Basics of Java – A brief history of Java

- Java is a general-purpose, object-oriented programming language developed by Sun Microsystems of USA in 1991.
- It was called Oak by James Gosling, one of the inventors of the language.
- It was designed for the development of software for consumer electronic devices like TVs, VCRs, toaster etc.
- The team was known as Green Project team by Sun.
- In 1994 the team developed a web browser called "HotJava" to locate and run applet programs on Internet.
- In 1995 Oak was renamed as Java.
- In 1996 Sun released Java Development Kit 1.0.
- Many java versions have been released till now. The current stable release of Java is Java SE 12

JAVA

One of the most popular programming language since 1996.

Has around 10 million developers worldwide.

According to Oracle 3 billion devices run Java.

Considered for small devices to enterprise applications.

Java is everywhere. In mobiles, client machine, server machines, embedded devices, smart phones, cloud,gaming consoles, **Blu-ray players**, **car navigation systems**, medical monitoring devices, parking meters etc.

It is a free and open source software.

It is cross-platform compatible.

It is a object oriented programming language.

"Write once Run anywhere"

Java Platforms / Editions

There are 4 platforms or editions of Java:

1) Java SE (Java Standard Edition) - It is a Java programming platform.

2) Java EE (Java Enterprise Edition) - It is an enterprise platform which is

mainly used to develop web and

enterprise applications.

3) Java ME (Java Micro Edition) - It is a micro platform which is mainly used to develop mobile applications.

4) JavaFX - It is used to develop rich internet applications.

Java Architecture

Java is the first programming language that is not tied to any particular hardware or operating system. Programs developed in Java can be executed anywhere on any system.

Java Features

A list of most important features of Java language is given below.

1. Simple - Java is very easy to learn, and its syntax is simple, clean and easy to understand.
2. Object-Oriented – Almost everything in Java is an object. All program code and data reside within objects and classes. It comes with an extensive set of classes, arranged in packages.
3. Portable - Java compiler generates bytecode instructions that can be implemented on any machine.
4. Platform independent - Java code can be run on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere(WORA).
5. Secured – Java systems not only verify all memory access but also ensure that no viruses are communicated with an applet. The absence of pointers in Java ensures that programs cannot gain access to memory locations without proper authorization.

6. Robust - Java is robust because: It uses strong memory management. There is a lack of pointers that avoids security problems. There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore. There are exception handling and the type checking mechanism in Java. All these points make Java robust.

7. High Performance - Java code is compiled into bytecode which is highly optimized by the Java compiler, so that the Java virtual machine (JVM) can execute Java applications at full speed. In addition, compute-intensive code can be re-written in native code and interfaced with Java platform via *Java Native Interface* (JNI) thus improve the performance.

8. Multithreaded – Multithreaded means handling multiple tasks simultaneously. Java supports multithreaded programs. For ex. one can download an applet from a distant computer while listening to an audio clip.
9. Distributed - Java is distributed because it facilitates users to create distributed applications in Java. It has the ability to share both data and programs. Java applications can open and access remote objects on internet.
10. Dynamic – It is capable of dynamically linking in new class libraries, methods and objects. It also supports functions from its native languages, i.e., C and C++.

Importance of Java to the Internet

In a network, two categories of objects are transmitted between the server and the personal computer

Passive information and Dynamic active programs.

Applet: An Applet is an application designed to be transmitted over the Internet and executed by a Java compatible web browser. When you use a Java-compatible Web browser, you can safely download Java applets without fear of viral infection or malicious intent. Java achieves this protection by Confining a Java program to the Java execution environment and not allowing it access to other parts of the computer

Security - When you use a Java-compatible Web browser, you can safely download Java applets without fear of viral infection

Portability – The output of java compiler bytecode which can be run on any platform.

Java Applets and Applications

Java can be used to create two types of programs: applications and applets. An application is a program that runs on your computer, under the operating system of that computer. When creating such programs, Java is more or less like any other programming language, C and C++.

An applet is an application designed to be transmitted over the Internet and executed by a java-compatible Web browser. An applet is actually a tiny java program, dynamically downloaded across the network, just like an image, sound file, or video clip. The important difference is that an applet is an intelligent program, not just and animation or media file. In other words, and applet is a program that can react to user input and dynamically change not just run the same animation or sound over and over.

Fundamentals of Object Oriented Programming

Object - Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc.An Object can be defined as an instance of a class.

The state of an object(properties or attributes) is represented by data fields with their current values. Ex. A circle object has a data field radius. Behavior of an object is defined by methods. Ex. getArea() and getPerimeter() are methods for circle object.

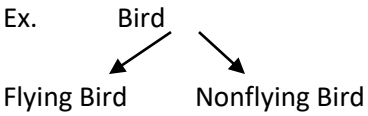
Class – is a template, blueprint that defines what an objects data fields and methods will be. It is a collection of objects of similar type.

Abstraction – refers to the act of representing essential features without including the background details or explanations.

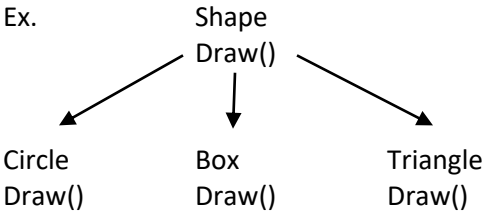
Ex. An object car seem one object from the outside but from the inside it comprises of many subsystems such as steering, brakes, sound systems etc.

Encapsulation – is a mechanism that binds together the data and methods into a single unit. The data is not accessible to the outside world and only those methods, which are wrapped in the class, can access it.

Inheritance – is the process in which one object acquires all the properties and behaviors of a parent object. Inheritance is Java is to create new classes that are built upon existing classes.



Polymorphism – is a concept by which one can perform a single action in different ways. An operation may exhibit different behavior in different instances.



Benefits of OOP

- Inheritance enables us to eliminate redundant code and extend the use of existing classes.
- Instead of starting the code from scratch it can be started from the already existing modules.
- Its feature of data hiding provides for building secure programs.
- More details of the model can be captured because of the approach, which is data centric.
- Programming based on object oriented model makes up gradation easier.
- It is easy to partition the work in a project based on objects.

- Software complexity can be easily managed.

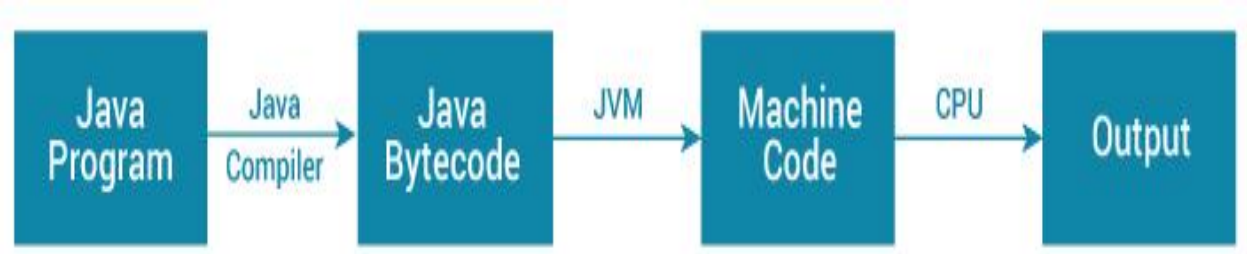
Java and C++

The features of C++ not taken up in Java or significantly modified are as listed:

- Operator Overloading
- Multiple Inheritance accomplished by feature called “interface”.
- Global Variables
- Pointers
- Destructor function is replaced by the finalize() function.
- Header files.

JVM - JVM (Java Virtual Machine) is an abstract machine that enables your computer to run a Java program.

When you run the Java program, Java compiler first compiles your Java code to bytecode. Then, the JVM translates bytecode into native machine code (set of instructions that a computer's CPU executes directly).

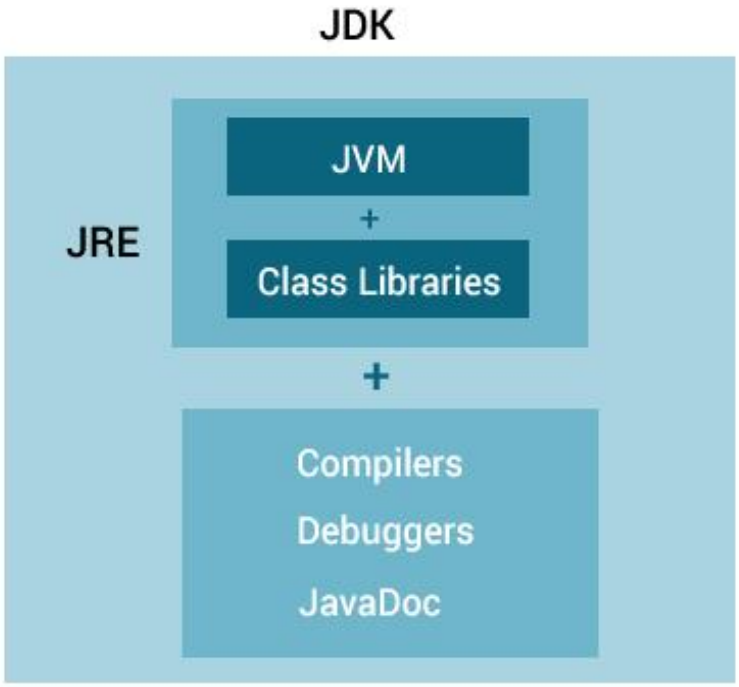


JRE (Java Runtime Environment) is a software package that provides Java class libraries, Java Virtual Machine (JVM), and other components that are required to run Java applications.

JRE is the superset of JVM.

JDK (Java Development Kit) is a software development kit required to develop applications in Java. When you download JDK, JRE is also downloaded with it.

In addition to JRE, JDK also contains a number of development tools (compilers, JavaDoc, Java Debugger, etc).



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

appletviewer (for viewing Java applets)
javac (Java Compiler)
java(Java Interpreter)
javap(java disassemble)
javah(for c header files)
javadoc(for creating HTML documents)
jdb(java debugger)

Application Programming Interface(API)

The Java Standard Library (or API) includes hundreds of classes and methods grouped into several functional packages. The most commonly used ones are:

Language Support Package: A collection of classes and methods required for implementing basic features of Java.

