CS405PC: JAVA PROGRAMMING

B.TECH II Year II Sem.

L	T	P	C
3	1	0	4

Course Objectives:

- To introduce the object oriented programming concepts.
- To understand object oriented programming concepts, and apply them in solving problems.
- To introduce the principles of inheritance and polymorphism; and demonstrate how they relate to the design of abstract classes
- To introduce the implementation of packages and interfaces
- To introduce the concepts of exception handling and multithreading.
- To introduce the design of Graphical User Interface using applets and swing controls.

Course Outcomes:

- Able to solve real world problems using OOP techniques.
- Able to understand the use of abstract classes.
- Able to solve problems using java collection framework and I/o classes.
- Able to develop multithreaded applications with synchronization.
- Able to develop applets for web applications.
- Able to design GUI based applications

UNIT - I

Object-Oriented Thinking- A way of viewing world – Agents and Communities, messages and methods, Responsibilities, Classes and Instances, Class Hierarchies-Inheritance, Method binding, Overriding and Exceptions, Summary of Object-Oriented concepts. Java buzzwords, An Overview of Java, Data types, Variables and Arrays, operators, expressions, control statements, Introducing classes, Methods and Classes, String handling.

Inheritance– Inheritance concept, Inheritance basics, Member access, Constructors, Creating Multilevel hierarchy, super uses, using final with inheritance, Polymorphism-ad hoc polymorphism, pure polymorphism, method overriding, abstract classes, Object class, forms of inheritance- specialization, specification, construction, extension, limitation, combination, benefits of inheritance, costs of inheritance.

UNIT - II

Packages- Defining a Package, CLASSPATH, Access protection, importing packages.

Interfaces- defining an interface, implementing interfaces, Nested interfaces, applying interfaces, variables in interfaces and extending interfaces.

Stream based I/O (java.io) – The Stream classes-Byte streams and Character streams, Reading console Input and Writing Console Output, File class, Reading and writing Files, Random access file operations, The Console class, Serialization, Enumerations, auto boxing, generics.

UNIT - III

Exception handling - Fundamentals of exception handling, Exception types, Termination or resumptive models, Uncaught exceptions, using try and catch, multiple catch clauses, nested try statements, throw, throws and finally, built- in exceptions, creating own exception sub classes.

Multithreading- Differences between thread-based multitasking and process-based multitasking, Java thread model, creating threads, thread priorities, synchronizing threads, inter thread communication.

UNIT - IV

The Collections Framework (java.util)- Collections overview, Collection Interfaces, The Collection classes- Array List, Linked List, Hash Set, Tree Set, Priority Queue, Array Deque. Accessing a Collection via an Iterator, Using an Iterator, The For-Each alternative, Map Interfaces and Classes, Comparators, Collection algorithms, Arrays, The Legacy Classes and Interfaces- Dictionary, Hashtable ,Properties, Stack, Vector

More Utility classes, String Tokenizer, Bit Set, Date, Calendar, Random, Formatter, Scanner

UNIT - V

GUI Programming with Swing – Introduction, limitations of AWT, MVC architecture, components, containers. Understanding Layout Managers, Flow Layout, Border Layout, Grid Layout, Card Layout, Grid Bag Layout.

Event Handling- The Delegation event model- Events, Event sources, Event Listeners, Event classes, Handling mouse and keyboard events, Adapter classes, Inner classes, Anonymous Inner classes.

A Simple Swing Application, Applets – Applets and HTML, Security Issues, Applets and Applications, passing parameters to applets. Creating a Swing Applet, Painting in Swing, A Paint example, Exploring Swing Controls- JLabel and Image Icon, JText Field, **The Swing Buttons**- JButton, JToggle Button, JCheck Box, JRadio Button, JTabbed Pane, JScroll Pane, JList, JCombo Box, Swing Menus, Dialogs.

