String args[])
    {
        try
        {
            int a=2,b=4,c=2,x=7,z;
            int p[]={2};
            p[3]=33;
            try
            {
                z=x/((b*b)-(4*a*c));
                System.out.println("The value of x is= " + z);
            }
            catch (ArithmeticException e)
            {
                System.out.println("Division by zero in arithmetic expression");
            }
        }
        catch(ArrayIndexOutOfBoundsException e)

```

```
        {  
            System.out.println("Array index is out-of-bounds");  
        }  
    }  
}
```

Output



```
C:\WINDOWS\system32\cmd.exe  
D:\Java Lab>javac NestedTry.java  
D:\Java Lab>java NestedTry  
Array index is out-of-bounds  
D:\Java Lab>_
```

Multiple Catch Statements:

Multiple catch statements handle the situation where more than one exception could be raised by a single piece of code. In such situations specify two or more catch blocks, each specify different type of exception.

The following program illustrates this concept.

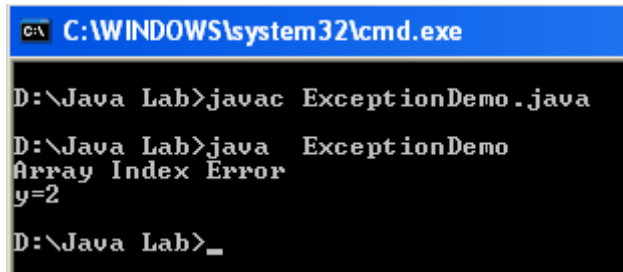
```
class ExceptionDemo  
{  
    public static void main(String args[])  
    {  
        int a[]={5,10};  
        int b=5;  
        try  
        {  
            int x=a[2]/(b-a[1]);  
        }  
        catch(ArithmeticException e)  
        {  
            System.out.println("Division by zero");  
        }  
        catch(ArrayIndexOutOfBoundsException e)  
        {  
            System.out.println("Array Index Error");  
        }  
        catch(ArrayStoreException e)  
        {  

```

```

        System.out.println("Wrong data type");
    }
    int y=a[1]/a[0];
    System.out.println("y=" + y);
}
}

```

Output


```

C:\WINDOWS\system32\cmd.exe
D:\Java Lab>javac ExceptionDemo.java
D:\Java Lab>java ExceptionDemo
Array Index Error
y=2
D:\Java Lab>_

```

Finally Statement

finally creates a block of code that will be executed after a try/catch block has completed. The finally block will execute whether or not an exception is thrown. If an exception is thrown, the finally block will execute even if no catch statement matches the exception.

Syntax:

```

finally
{
    // statements that executed before try/catch
}

```

Throwing our Own Exceptions (User defined Exceptions)

It is possible to create our own exception types to handle situations specific to our application. Such exceptions are called User-defined Exceptions. User defined exceptions are created by extending **Exception** class. The **throw** and **throws** keywords are used while implementing user-defined exceptions.

The following is the example for the user defined exceptions in java.

```

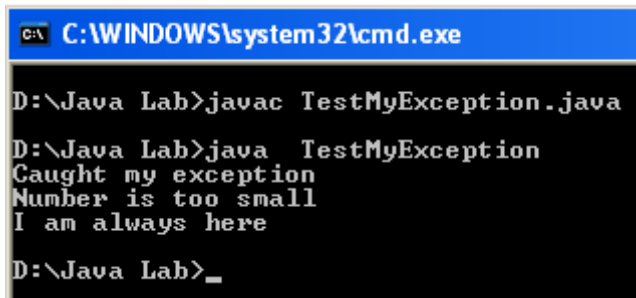
class MyException extends Exception
{
    MyException(String msg)
    {
        super(msg);
    }
}

```