Utilities 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 manipulation.

Networking Package: A collection of classes for communicating with other computers via Internet.

AWT Package: the abstract window Tool Kit package contains classes that implements platform independent graphical user interface.

Applet Package: This includes a set of classes that allows us to create Java applets.

Getting Started with JDK

```
class HelloWorld
{
 public static void main(String args[])
System.out.println("Hello, World!");
}
```

Text editors like Notepad, Edit, Wordpad can be used to type the program. The name of the program should be saved with the name of the class used in the program. It is case sensitive.

HelloWorld.java

C:\>javac HelloWorld.java //compile

The javac compiler creates a file called HelloWorld.class that contains bytecode version of the program.

C:\>java HelloWorld //to run

Output: Hello World!

Using Java with other tools- Following are some of Java-related Internet Technologies

Native Code – It refers to code that is native to a specific processor. Java can call native code quite easily. Access to native code determines access to millions of lines of code and uses it for stand alone, platform specific applications.

JavaScript – It is a separate programming language related to Java in some specific ways. It can be coded directly in an HTML document and gives the programmer more control over the browser.

Netscape Plug-Ins – Netscape communications has created a standard interface to its navigator browser products. The Plug-Ins are written in native code and are hence platform specific.

ActiveX – uses control based on the Component Object Model(COM). They are native code modules, and Microsoft support ActiveX on other platforms and for Windows. Microsoft has designed a VM that allows ActiveX and Java to communicate automatically.

JDBC – Java Database Connectivity is an API for linking Java programs to databases. It is similar to Microsoft's ODBC.(Open Database Connectivity)

JavaBeans is a portable, platform-independent model written in Java Programming Language. Its components are referred to as beans.

.

Chapter 6 : Exception Handling

Errors

Syntax errors arise because the rules of the language have not been followed. They are detected by the compiler.

Ex. missing ; missing ( ) {

Runtime errors occur when program is running. Run time errors are not detected by the java compiler. It is the JVM which detects it while the program is running.

Logic errors : These errors are due to the mistakes made by the programmer. It will not be detected by a compiler nor by the JVM. Errors may be due to wrong idea or concept used by a programmer while coding.

The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.

In Java, an exception is an unwanted or unexpected event, which occurs during the execution of a program ie., at run time, that disrupts the normal flow of the program's instructions.

The core advantage of exception handling is to maintain the normal flow of the application.

Ex.

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5;  //exception occurs
statement 6;
statement 7;
statement 8;
```

Suppose there are 8 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e., statement 6 to 8 will not be executed. If exception handling is performed, the rest of the statement will be executed.

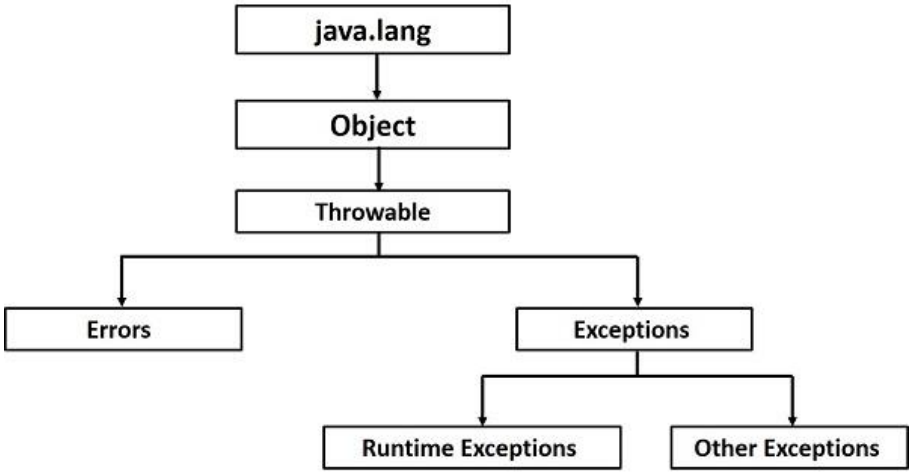
An exception can occur for many different reasons. Following are some scenarios where an exception occurs

- A user has entered an invalid data.
- A file that needs to be opened cannot be found.

- A network connection has been lost in the middle of communications or the JVM has run out of memory.

Java exception handling is managed by using five keywords: **try, catch, throw, throws and finally.**

Java Hierarchy



All exception classes are subtypes of the java.lang.Exception class.

The exception class is a subclass of the Throwable class.

Other than the exception class there is another subclass called Error which is derived from the Throwable class.

Errors are typically ignored in the code because one can rarely do anything about an error.

Example :

(1) JVM is out of Memory. Normally programs cannot recover from errors.

(2) If a stack overflow occurs then an error will arise. They are also ignored at the

time of compilation.

The Exception class has two main subclasses:

(1) Compiletime or IOException or Checked Exceptions class

(2) RuntimeException or Unchecked Exception class

(1)Compiletime or Checked Exceptions : These exceptions can be detected by java compiler while compiling. They are explicitly handled in the code itself with the help of try-catch blocks. These exceptions cannot be ignored.

(2) RuntimeException or Unchecked Exception - are those detected during runtime by JVM. These exceptions are not essentially handled in the program code. These include programming bugs, such as logic errors or improper use of an API. They are ignored at the time of compilation.

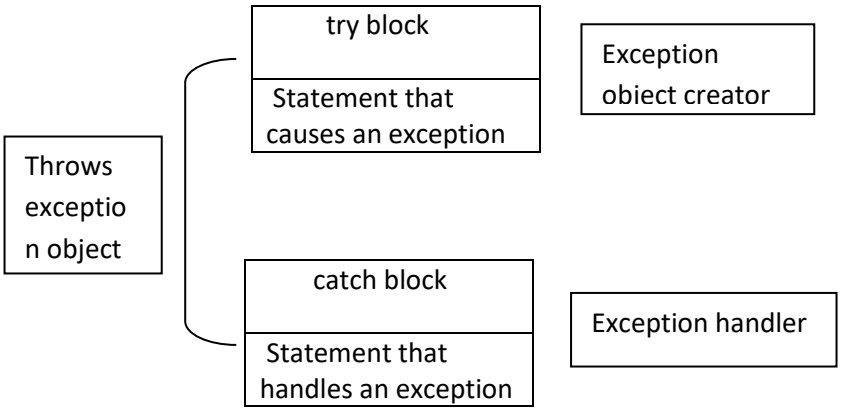
Some of Checked Exceptions – ClassNotFoundException, CloneNotSupportedExcept, NoSuchFieldException, NoSuchMethodException

Some of Unchecked Exceptions – ArithmeticException, ArrayStoreException, ArrayIndexOutofBoundsException, FileNotFound Exception

Exception handling

The basic concepts of exception handling are throwing an exception and catching it.

Hit the Exception
Throw the exception
Catch the exception
Handle the exception



Syntax

```
try
{
  Statement;
}
catch(Exceptiontype object)
{
  Statement;
}
```

The code which is prone to exceptions is placed in the try block. When an exception occurs, that exception occurred is handled by catch block associated with it. Every try block should be immediately followed either by a catch block or finally block.

A catch statement involves declaring the type of exception that is trying to catch. The exception is passed to the catch block much as an argument is passed into a method parameter.

Example :

```
class Excep1
{
 public static void main(String ar[])
{
 int a=20, b=10,c=10;
 int x,y;
try
{
  x=a/(b-c);
  System.out.println("X=  "+x);
  }
catch(ArithmeticException e)
{
  System.out.println("Division by zero");
}
y=a/(b+c);
System.out.println("Y= " +y);
}
}
```

Multiple Catch Statements

It is possible to have more than one catch statement in the catch block. When an exception in a try block is generated, the java treats the multiple catch statements like cases in a switch statement.

```
. . . . . .
. . . . . .
try
{
statement;
}
catch(Exception-Type-1 e)
{
statement;
}
catch(Exception-Type-2 e)
{
statement;
}
catch(Exception-Type-n e)
{
statement;
```

```
}
. . . . . .
. . . . . .
```

If an exception occurs, the exception is thrown to the first catch block in the list, if the datatype of the exception thrown matches ExceptionType-1 it gets caught there. Otherwise it is thrown to second catch block and so on.

Ex.

```
class Excep2
{
public static void main(String ar[])
{
  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);
}
}
```

Nested try statements

There could be situations where there is a possibility of generation of multiple exceptions of different types within a particular block of the program code. Try statements are used in such situation.

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

```
 System.out.println("try1");
 try
 {
  int a[] = new int[5];
  a[6] = 200;
 }
 catch(ArrayIndexOutOfBoundsException e1)
 {
  System.out.println("index out of bound");
  }
 int r = 200/0;
 }
 catch(ArithmeticException e2)
 {
  System.out.println("divide by zero");
  }
}
}
```

OUTPUT
try1
index out of bound
divide by zero

Methods available to Exceptions

All errors and exceptions are subclasses of class Throwable and thus can access the methods defined in it. The following are the most commonly used ones:

1. String getMessage() - gets the detail message
2. String toString() – returns a short description of the Throwable, including the detail message if there is one.
3. void printStackTrace()
   void printStackTrace(PrintStream) – prints the Throwable and the Throwable's call stack trace.

Ex.

```
public class ExMethods()
{
 public static void main(String ar[])
 {
  try
  {
   throw new Exception("new Exception");
  }
  catch(Exception e)
 {
   System.out.println("Caught Exception");
```

```
   System.out.println("e.getMessage " +e.getMessage());
   System.out.println("e.toString " +e.toString());
   System.out.println("e.printStackTrace()");
   e.printStackTrace();
  }
 }
}
```

Finally Statement

Java supports another statement known as finally statement that can be used to handle an exception that is not caught by any of the previous catch statements.We can put finally block after the try block or after the last catch block. The finally block is executed in all circumstances. Even if a try block completes without problems, the finally block executes.