TEXT BOOKS:

1. Java The complete reference, 9th edition, Herbert Schildt, McGraw Hill Education (India) Pvt. Ltd.
2. Understanding Object-Oriented Programming with Java, updated edition, T. Budd, Pearson Education.

REFERENCE BOOKS:

1. An Introduction to programming and OO design using Java, J. Nino and F.A. Hosch, John Wiley & sons
2. Introduction to Java programming, Y. Daniel Liang, Pearson Education.
3. Object Oriented Programming through Java, P. Radha Krishna, University Press.
4. Programming in Java, S. Malhotra, S. Chudhary, 2nd edition, Oxford Univ. Press.
5. Java Programming and Object-oriented Application Development, R. A. Johnson, Cengage Learning.

UNIT-I

Object Oriented Thinking

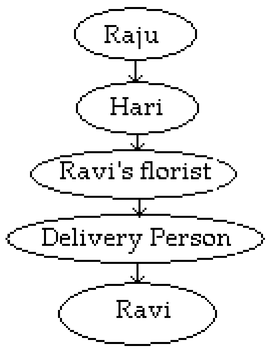
- When computers were first invented, programming was done manually by toggling in a binary machine instructions by use of front panel.
- As programs began to grow, high level languages were introduced that gives the programmer more tools to handle complexity.
- The first widespread high level language is FORTRAN. Which gave birth to structured programming in 1960's. The Main problem with the high level language was they have no specific structure and programs becomes larger, the problem of complexity also increases.
- So C became the popular structured oriented language to solve all the above problems.
- However in SOP, when project reaches certain size its complexity exceeds. So in 1980's a new way of programming was invented and it was OOP. OOP is a programming methodology that helps to organize complex programs through the use of inheritance, encapsulation & polymorphism.

A way of viewing world

- A way of viewing the world is an idea to illustrate the object-oriented programming concept with an example of a real-world situation.
- Eg: Raju wished to send some flowers to his friend for his birthday & he was living some miles away. He went to local florist and said the kind of flowers. He want to send to his friend's address. And florist assured those flowers will be sent automatically.

Agents and Communities

- An object-oriented program is structured as a community of interacting agents, called objects. Where each object provides a service (data and methods) that is used by other members of the community.
- Consider an eg, raju and ravi are good friends who live in two different cities far from each other. If Raju wants to send flowers to ravi, he can request his local florist 'hari' to send flowers to ravi by giving all the information along with the address.
- Hari works as an agent (or object) who performs the task of satisfying raju's request. Hari then performs a set of operations or methods to satisfy the request which is actually hidden from ravi, Hari forwards a message to ravi's local florist. Then local florist asks a delivery person to deliver those flowers to ravi. The objects or agents helping raju in solving the problem of sending flowers to his friend ravi can be shown in figure.



Responsibilities

- A fundamental concept in OOP is to describe behavior in terms of responsibilities. A Request to perform an action denotes the desired result. An object can use any technique that helps in obtaining the desired result and this process will not have any interference from other object. The abstraction level will be increased when a problem is evaluated in terms of responsibilities. The objects will thus become independent from each other which will help in solving complex problems. An Object has a collection of responsibilities related with it which is termed as 'protocol'
- The Operation of a traditional program depends on how it acts on data structures. Where as an OOP operates by requesting data structure to perform a service.

Messages & Methods

In object-oriented programming, every action is initiated by passing a message to an agent (object), which is responsible for the action. The receiver is the object to whom the message was sent. In response to the message, the receiver performs some method to carry out the request. Every message may include any additional information as arguments.

Classes & Instances

In object-oriented programming, all objects are instances of a class. The method invoked by an object in response to a message is decided by the class. All the objects of a class use the same method in response to a similar message.