```
class TestMyException
{
    public static void main(String args[])
    {
        int x=5,y=1000;
        try
        {
            float z=(float)x/(float)y;
            if(z<0.01)
            {
                throw new MyException("Number is too small");
            }
        }
        catch(MyException e)
        {
            System.out.println("Caught my exception");
            System.out.println(e.getMessage());
        }
        finally
        {
            System.out.println("I am always here");
        }
    }
}
```

Output:



```
C:\WINDOWS\system32\cmd.exe
D:\Java Lab>javac TestMyException.java
D:\Java Lab>java TestMyException
Caught my exception
Number is too small
I am always here
D:\Java Lab>_
```

APPLETS

13.1 Preparing to Write Applets

13.2 Applet Life Cycle

13.3 Applet Tag

13.4 Adding Applet to a HTML File

13.5 Running the Applet

13.1 PREPARING TO WRITE APPLETS

Until now we have been creating simple Java application program with a single main() method that created objects, set instance variables and ran methods. To write any applet, we will need to know:

1. When to use applets.
2. How an applet works,
3. What sort of features an applet has, and
4. Where to start, when we first create our own applet.

The following are the steps that are involved in developing and testing and applet.

1. Building an applet code(.java file)
2. Creating an executable applet(.class file)
3. Designing a web page using HTML
4. Preparing <Applet Tag>
5. Incorporating <Applet> tag into the web page.
6. Creating HTML file.
7. Testing the applet code.

To building the applet code two classes of java library are essential namely Applet and Graphics. The Applet class is contained in java.applet package provides life and behaviour to the applet through its methods such as init(), start() and paint(). Unlike with applications, where java calls the main() method directly to initiate the execution of the program, when an applet is

loaded java automatically calls a series of Applet class methods for starting running and stopping the applet code. The Applet class therefore maintains the life cycle of an applet.

Example:

```

public void paint(Graphics g)

import java.awt.*;

import java.applet.*;

.....

.....

public class applet classname extends Applet
{
.....