```
try
{
 . . . . . .
 . . . . . .
}
finally
{
. . . . . .
 . . . . . .
}
```

```
try
{
  . . . . . .
  . . . . . .
}
catch()
{
  . . . . . .
  . . . . . .
}
catch()
{
  . . . . . .
  . . . . . .
}
finally
{
. . . . . .
. . . . . .
}
```

- A catch clause cannot exist without a try statement.
- It is not compulsory to have finally clauses whenever a try/catch block is present.
- The try block cannot be present without either catch clause or finally clause.
- Any code cannot be present in between the try, catch, finally blocks.

Example

```
public class FinallyDemo
{
public static void main(String args[])
 {
   int num1 = 100; int num2 = 50; int num3 = 50; int result1;
  try
  {
    result1 = num1/(num2-num3);
    System.out.println("Result1 = " + result1);
   }
  catch(ArithmeticException g)
 {
   System.out.println("Division by zero");
   }
  finally
   {
    System.out.println("This is final");
   }
  }
  }
```

OUTPUT
Division by zero
This is final

Throwing your own Exception

Throw Statement – If we want to throw our own exceptions, can be done by using the throw statement as follows. The throw keyword is used to explicitly throw an exception from a method or any block of code. We can throw either checked or unchecked exception.
The flow of execution of program stop immediately after the throw statement is executed and the nearest enclosing try block is checked to see if it has a catch statement that matches the type of execution.

throw new exception_class(“error message”);

ex. throw new ArithmeticException(“dividing a number by 5 is not allowed in this program”);

```
 public class ExThrow
{
 static void Eligibilty(int age, int weight)
{
 if(age<12 && weight<40) {
 throw new ArithmeticException("Student is not eligible for registration");
 }
```

```
 else
 {
  System.out.println("Student Entry is Valid!!");
 }
}

 public static void main(String args[])
 {
    System.out.println("Welcome to the Registration process!!");
    Eligibilty(10, 39);
    System.out.println("Good luck");
 }
}
```

Throws Keyword is used for handling checked exceptions . By using throws we can declare multiple exceptions in one go.
The throws does the same thing that try-catch does but there are some cases where you would prefer throws over try-catch. If there are several methods that can cause exceptions, it is tedious to write try-catch for each method.

To overcome this problem is by using throws like this: declare the exceptions in the method signature using throws and handle the exceptions where you are calling this method by using try-catch.

Advantage of using this approach is that you will be forced to handle the exception when you call this method, all the exceptions that are declared using throws, must be handled where you are calling this method else you will get compilation error.

In this **example** the method Check() is throwing two checked exceptions so we have declared these exceptions in the method signature using throws Keyword. If we do not declare these exceptions then the program will throw a compilation error. Because IOException is a Checked Exception, which should be either handled or declared to be thrown.

```
import java.io.*;
class ThrowEx {
  void Check(int num)throws IOException, ClassNotFoundException{
    if(num==1)
      throw new IOException("IOException Occurred");
    else
      throw new ClassNotFoundException("ClassNotFoundException");
  }
}

public class Example1{
  public static void main(String args[]){
   try{
```

```
    ThrowEx obj=new ThrowEx();
    obj.Check(1);
   }catch(Exception ex){
    System.out.println(ex);
    }
  }
}
```

Output:
java.io.IOException: IOException Occurred

Difference between throw and throws keyword

No.	throw	throws
1)	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2)	Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
3)	Throw is followed by an instance.	Throws is followed by class.
4)	Throw is used within the method.	Throws is used with the method signature.
5)	You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.

JAVA TOKENS

Java Tokens are the smallest individual building block or smallest unit of a Java program; the Java compiler uses it for constructing expressions and statements. Java program is a collection of different types of tokens, comments, and white spaces.

There are various tokens used in Java:

- Reserved Keywords
- Identifiers
- Literals
- Operators
- Separators

White space is also considered as a token.

Java Character Set – The smallest unit of the Java language and is used to write the Java tokens. These characters are defined by the Unicode character set, an emerging standard that tries to create characters for a large number of scripts worldwide. It is a 16-bit character coding system and supports more than 34,000 defined characters derived from 24 languages.

Keywords – They are also known as reserved words. There are 60 words as keywords. As Java is a case sensitive language all the keywords are written using lower case letters.

Identifiers – They are used for naming classes, methods, variables, objects, labels, packages and interfaces. An identifier
1. can have alphabets, digits, underscore and dollar sign characters.
2. must not begin with a digit.
3. Uppercase and Lowercase letters are distinct.
4. They can be of any length.

Literals – or Constants are sequence of character (digits, letters and other characters) that represent constant value to be stored in a variable.

Different types of literals are
- Integer
- Floating Point
- Boolean
- Strings
- Characters

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

Separators – are symbols used to indicate where groups of code are divided and arranged.

Symbol	Name	Description
( )	Parentheses	Used to contain the lists of parameters in method definition and invocation. Also used for defining the precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to contains the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates the statements
,	Comma	Separates consecutive identifiers in a variable declarations. Also used to chain statements together inside a **for** statement
.	Period	Used to separate packages names from subpackages and classes. Also used to separate a variable or method from a reference variable.
::	Colons	Used to create a method or constructor reference (Added by JDK 8)

Constants – in Java refer to fixed values that do not change during the execution of a program.

Different types of constants are

1. Integer – is a sequence of digits. There are three types of integers
a. Decimal – set of digits, 0 through 9, preceded by an optional minus sign. It does not allow embedded spaces, commas and non-digit characters between digits.
Ex. 765 -439 0 87654 Invalid - 98 143 34.280 $234
b. Octal – integer constant consists of any combination of digits from the set 0 to 7 and it should begin with a 0.
Ex. 034 0 0564
c. Hexadecimal – integer constants consists of any combination of digits from 0 to 9 and is also has a combination of alphabets from 'A' to 'F' and it should begin with a 0x or 0X
Ex. 0X2 0X9F 0xbcd 0x

2. Floating Point – numbers are expressed as decimal numbers with an optional decimal point. A whole number followed by a decimal point and fractional part, which is an integer. It can also be expressed in exponential notation.
Ex. 0.0065 -0.35 .95 -.8 0.65e4 -1.2e-1

3. Character – A character constant contains a single character enclosed within a pair of single quote marks.
   Ex. '5' 'P' ';' ' '

4. String – It is a sequence of characters enclosed between double quotes. The characters may be alphabets, digits, special characters and blank spaces.
   Ex. "Hello Java" "2010" "4+7" "?...!"

Java Variables – A variable is an identifier that denotes a storage location used to store a data value. It takes different values at different times during the execution of the program. An variable may consist of alphabets, digits, the underscore and dollar sign subject to the following conditions

- They must not begin with a digit.
- Uppercase and lowercase are distinct.
- It should not be a keyword.
- White space is not allowed.
- Variable names can be of any length.

Scope of variables - There are three types of variables in java:

- Instance variable
- Class variable
- Local variable

Instance Variable – They are declared inside the class variable. They are created when the objects are instantiated and therefore they are associated with the objects. They take different values for each object.

Class Variable – They are declared inside the class. They are global to a class and belong to the entire set of objects that class creates. Only one memory location is created for each class variable.

Local Variable - A variable declared inside the body of the method is called local variable. They are not available for use outside the method definition. They can also be declared inside program blocks that are defined between an opening brace and a closing brace.

Data types - specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. Primitive data types: The primitive data types include boolean, char, byte, short, int, long, float and double.
2. Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays.

Java Primitive Data Types are grouped into following four categories:

- Integers – It includes byte,short,int,long
- Floating Point numbers – It includes float and double
- Character
- Boolean

Integers

a.Byte Data Type

The smallest integer type. It is an 8-bit signed two's complement integer range from -128 to 127 (inclusive). Its default value is 0.

Example: byte a,b;

b.Short Data Type

This is a 16-bit signed two's complement integer range from -32,768 to 32,767 (inclusive). Its default value is 0.

Example: short a,b;

c.Int Data Type

This is a 32-bit signed two's complement integer range from - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its default value is 0. The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example: int a,b;

d.Long Data Type

This is a 64-bit two's complement integer range from -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

Example: long a,b;

Floating Point numbers
Float - The float type uses 4 bytes of storage. They are useful when you require fractional part but do not require higher degree of precision.

Example**:** float root, average;

Double - It uses 8 bytes of storage.

Example: double root, average;

Char Data Type

Data type used to store characters is char. It is a single 16-bit Unicode character. It is 2 bytes long and has a range from 0 to 65,536 and there are no negative character.

Example**:** char a,b;

Boolean Data Type

The Boolean data type is used to store only two possible values: true and false.

Example: Boolean flag;
flag=true;

Type Casting

Whenever it is required to store a value of one type to another type. Then it is required to type cast the value to be stored by proceeding it with the type name in parenthesis.
The syntax is: type variable =(type) variable 2;

This process of converting one data type to another data type is called casting. Four integer types can be cast to any other type except Boolean.

Operators in java

Operator in java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in java which are given below:

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

(1) Arithmetic Operators

Arithmetic operators are used to construct mathematical expressions. These operators cannot be used on Boolean type.

+ Addition or unary plus, - Subtraction or unary minus, * Multiplication, / Division, % Modulo division

Integer Arithmetic – When both the operators in an expression are integer type then the operation is called integer arithmetic.
Ex. When a=14, b=4 then a-b =10, a+b=18, a*b = 56, a/b=3, a%b = 2

Real Arithmetic – When both the operands in an expression are real type then the operation is called real arithmetic.

Ex. When a=10.3, b=6.4 then a+b=16.7, a-b=3.9, a*b=65.920006, a/b=1.609375, a%b=3.9

Mixed Mode Arithmetic – When one of the operands is real and other is integer. If either of the operands is real type then the result will be real type.

Ex. 20/2.5=8.0

(2) Relational Operators

Whenever it is required to compare the two quantities depending upon their relation and take certain decision then these comparisons are done with the help of relational operators.

Operators	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

(3) Logical Operators

They are used when it is required to combine the result of two or more relational expression.

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

(4) Assignment Operators

They are used to assign a value of an expression to a variable. Java has a set of 'shorthand' assignment operators which are used in the form

v op=exp; where v is a variable, exp is an expression, op is a Java binary operator

a+=1, a-=1, a*=n+1, a/=n+1, a%=b

(5) Increment and Decrement Operators

++ and --

++m --m (prefix) m++ m--(postfix)

(6) Conditional Operators

The character '?' and ':' are ternary operators. This operator is used to construct conditional expressions in the form

exp1 ? exp2 : exp3

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

(7) Bitwise Operators

These operators are used to manipulate data at bit level and they may not be applied to float or double type.

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Ones Compliment
<<	Shift left
>>	Shift right
>>>	Shift right with zero fill

When >> operator is used the vacated bits on the left are filled with the original sign bit, if >>> operator is used they are filled with 0. Ex. 10111010>>4=11111011 10111010>>>4 = 00001011

(8) Special Operators

Java supports some special operators such as:

Instanceof : It is an object reference operator and returns true if the object on left hand side is an instance of class given on right hand side else it return false. It determines if the object belongs to that class or not.

Ex. person instanceof student - is true if person belongs to class student else false

Dot operator(.) : It is used to access the instances variables and methods of objects and also for accessing classes and sub packages from a package.

Ex. person.age reference to variable age
person.salary() reference to method salary

Precedence of Operators – Order of evaluation of the operator in a single expression which has more than one operator.

Control Flow Statements

If and If else statements

if statement is used to test the condition. It checks boolean condition: true or false.