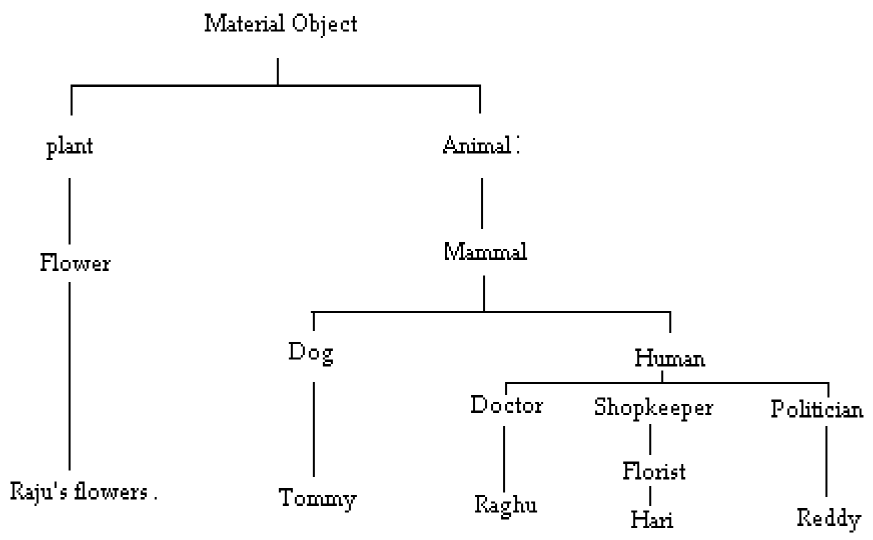
All objects are said to be the instances of a class.

For e.g., If 'flower' is a class then Rose is its instance

Class Hierarchies (Inheritance)

A graphical representation is often used to illustrate the relationships among the classes (objects) of a community. This graphical representation shows classes listed in a hierarchical tree-like structure. In this more abstract class listed near the top of the tree, and more specific classes in the middle of the tree, and the individuals listed near the bottom.

In object-oriented programming, classes can be organized into a hierarchical inheritance structure. A child class inherits properties from the parent class that higher in the tree.



Class hierarchy for Different kinds of Material

Object Oriented Thinking

Let 'Hari' be a florist, but florist more specific form of shot keeper. Additionally, a shop keeper is a human and a human is definitely a mammal. But a mammal is an animal & animal is material object.

Method Binding

When the method is super class have same name that of the method in sub class, then the subclass method overridden the super-class method. The program will find out a class to which the reference is actually pointing and that class method will be binded.

E.g.

```
class parent
{
  void print()
{
  System.out.println("From Parent");
}
}
class  child extends parent
{
void print()
{
 System.out.println("From Child");
}
}
class  Bind
{
Public static void main(String arg[])
{
 child ob=new child();
 ob.print();
}
}
```

Output: From Child

The child's object 'ob' will point to child class print() method thus that method will be binded.

Overriding

When the name and type of the method in a subclass is same as that of a method in its super class. Then it is said that the method present in subclass overrides the method present in super class. Calling an overridden method from a subclass will always point to the type of that method as defined by the subclass, where as the type of method defined by super class is hidden.

E.g. (above 'method binding' example)

Exceptions

Exception is a error condition that occurs in the program execution. There is an object called 'Exception' object that holds error information. This information includes the type and state of the program when the error occurred.

E.g. Stack overflow, Memory error etc

Summary of OOP concepts

1. Everything is an object.
2. Computation is performed by objects communicating with each other, requesting that other objects perform actions. Objects communicate by sending & receiving *messages*. A message is a request for an action bundled with whatever arguments may be necessary to complete the task.
3. Each object has its own *memory*, which consists of other objects.
4. Every Object is an *instance* of class. A class simply represents a grouping of similar objects, such as integers or lists.
5. The class is the repository for *behavior* associated with an object. That is all objects that are instances of same class can perform the same actions.
6. Classes are organized into a singly rooted tree structure, called *inheritance hierarchy.*

OOP Concepts

OOP stands for Object-Oriented Programming. OOP is a programming paradigm in which every program is follows the concept of object. In other words, OOP is a way of writing programs based on the object concept.