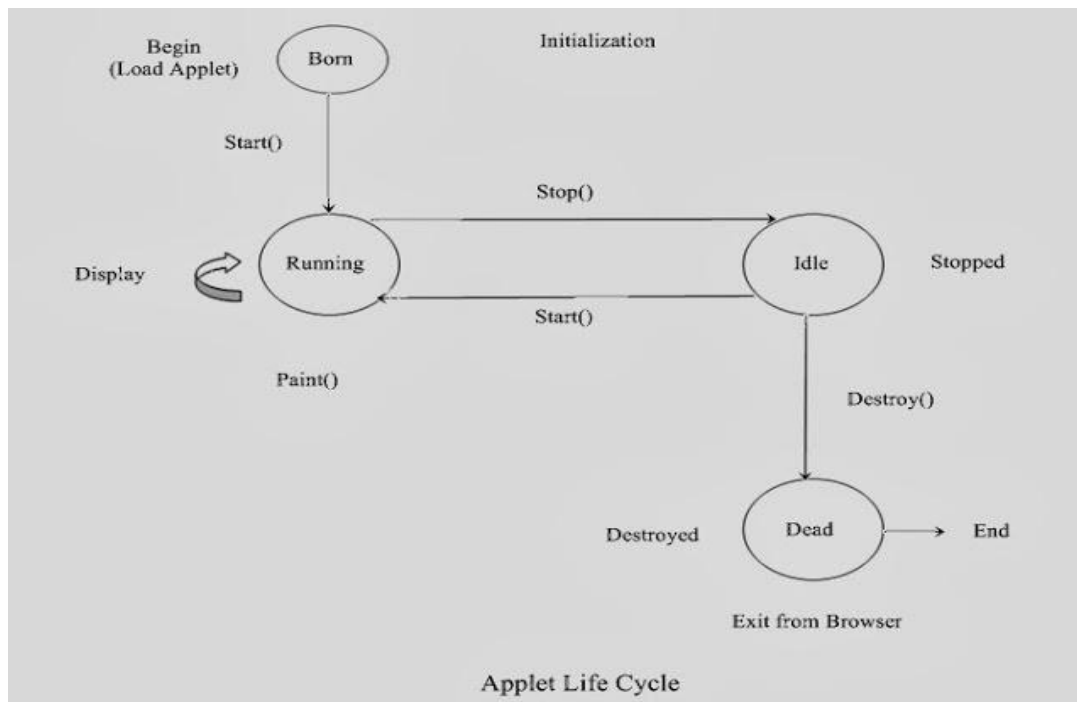
.....statements

public void paint(Graphics g)
{
.....//Applet operations code
}
.....
.....
}

```

13.2 APPLETS LIFE CYCLE

An applet is a window based event driven program and it waits until an event occurs. The AWT notifies the applet about an event by calling an event handler that has been provided by applet. Applet should not enter a "mode" of operation in which it maintains control for an extended period. In these situations you must start an additional thread of execution.



Syntax

```

public class AppProgram extends Applet
{
    public void init()
    {
        //initializtion
    }
    public void start()
    {
        //start or resume execution
    }
    public void stop()
    {
        //suspend execution
    }
}

```

```

}

public void destroy()
{
    //perform shutdown activities
}

public void paint(Graphics g)
{
    //redisplay contents of window
}
}

```

Initialization State

When the browser downloads an HTML page containing applets, it creates an instance for each of the Applet classes, using the no arg constructor. Applet must have a no argument constructor otherwise it cannot be loaded. Initialization can be done through `init()`. The `init()` method is the first method to be called. It is used to initialize the applet each time it is reloaded. Applets can be used for setting up an initial state, loading images or fonts, or setting parameters.

Syntax:

```

public void init()
{
    //code here
}

```

Running State

Immediately after calling `init()`, the browser calls the `start()` method. `start()` is also called when user returns to an HTML page that contains the applet. So, if the user leaves a web page and comes back, the applet resumes execution at `start()`. So, when the applet calls the `start()` method, it is called its running state. Staring can also occur if the applet is already in "stopped" (idle) state.

Syntax:

```

public void start()
{
.....
(Action) _____
}

```

Idle or Stopped State

When we leave the page containing the currently running applet, then it stop running and becomes idle. We can also do so by calling stop() Method explicitly.

Syntax:

```

public void stop()
{
.....
(Action) _____
}

```

Dead State

When an applet is completely removed from the Memory, it is called dead. This occurs automatically by invoking the destroy() method when we quit the browser Like initialization, dead state occurs only once in the applet's life cycle

Syntax:

```

public void destory()
{
.....
(Action) _____
}

```

Display State

Painting is how an applet displays something on screen-be it text, a line, a colored background, or an image. The paint() method is used for displaying anything on the applet paint() method takes an argument, an instance of class graphics. The code given can be as follows:

Syntax:

```
public void paint(Graphics g)
{
}
```

13.3 APPLET TAG

The <Applet...> tag supplies the name of the applet to be loaded and tells the browser how much space the applet requires. The ellipsis in the tag <Applet...> indicates that it contains certain attributes that must specified. The <Applet> tag given below specifies the minimum requirements to place the Hellojava applet on a web page.

```
<Applet
Code=Hellojava.class
width=400
Height=200>
</Applet>
```

The applet tag discussed above specified the three things:

- 1) Name of the applet
- 2) Width of the applet (in pixels)
- 3) Height of the applet (in pixels)

This HTML code tells the browser to load the compiled java applet Hellojava.class, which is in the same directory as this HTML file. It also specifies the display area for the applet output as 400 pixels width and 200 pixels height. We can make this display area appear in the center of screen by using the CENTER tags as showsn below:

```
<CENTER>
<Applet>
-----
-----
-----
</Applet>
</CENTER>
```

13.4 ADDING APPLET TO A HTML FILE

To execute an applet in a web browser, you need to write a short HTML text file that contains the appropriate APPLET tag. Here is the HTML file that executes **SimpleApplet**:

```
<Applet code = Hellojava.class width = 400 Height = 200 >
</Applet>
```

The width and height attributes specify the dimensions of the display area used by the applet. After you create this file, you can execute the HTML file called RunApp.html (say) on the command line.