```
if(Boolean expression)
{
Statements;
}
```

if-else statement also tests the condition. It executes the if block if condition is true otherwise else block is executed.

```
if(Boolean expression)
{
  Statement_1;
}
else
{
 Statement_2;
}
```

The nested if statement represents the if block within another if block. Here, the inner if block condition executes only when outer if block condition is true.

```
if(Boolean expression1)
{
 Statement_1;
}
else if(Boolean expression2)
{
 Statement_2;
}
else if(Boolean expression3)
{
```

```
  Statement_3;
}
```

Switch Statement

Switch statement executes one statement from multiple conditions. It is used instead of multiple if-else-if statement. Switch statement works on a single integer or character constant value only.

```
switch(choice)
{
  case 1: statment1;
  break;
  case 2: statement2;
  break;
  case 3: statement3;
  break;
  default:  statement;
}
```

For loop

It is a control flow statement that iterates a part of the programs multiple times. If the number of iteration is fixed, it is recommended to use for loop.

```
for(initialization; test condition; increment)
{
   statement;
}
```

```
for(i=1;i<=10;i++)
{
   System.out.println(i);
}
```

While loop

It is a control flow statement that executes a part of the programs. If the number of iteration is not fixed, it is recommended to use while loop.

```
while(condition)
{
  Statement;
}
```

```
int i=1;
while(i<=10)
{
```

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

do while loop

It is a control flow statement that executes a part of the programs at least once and the further execution depends upon the given boolean condition. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop.

```
do
{
 Statement;
} while(condition);
```

```
int i=1;
do
{
  System.out.println(i);
  I++;
}while(i<=10);
```

Break Statement

When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.

break is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop. Break statement is used in all types of loops such as for loop, while loop and do-while loop.

Continue Statement

The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately. It can be used with for loop or while loop. continue statement is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition. In case of an inner loop, it continues the inner loop only.

comments

There are 3 types of comments in java.

1. Single Line Comment - //This is single line comment
2. Multi Line Comment - /* */
3. Documentation Comment - /** */

Packages and Interfaces

Defining an interface

An interface in java is a blueprint of a class. It has static constants and abstract methods.

An interface can have methods and variables, but the methods declared in interface are by default abstract (only method signature, no body).

- The members of an interface are always declared as constant, ie., their values are final.
- The methods in an interface are abstract in nature.
- It cannot be used to declare objects. It can only be inherited by a class.
- It can only use the public access specifier.
- At the time of declaration, interface variable must be initialized.
- An interface does not contain any constructors.
- An interface cannot extend classes.

```
interface <interface_name>
{
      // Constant declaration;
// method declaration
}
```

Ex.

```
 interface test
{
int a=10;
void show();
}
```

Extending Interfaces - An interface can extend another interface in the same way that a class can extend another class. The extends keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

Interface name2 extends name1

```
interface Item1
{
 int code=101;
 String n=”Ram”;
}
interface Item2
{
 void display();
}
interface item extends item1, item2
{

}
```

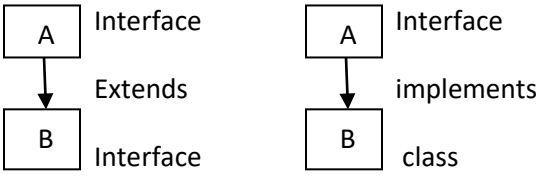
Implementing Interfaces

A class declares the implementation through an implements clause in the class declaration and then create the methods defined by the interface.

```
class classname implements interface1,interface2, interface3
{
  body;
}
```

Ex.

```
interface i1
{
 int a=10;
void display();
}
class c1 implements i1
```

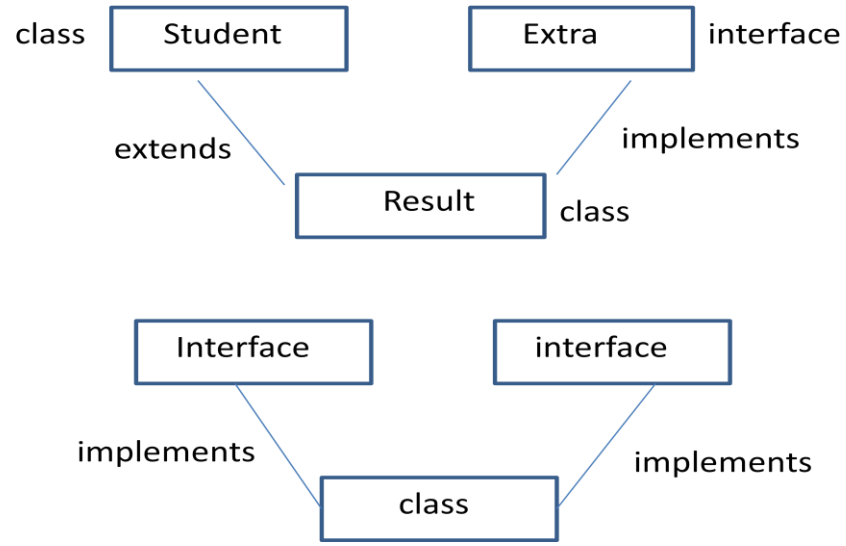


Example:

```
interface A
{
 int num=500;
 void dispa();}
interface B
{
 void dispb();}
class C implements A,B
{
 public void dispa()
 {
  System.out.println("Number "+num);
 }
 public void dispb()
 {
  System.out.println("Method from B interface");
 }
}
class Intmain
{
 public static void main(String ar[])
```

```
 {
  C obj=new C();
  obj.dispa();
  obj.dispb();
 }
}
```

Multiple Inheritance in Interface




```
class Student
{
 int m1,m2;
 void inputm(int x,int y)
 {
  m1=x;
  m2=y;
 }
 void showm()
 {
  System.out.println("First"+m1);
  System.out.println("Second"+m2);}}
interface Extra
{
 int E=10;
 void emarks();
}
class Result extends Student implements Extra
{
 int total;
 public void emarks()
```

```
 {
  System.out.println("Extracurricular marks"+E);
 }
 void disp()
 {
  showm();
  emarks();
  total=m1+m2+E;
  System.out.println("Total"+total);
 }
}
class Imain
{
 public static void main(String ar[])
{
 Result obj=new Result();
 obj.inputm(50,60);
 obj.disp();
}
}
```

Packages

A java package is a group of similar types of classes, interfaces and sub-packages. The grouping is done according to functionality. They act as containers for classes.

Benefits of package

- The classes contained in the packages of other programs can be easily reused.
- Two classes in two different packages can have a same name.
- They always provide access protection.

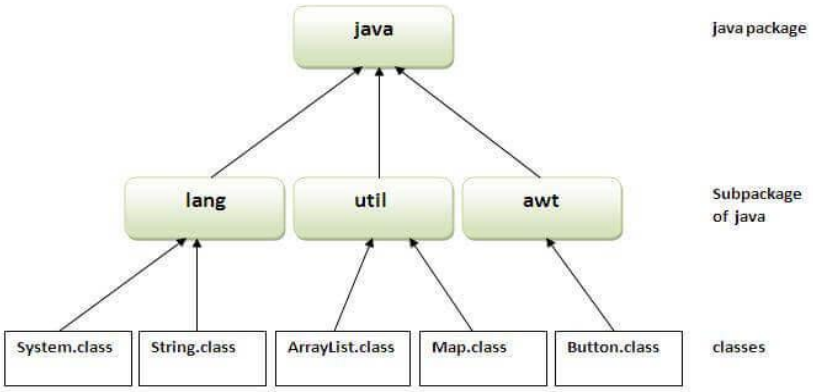
Defining a package

package packagename;

ex. package testpack;

Types of packages

(1) Built-in packages - are library of prewritten classes. The library is divided into packages and classes. There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.



(2) User defined packages These are the packages that are defined by the user.

Naming conventions - Packages can be named using the standard java naming rules.

Packages begin with lowercase letters, which makes it different from class names.

Creating packages – It involves the following steps

1. Declare the package at the beginning of a file using the form
   package packagename;
2. Define the class that is to be put in the package and declare it public.
3. Create a subdirectory under the directory where the main source files are stored.
4. Store the listing as the classname.java file in the subdirectory created.
5. Compile the file. This creates .class file in the subdirectory.

Ex. package firstpack;
public class FirstClass
{
. . . . . .
}

In the example the file is saved as FirstClass.java and is located in a directory named firstpack. When the source file is compiled, it will create a .class file and store it in the same directory.

Accessing a package

Java package can be accessed by either using a fully qualified class name or using import statement.

Fully qualified name – If it is used only declared class of this package will be accessible. It is generally used when two packages have same class name eg. Java.util and java.sql packages contain Date class.

The general form of import statement is

import package1 [.package1] [.package2].classname;

ex. import java.util.Scanner;

Another approach is
import packagename.*;
ex. Import java.util.*;

CLASSPATH

The setting of this variable is used to provide the root of any package hierarchy to java compiler. CLASSPATH is an environment variable needed for the Java compiler and runtime to locate Java packages used in a Java program.

You need to set the CLASSPATH if you need to load a class that is not present in the current directory or any sub-directories.

Ex. SET CLASSPATH=.;C:\;

Creating and Using packages(importing a package)

```
package package1;
public class ClassA
{
  public void dispA()
 {
  System.out.println("Class A");
}
}
```

The source file is saved as ClassA.java and is stored in subdirectory package1. The compiled file ClassA.class will be stored in the same subdirectory.