The object-oriented programming paradigm has the following core concepts.

- Class
- Object
- Encapsulation
- Inheritance
- Polymorphism
- Abstraction

Class

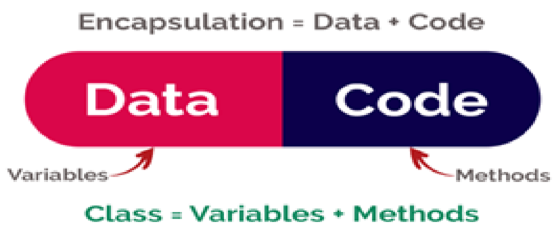
Class is a blue print which is containing only list of variables and methods and no memory is allocated for them. A class is a group of objects that has common properties.

Object

- Any entity that has state and behavior is known as an object.
- For example a chair, pen, table, keyboard, bike, etc. It can be physical or logical.An Object can be defined as an instance of a class.
- **Example:** A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

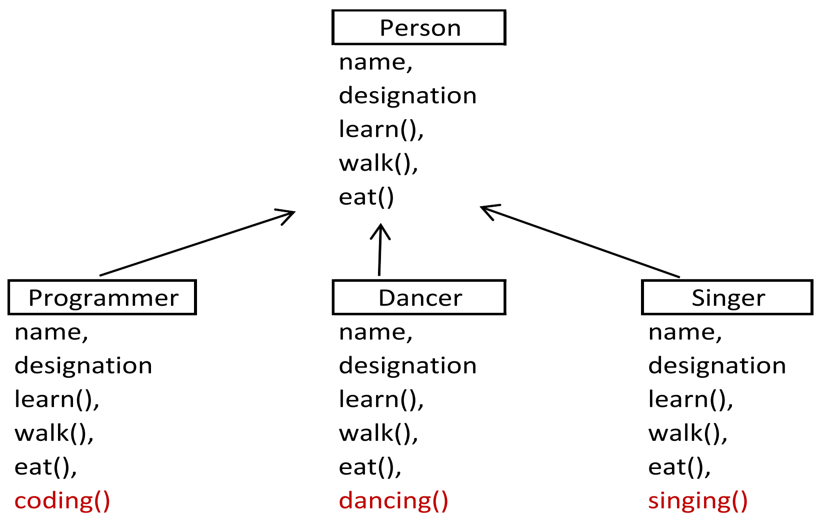
Encapsulation

- Encapsulation is the process of combining data and code into a single unit.
- In OOP, every object is associated with its data and code.
- In programming, data is defined as variables and code is defined as methods.
- The java programming language uses the class concept to implement encapsulation.



Inheritance

- Inheritance is the process of acquiring properties and behaviors from one object to another object or one class to another class.
- In inheritance, we derive a new class from the existing class. Here, the new class acquires the properties and behaviors from the existing class.
- In the inheritance concept, the class which provides properties is called as parent class and the class which recieves the properties is called as child class.



Polymorphism

- Polymorphism is the process of defining same method with different implementation. That means creating multiple methods with different behaviors.
- The java uses method overloading and method overriding to implement polymorphism.

- Method overloading - multiple methods with same name but different parameters.
- Method overriding - multiple methods with same name and same parameters.

Abstraction

- Abstraction is hiding the internal details and showing only esential functionality.
- In the abstraction concept, we do not show the actual implemention to the end user, instead we provide only esential things.
- For example, if we want to drive a car, we does not need to know about the internal functionality like how wheel system works? how brake system works? how music system works? etc.

Java Buzzwords(Java Features)

Following are the list of buzzwords:

- Simple
- Secure
- Portable
- Object-oriented
- Robust
- Multithreaded
- Architecture-neutral
- Interpreted
- High performance
- Distributed
- Dynamic

Simple

- Java programming language is very simple and easy to learn, understand, and code.
- Most of the syntaxes in java follow basic programming language C and object-oriented programming concepts are similar to C++.
- In a java programming language, many complicated features like pointers, operator overloading, structures, unions, etc. have been removed.
- One of the most useful features is the garbage collector it makes java more simple.