```
c:\>appletviewer RunApp.html
```

13.5 RUNNING THE APPLET

To execute an applet with an applet viewer, you may also execute the HTML file in which it is enclosed, eg.

```
c:\>appletviewer RunApp.html
```

Execute the applet the applet viewer, specifying the name of your applet's source file. The applet viewer will encounter the applet tage within the comment and execute your applet.

Example :

The Following will be saved with named FirstJavaApplet.java:

```
import java.awt.*;
import java.applet.*;
public class FirstJavaApplet extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("My First Java Applet Program",100,100);
    }
}
```



```
}
```

After creating FirstJavaApplet.java file now you need create FirstJavaApplet.html file as shown below:

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>My First Java Applet Program</TITLE>
```

```
</HEAD>
```

```
<BODY>
```

```
<APPLET Code="FirstJavaApplet.class" Width=300 Height=200>
```

```
</APPLET>
```

```
</BODY>
```

```
</HTML>
```

OUTPUT:



THE GRAPHICS CLASS

14.1 Lines and Rectangles

14.2 Circles and Ellipses

14.3 Drawing Arcs

14.4 Drawing Polygons

14.5 Line Graphs

14.1 LINES AND RECTANGLES

LINES:

In order to draw a line, you need to use the `drawLine` method of the `Graphics` class. This method takes four parameters, the starting x and y coordinates and the ending x and y coordinates.

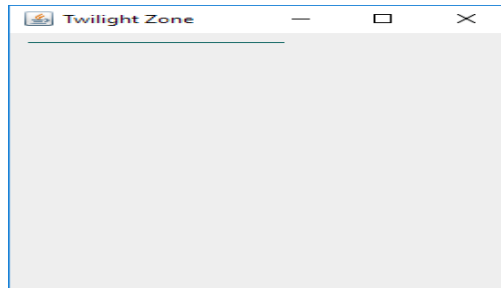
Let's create a `paint` method that we will be adding to. We've already created the main method that runs the code, so we can simply add our shapes as we go.

Example:

```
public void paint(Graphics g)
{ //custom color
    String hexColor = new String("0x45e5B");
    g.setColor(Color.decode(hexColor));
```

```
//draw a line (starting x,y; ending x,y)
g.drawLine(10, 10, 40, 10);
}
```

Output

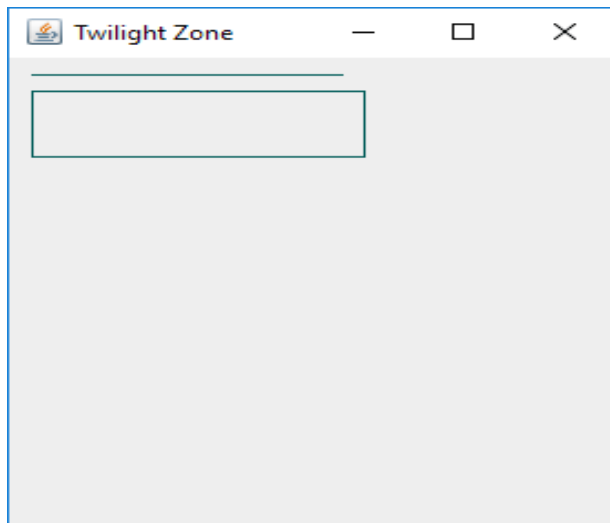


RECTANGLES:

For our line, we used the Graphics class. We can make use of the newer **Graphics2D** class, which allows some more options when it comes to 2-D shapes, including thickness, anti-aliasing, etc. Add code to the beginning of the paint method to create a Graphics 2D instance that casts the Graphics class to Graphics2D:

```
//draw rectangle
g2.drawRect(10, 20, 150, 40);
g2.setColor(Color.decode(hexColor));
```

Output



14.2 CIRCLES AND ELLIPSES

The Graphics class does not have any method for circles or ellipses. However, the drawOval() method can be used to draw a circle or an ellipse.