```
import package1.ClassA;
class PTest1
{
  public static void main(String ar[])
 {
  ClassA obj=new ClassA();
  obj.dispA();
}
}
```

The file is saved as PTest1.java and then compiled. The source file and compiled file are stored in the directory of which package1 was a subdirectory.

Subclassing an imported class

```
package package2;
public class Class1
{
  protected int val=50;
  public void dispA()
 {
  System.out.println(“Class A”);
  System.out.println(“Value=”+val);
}
}
import package1.Class1;
class Class2 extends Class1
{
int n=20;
 void dispB()
{
System.out.println(“Class 2”);
System.out.println(“Value=”+val);
 System.out.println(“n=” +n);
}
}
class PTest2
{
 public static void main(String ar[])
{
Class2 obj=new Class2();
  obj.dispA();
  obj.dispB();
}
}
```

Example of package when name of classname is same in two packages

```
package package_one;
public class ClassOne
{
 public void methodOne()
 {
  System.out.println("Hello i am Class one");
 }
}

package package_two;
public class ClassOne
{
 public void methodTwo()
```

```
 {
  System.out.println("Hello i am Class two");
 }
}

import package_one.ClassOne;
public class PackTest
{
 public static void main(String[] args)
 {
  ClassOne a = new ClassOne();
  a.methodOne();
  package_two.ClassOne b=new
  package_two.ClassOne();
  b.methodTwo();
 }
}
```

Access Protection

The following type of visibility for class members are defined

Subclasses in the same package
Non-subclasses in the same package
Subclasses in different packages
Classes that are neither in the same package not subclasses

public - can be accesses from anywhere.
private – members cannot be seen outside of their class
protected –allow the direct subclass to have access to the members.
no-modifier(default) – members are visible to other classes and subclasses in the same package.

Java API Package

All the core java classes are stored inside some named packages and those packages are stored in a package called java. These packages are referred to as Java API packages. Java API provides a large number of classes grouped into different packages according to functionality. These packages use namespace partitioning, which means that every class contained in a package has a unique name that cannot conflict with class names defined elsewhere.

The commonly used packages are

Package java.lang– Contains the main language support classes.
Package java.util– Contains language support classes such as vectors, hash tables, random nos etc.
Package java.io– Contains Input/Output support classes.
Package java.awt– Contain classes for implementing the components for graphical user interfaces
Package java.net– Contain classes for supporting networking operations.
Package java.applet – Contains classes for creating Applets.

Working with Graphics

The Graphics class includes methods for drawing many different types of shapes, from simple lines to polygons to text in a variety of fonts. Java.awt.Graphics is an abstract class that defines a number of graphical operations. Its 47 public methods can be used for rendering images and text, drawing and filling shapes, clipping graphical operations, and much more.

Uses of class java.awt.Graphics

java.awt – contains all the classes for creating user interfaces and for painting graphics and images.

java.awt.image – provides classes for creating and modifying images.

java.awt.print – provides classes and interfaces for a general printing API.

java.beans, java.awt.image, javax.swing etc.

Uses of Graphics in java.awt

Methods in java.awt- abstractGrahics,

Component.getGraphics()

Graphics.create()

Graphics.create(int x, int y, int width, int height)

Image.getGraphics()

PrintJob.getGraphics()

Custom Painting

Drawing Lines – Lines are drawn by means of the drawLine() method

public abstract void drawLine(int x1, int y1, int x2, int y2)

drawLine() draws a line, using the current color, between the points(x1,y1) and (x2,y2) in this graphics context's coordinate system.

Drawing Rectanges – The drawRect() method is used to display an outlined rectangle. This method takes four arguments, the coordinates of the upper-left corner of the rectangle plus the width and the height of the rectangle.

public void drawRect(int x, int y, int width, int height)

Ex. g.drawRect(10,20,40,50);

To draw a solid box use fillRect() method.

public abstract void fillRect(int x, int y, int width, int height)

Drawing Ellipses and Circles

To draw an Oval, drawOval() method is used and for filling an oval the fillOval() method is used.

public abstract void drawOval(int x, int y, int width, int height)

public abstract void fillOval(int x, int y, int width, int height)

Drawing Arcs

public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)

public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)

Drawing Polygons

public abstract void drawPolygon(int xPoints[], int yPoints[], int nPoints)

public abstract void fillPolygon(int xPoints[], int yPoints[], int nPoints)

Ex 1

```
import java.awt.*;
import java.applet.*;
//<applet code=Gra1 width=500 height=500></applet>
public class Gra1 extends Applet
{
 public void init()
 {
  setBackground(Color.black);
  setForeground(Color.white);
 }
 public void paint(Graphics g)
 {
  int x[]={200,250,280,290,200,150};
  int y[]={100,200,300,400,500,450};
  g.setColor(Color.red);
  g.setFont(new Font("Arial",1,20));
  g.drawString("Welcome to CE",40,50);
  g.setColor(Color.green);
  g.drawRect(40,70,100,40);
  g.fillRect(40,120,100,40);
  g.drawOval(40,170,30,60);
  g.fillOval(90,170,30,60);
  g.drawLine(40,240,140,240);
  g.drawArc(40,260,60,100,60,60);
  g.drawOval(40,280,30,30);
```

```
  g.drawPolygon(x,y,6);
 }
  }
```

Ex. 2

```
import java.awt.*;
import java.applet.*;
public class Smile extends Applet
{
   public void paint(Graphics g)
   {
      setBackground(Color.BLACK);
      g.setColor(Color.YELLOW);
      g.fillOval(30,50,100, 100);
      g.setColor(Color.BLACK);
      g.fillOval(55,75,10, 10);
      g.fillOval(95,75,10, 10);
      g.drawArc(60, 85, 40,40,-30,-120);
   }
 }
//<applet code="Smile.class" width="200" height="200"> </applet>
```

Applet Programming

An applet is a Java program that can be embedded into a web page. It runs inside the web browser and works at client side. An applet is embedded in an HTML page using the APPLET or OBJECT tag and hosted on a web server. It can perform arithmetic operations, display graphics, play sounds, accept user input, create animation, and play interactive games. They are used to make the web site more dynamic and entertaining.

Applets are used to make the web site more dynamic and entertaining.

1. All applets are sub-classes (either directly or indirectly) of java.applet.Applet class.
2. Applets are not stand-alone programs. Instead, they run within either a web browser or an applet viewer. JDK provides a standard applet viewer tool called applet viewer.
3. In general, execution of an applet does not begin at main() method.
4. Output of an applet window is not performed by System.out.println(). Rather it is handled with various AWT methods, such as drawString().

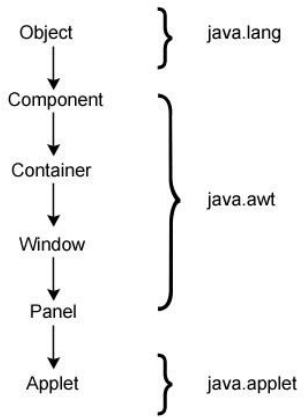


Figure: Applet Hierarchy

Advantages of Applet

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many platforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

- Plugin is required at client browser to execute applet.

Life Cycle of an Applet

The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

- The init() method
- The start() method
- The stop() method
- The destroy() method

The init() method – to initialize all the variables and objects in applet. It is called only once.

The start() method – is called immediately after the init() method. It is used to start the Applet.

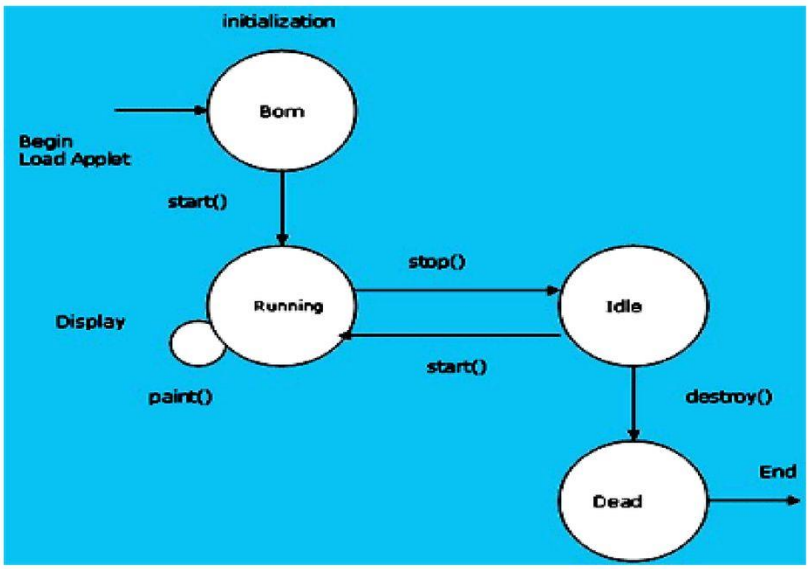
The stop() method - is called when a web browser leaves the HTML document containing the applet—when it goes to another page, for example. When stop( ) is called, the applet is probably running.

The destroy() method – is called when the environment determines that the applet needs to be removed completely from memory. This method can be used to perform clean-up operations like closing a file or free any other resources that the applet might be using.

paint() method - public void paint(Graphics g): is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

```
public void paint (Graphics g)
{
g.drawstring(“first applet”,30,20);
}
```

Life Cycle of an Applet



Commonly used methods used

1. public abstract void drawString(String str, int x, int y): is used to draw the specified string.
2. public void drawRect(int x, int y, int width, int height): draws a rectangle with the specified width and height.
3. public abstract void fillRect(int x, int y, int width, int height): is used to fill rectangle with the default color and specified width and height.
4. public abstract void drawOval(int x, int y, int width, int height): is used to draw oval with the specified width and height.
5. public abstract void fillOval(int x, int y, int width, int height): is used to fill oval with the default color and specified width and height.
6. public abstract void drawLine(int x1, int y1, int x2, int y2): is used to draw line between the points(x1, y1) and (x2, y2).
7. public abstract booleandrawImage(Image img, int x, int y, ImageObserver observer): is used draw the specified image.
8. public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle): is used draw a circular or elliptical arc.
9. public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle): is used to fill a circular or elliptical arc.
10. public abstract void setColor(Color c): is used to set the graphics current color to the specified color.

11. public abstract void setFont(Font font): is used to set the graphics current font to the specified font.

The Applet Tag – To run an applet, add the applet to an HTML page using the <APPLET> tag. The most commonly used attributes are CODE, HEIGHT, WIDTH, CODEBASE, ALT. Parameters can be passed to an applet using the PARAM tag.

Applet can be run using

(a) appletviewer
(b) web browser

The general format is

```
import java.awt.*;
import java.applet.*;
. . . . . .
. . . . . .
public class appletclassname extends Applet
{
 . . . . . .
public void paint(Graphics g)
{
 . . . . . .
. . . . . .
}
. . . . . .
}
```

(a) Running the program using appletviewer

```
import java.applet.*;
import java.awt.*;
public class AppFirst extends Applet
{
 public void paint(Graphics g)
{
  g.drawString("Welcome to AIT",100,100);
}
}
```

Execution – save it as AppFirst.java

```
c\>javac AppFirst.java
c\>appletviewer AppFirst.java
```

(b) Running the program using web browser

1. Write Applet Code
2. Save -> App2.java
3. Compile -> App2.class
4. Design a webpage using HTML tags.
5. Save -> App2.html
6. Open the App2.html in web browser(Internet Explorer)

Ex.