Like rectangle methods, the drawOval() method draws outline of an oval, and the fillOval() method draws a solid oval.

Example:

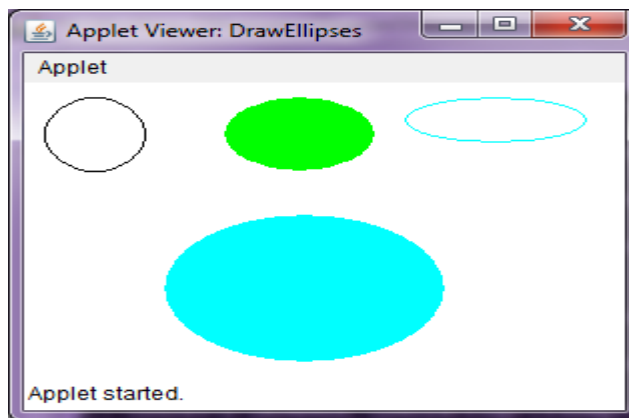
```
import java.awt.*;  
import java.applet.*;  
public class DrawEllipses extends Applet  
{
```

```

public void paint(Graphics g)
{
    g.drawOval(10, 10, 50, 50);
    g.setColor(Color.GREEN);
    g.fillOval(100, 10, 75, 50);
    g.setColor(Color.cyan);
    g.drawOval(190, 10, 90,30);
    g.fillOval(70, 90, 140, 100);
}
}

```

OUTPUT



14.3 DRAWING ARCS

A segment of an oval is an arc. The arc angle can be positive (sweeps anti-clockwise) or negative (sweeps clockwise). As with other figures, the arcs can be outline or solid.

Supporting methods from java.awt.Graphics class

1. void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle): draws an outline arc.
2. void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle): draws a filled (solid) arc.

where

- x and y: Indicates the x and y coordinates where the arc is to be drawn
- width and height: Indicates the width and height of the arc
- startAngle: Dictates the starting angle of the arc
- arcAngle: The angular extent of the arc (relative to the start angle)

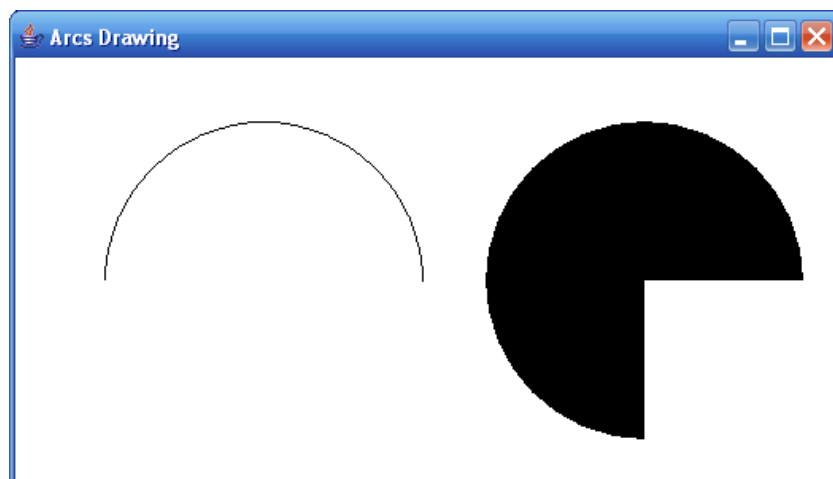
Example:

```
import java.awt.*;

public class ArcsDrawing extends Frame
{
public ArcsDrawing()
{
setTitle("Arcs Drawing");
setSize(525, 300);
setVisible(true);
}

public void paint(Graphics g)
{
g.drawArc(60, 70, 200, 200, 0, 180);
g.fillArc(300, 70, 200, 200, 0, 270);
}

public static void main(String args[])
{
new ArcsDrawing();}}
```

Output:

14.4 DRAWING POLYGONS

Polygon is a closed figure with finite set of line segments joining one vertex to the other. The polygon comprises of set of (x, y) coordinate pairs where each pair is the vertex of the polygon. The side of the polygon is the line drawn between two successive coordinate pairs and a line segment is drawn from the first pair to the last pair.