```
import java.applet.Applet;
import java.awt.Graphics;
public class App2 extends Applet
{
 public void paint(Graphics g)
 {
  g.drawString("Hello Java",50,50);
 }
}
```

Save file App2.java
Compile javac App2.java

```
<HTML>
<HEAD>
<TITLE>APPLET EXAMPLE</TITLE>
</HEAD>
<BODY BGCOLOR=BLUE>
<APPLET code="App2.class" width="200" height="200">
</APPLET>
</BODY>
</HTML>
```

Save the file as App2.html. Open in the web browser

Restrictions and Limitations

1. Applets do not use the main() method for initiating the execution of the code. When loaded, they automatically call certain methods of Applet class to start and execute the applet code.
2. They cannot be run independently. They are run from inside a web page using a special feature known as HTML tag.
3. They cannot read from or write to the files in the local computer.
4. They cannot communicate with other servers on the network.

5. They cannot run any program from the local computer.
6. They are restricted from using libraries from other languages such as C or C++.

Taking Advantage of the Applet API

applets can do the following:

- Be notified by the browser of milestones.
- Load data files specified relative to the URL of the applet or the page in which it is running.
- Display short status strings.
- Make the browser display a document.
- Find other applets running in the same page.
- Play sounds.
- Get parameters specified by the user in the <APPLET> tag.

Finding and Loading Data Files

Whenever an applet needs to load some data from a file that's specified with a relative URL (a URL that doesn't completely specify the file's location), the applet usually uses either the code base or the document base to form the complete URL

The code base, returned by the `Applet getCodeBase` method, is a URL that specifies the directory from which the applet's classes were loaded.

The document base, returned by the `Applet getDocumentBase` method, specifies the directory of the HTML page that contains the applet.

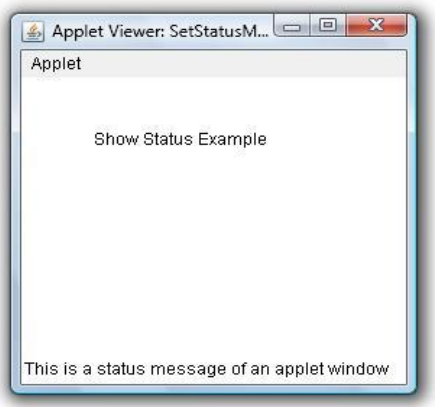
Unless the <APPLET> tag specifies a code base, both the code base and document base refer to the same directory on the same server.

Displaying Short Status Strings

An applet can also output small messages to the status window of the browser or applet viewer on which it is running. This string appears on the status line at the bottom of the applet viewer window.

showStatus() method is used for status window display.

```
public void paint(Graphics g)
{
  g.drawString(“Show Status Example” 50,50);
  showStatus(“This is a status message of an applet window”);
}
```



Displaying Documents in Browser

A Java applet can load a web page in a browser window using the `showDocument` methods in the `java.applet.AppletContext` class.

The two forms of showDocument:

public void showDocument(URL) – displays the document at the specified URL.
public void showDocument(URL, String targetWindow) - displays the specified document at the specified location within the browser window

The two-argument form of `showDocument` lets you specify the window or HTML frame in which to display the document. The second argument can have one of the folllowing values:

`"_blank"` – Display the document in a new, nameless window.

`"windowName"` – Displays the document in a window named windowName. This window is created if necessary.

`"_self"` – Display the document in the window and frame that contain the applet.

`"_parent"` – Display the document in parent frame of the applet's frame. If the applet frame has no parent frame, this acts the same as `"_self"`.

`"_top"` – Display the document in the top-level frame. If the applet's frame is the top-level frame, this acts the same as `"_self"`.

Playing Sounds

In the Java Applet package the Applet class and AudioClip interface provide basic support for playing sounds.

The following methods return an object that implements the AudioClip interface

getAudioClip(URL), getAudioClip(URL, String)

The following methods play the AudioClip corresponding to the specified URL.

play(URL), play(URL, String)

The AudioClip interface defines the following methods

loop() – Starts playing the clip repeatedly

play() – plays the clip once.

stop() – stops the clip. Works with both looping and one-time sounds.

Defining and Using Applet Parameters

Java allows users to pass user-defined parameters to an applet with the help of <PARAM>tags. The <PARAM>tag has a NAME attribute which defines the name of the parameter and a VALUE attribute which specifies the value of the parameter. In the applet source code, the applet can refer to the parameter by its NAME to find its value. The syntax of the <PARAM>tag is:

```
<APPLET>
 <PARAM NAME=parameter1_name VALUE=parameter1_value>
 <PARAM NAME=parameter2_name VALUE=parameter2_value>
 <PARAM NAME=parametern_name VALUE=parametern_value>
 </APPLET>
```

Ex.

```
<APPLET CODE="AppletParameterTest.class" WIDTH="400" HEIGHT="50">
<PARAM NAME="font" VALUE="Dialog">
<PARAM NAME="size" VALUE="24">
<PARAM NAME="string" VALUE="Hello, world ... it's me. :)">
 </APPLET>
```

4. Arrays Strings and Vectors

An array is a group of contiguous or related data items that have a common name.

- In Java all arrays are dynamically allocated.
- Since arrays are objects in Java, we can find their length using member length.
- A Java array variable can also be declared like other variables with [ ] after the data type.
- The variables in the array are ordered and each have an index beginning from 0.
- Java array can be also be used as a static field, a local variable or a method parameter.
- The size of an array must be specified by an int value and not long or short.

One-dimensional array (Single) :

The general form of a one-dimensional array declaration is

type var-name[ ] or type[ ] var-name

Ex. int sum[];
int[] sum;

The element type for the array determines what type of data the array will hold.

```
int arr[];
or int[] arr;
byte arr[];
short arr[];
boolean arr[];
long arr[];
float arr[];
double arr[];
char arr[];
```

Instantiating an Array in Java

When an array is declared, only a reference of array is created. To actually create or give memory to array, it is created using new.

var-name = new type [size];

```
int arr[];                  //declaring array
arr = new int[20];          // allocating memory to array
OR
int[] arr = new int[20];  // combining both statements in one
```

First declare a variable of the desired array type. Second allocate the memory that will hold the array, using new, and assign it to the array variable. Thus, in Java all arrays are dynamically allocated.

Array Literal

In a situation, where the size of the array and variables of array are already known, array literals can be used.

```
int[] arr = new int[]{ 1,2,3,4,5,6,7,8,9,10 };      // Declaring array literal
```

To get the size of an array, write **arrayname.length**

Ex. int n = sum.length;

Multidimensional Arrays

They are arrays of arrays with each element of the array holding the reference of other array.

```
int[][] arr = new int[10][20];              //a 2D array or matrix

int[][][] arr = new int[10][20][10];       //a 3D array
```

Arrays can be passed to method and a method can also return array.

Strings

Strings represent a sequence of characters. The strings are implemented as objects of type String. (java.lang). Java strings are immutable. Once they are created their values cannot be changed. All string variables are instances of the string class.

(a) By string literal
```
String str=”Java”;
System.out.println(str);
```

(b) By new keyword
```
String str;
str=new String(“Java”);
System.out.println(str);
Str=new String();
```

String Class

Strings are class objects and implemented using two classes, String class and String Buffer class. String objects are read only. String Buffer class is used to manipulate strings.

Length of string

```
String str=”Java”;
System.out.println(str.length());
```

Example:

```
class Test
{
  public static void main(String ar[])
  {
   String str1=”computer”;
   String str2=new String(“JAVA”);
   char name[]={‘C’,’O’,’M’,’P’,’U’,’T’,’E’,’R’};
   String str3=new String(name);
   String str4=new String(name,3,5);
   String str5=new String(str4);
   System.out.println(“\n\t 1st String “ +str1);
   System.out.println(“\n\t 2nd String “ +str2);
   System.out.println(“\n\t  3rd String “ +str3);
   System.out.println(“\n\t  4th String “ +str4);
   System.out.println(“\n\t 1st String “ +str5);
   System.out.println(“Length of first string”,+str1.length());
}
}
```

StringBuffer class – is a thread-safe, mutable sequence of characters.

- A string buffer is like a string, but can be modified.
- It contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls.
- Every string buffer has a capacity.

String Constructors

1. StringBuffer() - It reserves room for 16 characters without reallocation.
   StringBuffer str=new StringBuffer();

2.StringBuffer(int capacity/size) - It accepts an integer argument that explicitly sets the size of the buffer.
  StringBuffer str=new StringBuffer(50);

3.StringBuffer(String str) It accepts a **String** argument that sets the initial contents of the StringBuffer object and reserves room for 16 more characters without reallocation.
   StringBuffer str=new StringBuffer(“Java”);

Methods

1. length( ) and capacity( ): The length of a StringBuffer can be found by the length( ) method, while the total allocated capacity can be found by the capacity( ) method.

2. append(): It is used to add text at the end of the existence text.

3. insert( )**:** It is used to insert text at the specified index position.

Example:

```
class StrBufclass
{
 public static void main(String ar[])
 {
  String fn,ln;
  StringBuffer s1=new StringBuffer();
  System.out.println("\n\t S1 length :"+s1.length());
  System.out.println("\n\t S1 Capacity :"+s1.capacity());
  StringBuffer s2=new StringBuffer(10);
  System.out.println("\n\t S2 length :"+s2.length());
  System.out.println("\n\t S2 Capacity :"+s2.capacity());
  StringBuffer s3=new StringBuffer("JAVA");
  System.out.println("\n\t S3 :"+s3);
  System.out.println("\n\t S3 length :"+s3.length());
  System.out.println("\n\t S3 Capacity :"+s3.capacity());
  StringBuffer s = new StringBuffer("\nComputer");
       s.append("Dept");
  System.out.println(s);
  StringBuffer sn = new StringBuffer("ComputerDept");
       sn.insert(8, "Engg");
  System.out.println(sn);
 }
}
```

String Concatenation

(1)By +(Concatenation) operator

Ex. String str1=”Computer” + “Department”;
String str2=str1+”AIT”;
String str3=str1+str2;

Output -> str3=ComputerDepartmentAIT

Ex. int n=10;
String str=”Java”;
String str1=n+str;
Output -> str1=10Java

String str1=str+n+2;
Output -> Java102

(2) By String Concat() method

String str1=”Computer”;
String str2=”Dept”;

String str3=str1.Concat(str2);

Str3=ComputerDept

Example:

```
import java.util.*;
class Sconcat
{
 public static void main(String ar[])
 {
  int n=50;
  String s1="Welcome"+"to";
  String s2=s1+"AIT";
  String s3=s1+s2;
  String s4=n+10+s1;
  String s5=s1+n+10;
  String s6=s1.concat("CS Dept");
  String s7="1styear".concat(s6);
  System.out.println("String1" +s1);
  System.out.println("String2" +s2);
  System.out.println("String3" +s3);
  System.out.println("String4" +s4);
  System.out.println("String5" +s5);
  System.out.println("String6" +s6);
  System.out.println("String7" +s7);
 }
}
```

Vector and Wrapper Class

The Vector class implements a resizable array of objects. They implement a dynamic array that means it can grow or shrink as required. Items can be added to the beginning, middle or at the end of a vector and any element in the vector can be accessed with an array index.

There are three kinds of methods in vector

Methods to modify Vectors
Methods to get value out of Vectors
Methods that manage how the vector grows when it needs more capacity.

Constructor:

Vector() : Creates a default vector of initial capacity 10.
Vector(int initialCapacity) : It constructs an empty vector with the specified initial capacity
Vector(int initialCapacity, int capacityIncrement) : It constructs an empty vector with the specified initial capacity and capacity increment.

Vector(Collection c) : Creates a vector that contains the elements of collection c.

Increment of vector capacity: If increment is specified, Vector will expand according to it in each allocation cycle but if increment is not specified then vector's capacity get doubled in each allocation cycle.
Vector defines three protected data member:

int capacityIncrement : Contains the increment value.
int elementCount : Number of elements currently in Vector stored in it.
Object elementData[] : Array that holds the vector is stored in it.

Simple data types cannot be directly stored in vector, they can only store objects.

Working with Vector Methods – Following are some of the vector methods

1. addElement( ) : It inserts the element at the end of the Vector.
2. add() : It is used to append the specified element in the given vector.
3. capacity() : This method returns the current capacity of the vector.
4. remove() : It is used to remove the specified element from the vector.
5. removeElementAt() : It is used to delete the component at the specified index.
6. insertElementAt() : It is used to insert the specified object as a component in the given vector at the specified index.
7. clear() : It is used to delete all of the elements from this vector.
8. size( ) : It is used to get the number of components in the given vector.
9. setSize(int size) : It changes the existing size with the specified size.
10. elementAt(int index) : It returns the element present at the specified location in Vector.

Example

```
import java.util.*;
class VMain{
   public static void main(String[] args)
  {
      //Create vectors v1, v2,v3 and v4
      Vector<Integer> v1 = new Vector<>();  //a vector with default constructor
      Vector<Integer> v2 = new Vector<>(20);  // a vector of given Size
      //initialize vector v2 with values
      v2.add(10);
      v2.add(20);
      v2.add(30);
      Vector<Integer> v3 = new Vector<>(5,10);  //Size and Increment
      v3.add(1);
      v3.add(2);
      v3.add(3);
      v3.add(4);
      v3.add(5);
      v3.add(6);
```

```
      System.out.println("Vector v1 Contents:" +v1);
      System.out.println("Vector v2 Contents:" +v2);
      System.out.println("Vector v3 Contents:" +v3);
      System.out.println("Vector v3 capacity:" +v3.capacity());
      System.out.println("Vector v3 size:" +v3.size());
    }
}
```

Ex.

```
import java.util.*;
class Vector1
{
 public static void main(String ar[])
 {
  Vector<Integer> vobj=new Vector<>();
  vobj.add(5);
  vobj.add(7);
  vobj.add(9);
  vobj.add(2);
  vobj.add(1);
  System.out.println("Vector is " +vobj);
  System.out.println("current capacity of Vector object is "+vobj.capacity());
  vobj.addElement(4);
  System.out.println("Vector is " +vobj);
  System.out.println("Remove element at index 4 "+vobj.remove(4));
  System.out.println("Vector after removal" +vobj);
  vobj.removeElementAt(3);
  System.out.println("Vector after removal" +vobj);
  vobj.insertElementAt(12, 1);
  System.out.println("Values in Vector object are :" +vobj);
 }
}
```

Wrapper Classes - provide a way to **use** primitive data types (int, char, short, byte, etc) as objects. To convert primitive data types into object and vice-versa. Classes of java.long package are known as wrapper class in java.

Autoboxing – Automatic conversion of primitive to object. Or The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing.

Unboxing – object to primitive. Or The automatic conversion of wrapper type into its corresponding primitive type is known as unboxing.

Primitive type	Wrapper class
boolean	Boolean
byte	Byte
short	Short
int	Integer

long	Long
float	Float
double	Double
char	Character

Primitive to wrapper

```
import java.util.*;
class wrapper
 {
 public static void main(String ar[])
 {
 int i=100;  //I primitive
 Integer iobj = Integer.valueOf(i); // j object
 Integer j=i;      // autoboxing
 System.out.println(i + " " +iobj+ " " +j);
 }
}
```

Wrapper to primitive

```
class wrapper2
{
 public static void main(String ar[])
{
 Integer iobj1 = new Integer(100);
int i=iobj1.intValue();
int j=iobj1;       //unboxing
System.out.println(i+j);
}
}
```

Chapter 3

Objects and Classes in Java

Class - A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. Class is not a real world entity. It does not occupy memory. In classname first letter of every word should be capital letter.

```
class ClassName
{

}
```

Data is encapsulated in a class by placing data fields inside the body of the class definition. These variables are called **instance variables**(also known as member variables) because they are created whenever an object of the class is instantiated. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created.

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

A class in Java can contain:

- Fields
- Methods
- Constructors
- Blocks
- Nested class and interface

Method – A set of codes which perform a particular task. It is equivalent to a function, procedure in other language, but it must be defined within a class.

It consists of two parts: method declaration and method body.
Method declaration contains method name, method parameters and method return data type.The method body is where all the action takes place.

Advantages
code reusability
code optimization

Syntax:
returnType methodName(parameter-list)

```
{
  body of method
}
```

Ex.

```
class Rectangle
{
  int length,width;
  void getData(int x,int y) //method declaration
  {
   length=x;
   width=y;
  }
  int rectArea()  //another method
  {
   int area=length*width;
   return(area);
  }
}
```

Instance variables and methods in classes are accessible by all the methods in the class but a method cannot access the variables declared in other methods.

Object - An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc.

- An object is a real-world entity.
- An object is a runtime entity.
- It occupies memory.
- The object is an instance of a class.

Object consists of (ex. dog)

Identity – ex. name
State/attribute – ex. color, breed, age
Behavior – ex. eat, run, bark
(behavior represents method)

Creating object in java

1. new keyword
2. newInstance() method
3. clone() method

4. deserialization
5. factory method

Objects can be created using new keyword

1. Declaration – It is declaring a variable name with an object type.
Rectangle rect1;
2. Instantiation – This is when memory is allocated for an object.
'new' keyword is used to create an object.
rect1=new Rectangle();

Rectangle rect1=new Rectangle(); // to create object

Two or more references can be created to the same object.

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

Accessing class members

Instance variables and methods cannot be accessed directly, so dot operator is used.
objectname.variablename=value;
objectname.methodname(parameter-list);

Ex.
rect1.length=15;
rect1.width=10;
rect2.length=20;
rect2.width=12;

Assigning values to the instance variables by using method

Rectangle rect1=new Rectangle();
Rect1.getData(15,10);

Example showing classes, methods and objects

```
class Rectangle
{
 int length,width;          //declaration of variables
 void getData(int x, int y)   //definition of method
{
 length=x;
```

```
 width=y;
}
int rectArea()                //definition of another method
{
 int area=length*width;
 return(area);
}
}
class RectArea       //class with main method
{
 public static void main(String args[])
 {
  int area1,area2;
  Rectangle rect1=new Rectangle();  //creating objects
  Rectangle rect2=new Rectangle();
  rect1.length=15;                  //accessing variables
  rect1.width=10;
  area1=rect1.length * rect1.width;
  rect2.getData(20,12);   //accessing methods
  area2=rect2.rectArea();
  System.out.println("Area1=" +area1);
  System.out.println("Area2=" +area2);
 }
}
```

Constructors

It is used to initialize the instance variables. A constructor is a special method that is used to initialize a newly created object and is called just after the memory is allocated for the object.

It has the following characteristics:

- The constructor's name and class name should be same. And the constructor's name should end with a pair of simple braces
- A constructor may or may not have parameters.
- A constructor does not return any value, not even 'void'.
- A constructor is automatically called and executed at the time of creating an object.
- A constructor is called and executed only once per object. This means when we create an object, the constructor is called. When we create second object, again the constructor is called second time.

There are two types of constructors in Java:

1.Default constructor

2.Parameterized constructor

Ex.

```
class Perimeter
{
int l, b;
Perimeter()       //default constructor
{
l=0;
b=0;
}
Perimeter(int x, int y)     //parameterized constructor
{
 l=x;
b=y;
}
void cal_perimeter()
{
 int p;
p=2*(l+b);
System.out.println("The perimeter of the rectangle is"+p);
}
}
class Conperimeter
{
 public static void main(String args[])
{
 perimeter p1=new Perimeter();
perimeter p2=new Perimeter(5,10);
p1.cal_perimeter();
p2.cal_perimeter();
}
}
```

Overloading Methods

Method overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different. Method overloading is an example of Static Polymorphism/compile time polymorphism.

In order to overload a method, the argument lists of the methods must differ in either of these:
1.Number of parameters –
   add(int, int)
   add(int,int,int)
2.Data type of parameters -

add(int,int)
add(int,float)

3. Sequence of data type of parameters -
add(int,float)
add(float,int)

```
class Sum
{
  int sum(int x,int y)
  {
   return(x+y);
  }
 int sum(int x,int y,int z)
{
 return(x+y+z);
}
double sum(double x, double y)
{
 return(x+y);
}
public static void main(String args[])
{
  Sum s=new Sum();
  System.out.println(s.sum(10,20));
  System.out.println(s.sum(10,20,30));
  System.out.println(s.sum(10.5,20.5));
 }
}
```

Constructor Overloading

It allows a class to have more than one constructor having different argument lists.

```
class Room
{
 float length;
 float breadth;
  Room(float x, float y)    //constructor1
  {
  length=x;
  breadth=y;
  }
  Room(float x)             //constructor2
  {
  length=breadth=x;
  }
  int area()
  {
```

```
  return(length+breadth);
  }
}
class Overload
{
  public static void main(String args[])
  {
   Room room1=new Room(25.0,15.0);
   Room room2=new Room(20.0);
   room1.area();
   room2.area();
  }
}
```

Access Control Modifiers – Public, Private and Protected

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.It provides highest degree of protection. They are accessible only with their own class. They cannot be inherited by subclasses and are therefore inaccessible in subclasses.

Default: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

Protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.It makes the fields visible not only to all classes and subclasses in the same package but also to subclasses in other packages

The **protected access modifier** is accessible within package and outside the package but through inheritance only.The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

Public: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.Any variable or method is visible to the entire class in which it is defined. Its visibility and accessible is everywhere.

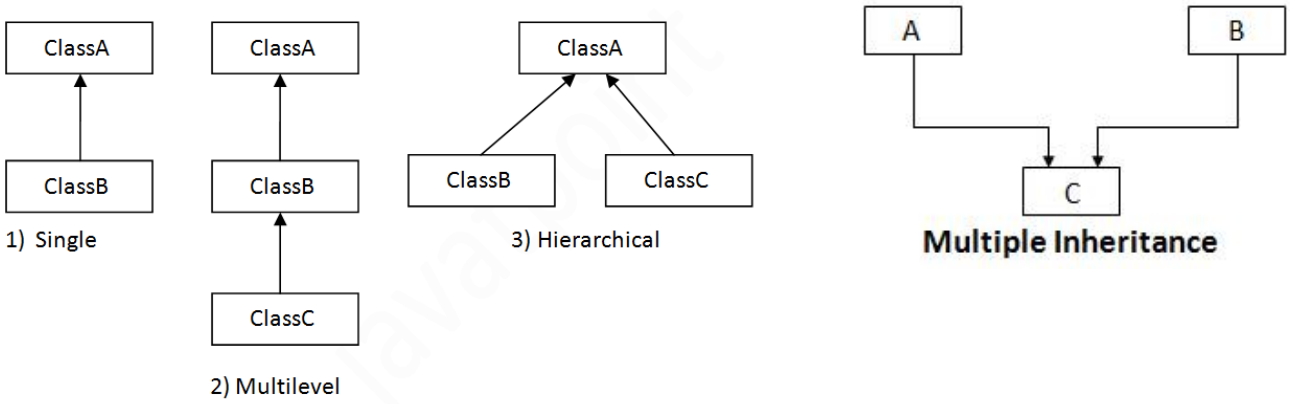
Non-Access Modifiers - do not control access level, but provides other functionality. There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc

Ex. The static **modifier** for creating class methods and variables. ... The abstract **modifier** for creating abstract classes and methods. The synchronized and volatile **modifiers**, which are used for threads.

Inheritance

Deriving new classes from existing classes such that the new classes acquire all the features of existing classes is called inheritance. The old class is known as the base class or super class or parent class and the new class is called subclass or derived class or child class.

1. Single inheritance : when one class inherits only one class(only one super class)
2. Multiple inheritance : when one class inherits from more than one class(several super classes)
3. Hierarchical inheritance : when many classes inherit from the same class.(one super class, many subclasses)
4. Multilevel inheritance : when one class inherits from a already derived class.



Multiple inheritance is not directly implemented and Java uses a secondary interfaces for this purpose. Multiple inheritance is not supported because it leads to deadly diamond problem. A class cannot extend more than one class.

Defining a subclass

```
class subclassname extends superclassname
{
  variable declaration;
  methods declaration;
}
```

The keyword extends signifies that the properties of the superclassname are extended to the subclassname. The subclass will now contain its own variables and methods as well as those of the superclass.

Multilevel Inheritance

When a class extends a class, which extends anther class then this is called multilevel inheritance. For example class C extends class B and class B extends class A then this type of inheritance is known as multilevel inheritance.

```
class A
{
  ……
  ……
}
class B extends A
{
  …….
  …….
}
class C extends B
{
  …….
  …….
}
```

Hierarchical Inheritance

When more than one classes inherit a same class then this is called hierarchical inheritance. For example class B, C and D extends a same class A. when a class has more than one child classes (sub classes) or in other words more than one child classes have the same parent class.

```
 class A
{
  ……
  ……
}
class B extends A
{
  …….
  …….
}
class C extends A
{
  …….
  …….
}
class D extends A
{
  …….
  …….
}
```

super keyword -The super keyword refers to superclass (parent) objects. It is used to call superclass methods, and to access the superclass constructor. The most common use of the super keyword is to eliminate the confusion between superclasses and subclasses that have methods with the same name. In Java every subclass can refer to its immediate superclass by super.

1.Use of super with variables **:** when a derived class and base class has same data members.
2.Use of super with methods : whenever a parent and child class have same named methods.
3.Use of super with constructors : super keyword can also be used to access the parent class constructor.

```
class P     //super class
{
  int num;
  void msg()
  {
    System.out.println("no. in superclass"+num);
  }
}
class C extends P   //subclass
{
   int num;
   C(int a, int b)       //constructor
   {
    Super.num=a;   //use super to access variable
    num=b;
  }
 void msg()
 {
  System.out.println("no. in subclass"+num);
 }
 void disp()
 {
  super.msg();      //use super to access method
  msg();
 }
}
class Test
{
  public static void main(String arg[])
  {
    C obj=new C(10,20);
    super.num=a;
    num=b;
   obj.disp();
  }
}
```

Method Overriding

- Declaring a method in sub class which is already present in parent class is known as method overriding.
- Overriding is done so that a child class can give its own implementation to a method which is already provided by the parent class.
- The method in parent class is called overridden method and the method in child class is called overriding method.
- Method overriding is used for runtime(dynamic) polymorphism
- The main advantage of method overriding is that the class can give its own specific implementation to a inherited method without even modifying the parent class code.

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

```
class Super
{
  int a=10;
  void disp()
  {
System.out.println(“Superclass value”+a);
  }
}
class Sub extends Super
{
  int b=20;
  void disp()
{
System.out.println(“Superclass”+a);
System.out.println(“Subclass”+b);
}
}
class Main
{
  public static void main(string args[])
 {
  Sub obj=new Sub();
obj.disp();
}
}
```

```
Superclass 10
Superclass 20
```

Static, Final and Abstract Modifiers

Static - Static Modifier is used for creating a static variable and static method. In Java, the static modifier means something is directly related to a class: if a field is static, then it belongs to the class; if a method is static, then it belongs to the class. So static variable is called class variable and static method is called class method.

Without creating objects, the static variable and static method can be accessed.
Ex. Counter.count;
Counter.disp();

Static Variable

- They are initialized only once, at the start of the execution.
- They can be accessed directly by the class name and doesn't need any object.
- It is variable whose single copy in memory is shared by all objects, any modification to it will also affect other objects.

Static int num;

Example static variable

```
class Counter
{
 static int count=0;
 Counter()
 {
  count++;
  System.out.println(count);
 }
 public static void main(String arg[])
 {
  Counter c1=new Counter();
  Counter c2=new Counter();
  Counter c3=new Counter();
 }
}
```

Static Method

- They belongs to the class and not its instances.
- They are executed first and then objects are created. So they cannot access instance variable.
- They can access static variables.

Example static method

```
class Test
{
 static int cube(int x)  //class method
```

```
 {
  return(x*x*x);
 }
}
class Main
{
 public static void main(String ar[])
 {
  int y=Test.cube(6);
  //method is called using class
  System.out.println("Cube = "+y);
 }
}
```

Final Modifier – Final modifier can be associated with methods, classes and variables.

final variables are nothing but constants. We cannot change the value of a final variable once it is initialized.

Using final to define constants – To make a local variable, class variable and instance variable a constant declare it final. A final variable may only be assigned to once and its value will not change and can help avoid programming errors.

```
final int COUNT;
final int COUNT=100;
static final int COUNT;
```

final method - the method cannot be overridden by subclasses.

```
final void disp()
{

}
```

final class - A class that is declared final cannot be subclassed(cannot be inherited)

```
 final class Test
  {

  }
```

Abstract modifier

Abstract class - A class which is declared as abstract is known as an **abstract class.**

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.

- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final method.

Abstract method - A method which is declared as abstract and does not have implementation(body) is known as an abstract method. It can be implemented by subclass. If any class have abstract method then the class must be abstract.

```
abstract class XYZ
{
 abstract void disp();
}
```

Example

```
abstract class Super
{
 abstract void disp();
 void display();
 {
  System.out.println("Method from Super class");
 }
}
class Sub extends Super
{
  void disp()  //implementation of disp
  {
   System.out.println("Method defined in Sub class");
  }
}
class Abmain
 {
  public static void main(String ar[])
  {
    Sub obj=new Sub();   //object created using subclass
    obj.disp();
    obj.display();
  }
}
```