We can draw Polygon in java applet by three ways :

`drawPolygon(int[] x, int[] y, int numberofpoints)` : draws a polygon with the given set of x and y points.

Example:

```
import java.awt.*;
import javax.swing.*;

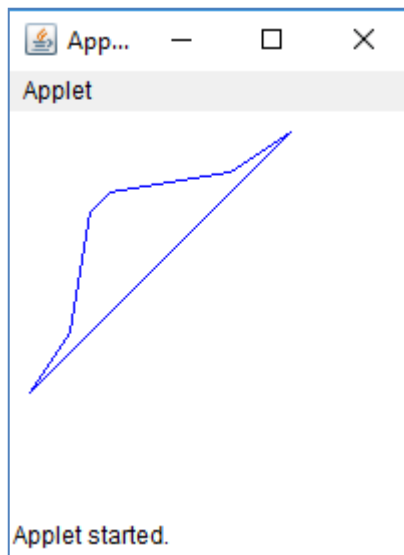
public class poly extends JApplet
{ // called when applet is started
public void init()
{ // set the size of applet to 300, 300
setSize(200, 200);
show();
} // invoked when applet is started
public void start()
{
} // invoked when applet is closed
public void stop()
{
}

public void paint(Graphics g)
{ // x coordinates of vertices
int x[] = { 10, 30, 40, 50, 110, 140 }; // y coordinates of vertices
int y[] = { 140, 110, 50, 40, 30, 10 }; // number of vertices
int numberofpoints = 6; // set the color of line drawn to blue
```

```

g.setColor(Color.blue);           // draw the polygon using drawPolygon function
g.drawPolygon(x, y, numberofpoints);
}
}

```

Output:**14.5 LINE GRAPHS**

We can design applets to draw line graphs to illustrate graphically the relationship between two variables

```

import org.jfree.chart.ChartPanel;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.JFreeChart;
import org.jfree.ui.ApplicationFrame;
import org.jfree.ui.RefineryUtilities;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;

public class LineChart_AWT extends ApplicationFrame {

```



```

public LineChart_AWT( String applicationTitle , String chartTitle ) {
    super(applicationTitle);
    JFreeChart lineChart = ChartFactory.createLineChart(
        chartTitle,
        "Years","Number of Schools",
        createDataset(),
        PlotOrientation.VERTICAL,
        true,true,false);

    ChartPanel chartPanel = new ChartPanel( lineChart );
    chartPanel.setPreferredSize( new java.awt.Dimension( 560 , 367 ) );
    setContentPane( chartPanel );
}

private DefaultCategoryDataset createDataset( ) {
    DefaultCategoryDataset dataset = new DefaultCategoryDataset( );
    dataset.addValue( 15 , "schools" , "1970" );
    dataset.addValue( 30 , "schools" , "1980" );
    dataset.addValue( 60 , "schools" , "1990" );
    dataset.addValue( 120 , "schools" , "2000" );
    dataset.addValue( 240 , "schools" , "2010" );
    dataset.addValue( 300 , "schools" , "2014" );
    return dataset;
}

public static void main( String[ ] args ) {
    LineChart_AWT chart = new LineChart_AWT(
        "School Vs Years" ,
        "Nuner of Schools vs years");

    chart.pack( );
    RefineryUtilities.centerFrameOnScreen( chart );
    chart.setVisible( true );
}
}

```

Output