Secure

It is a more secure language compared to other language; In this language, all code is covered in byte code after compilation which is not readable by human.

Portability

If any language supports platform independent and architectural neutral feature known as portable. The languages like C, CPP, Pascal are treated as non-portable language. Java is a portable language.

Object-Oriented

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.

Robust

Simply means of Robust are strong. It is robust or strong Programming Language because of its capability to handle Run-time Error, automatic garbage collection, the lack of pointer concept, Exception Handling. All these points make It robust Language.

Multithreaded

Java supports multithreaded programming, which allows you to write programs that do many things simultaneously.
A multithreaded program contains two or more parts that can run concurrently.Each part of such program is called a thread.

Architecture-Neutral

Architecture represents processor.A Language or Technology is said to be Architectural neutral which can run on any available processors in the real world without considering their development and compilation.

Interpreted and High Performance

It have high performance because of following reasons;

- This language **uses Bytecode** which is faster than ordinary pointer code so Performance of this language is high.
- **Garbage collector**, collect the unused memory space and improve the performance of the application.
- It has **no pointers** so that using this language we can develop an application very easily.
- It **support multithreading**, because of this time consuming process can be reduced to executing the program.

Distributed

Java is designed for the distributed environment of the Internet because it handles TCP/IP protocols. Java also supports Remote Method Invocation (RMI). This feature enables a program to invoke methods across a network.

Dynamic

It supports Dynamic memory allocation due to this memory wastage is reduce and improve performance of the application. The process of allocating the memory space to the input of the program at a run-time is known as dynamic memory allocation, To programming to allocate memory space by dynamically we use an operator called 'new' 'new' operator is known as dynamic memory allocation operator.

Overview of Java

- Java is a computer programming language.
- Java was created based on C and C++.
- Java uses C syntax and many of the object-oriented features are taken from C++.
- Before Java was invented there were other languages like COBOL, FORTRAN, C, C++, Small Talk, etc.
- These languages had few disadvantages which were corrected in Java.
- Java also innovated many new features to solve the fundamental problems which the previous languages could not solve.
- Java was developed by **James Gosling**, Patrick Naughton, Chris warth, Ed Frank and Mike Sheridon at Sun Microsystems in the year 1991.
- This language was initially called as "OAK" but was renamed as "Java" in 1995.
- The primary motivation behind developing java was the need for creating a platform independent Language (Architecture Neutral), that can be used to create a software which can be embedded in various electronic devices such as remote controls, micro ovens etc.
- The problem with C, C++ and most other languages is that, they are designed to compile on specific targeted CPU (i.e. they are platform dependent), but java is platform Independent which can run on a variety of CPU's under different environments.
- The secondary factor that motivated the development of java is to develop the applications that can run on Internet. Using java we can develop the applications which can run on internet i.e. Applet. So java is a platform Independent Language used for developing programs which are platform Independent and can run on internet.

Datatypes

Java programming language has a rich set of data types. The data type is a category of data stored in variables. In java, data types are classified into four types and they are as follows.

Integers

This group includes byte, short, int, and long.

Name	Width	Range
Long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
Int	32	–2,147,483,648 to 2,147,483,647
Short	16	–32,768 to 32,767
Byte	8	–128 to 127

Syntax: byte b, c;
int i;
short s;
long distance;

Floating point types

- Floating-point numbers, also known as real numbers, are used when evaluating expressions that require fractional precision.
- There are two kinds: float and double.

Name	Width in Bits	Approximate Range
Double	64	4.9e–324 to 1.8e+308
float	32	1.4e–045 to 3.4e+038

Syntax: float hightemp, lowtemp;
Double radius, pi;

Characters

In Java, the data type used to store characters is char.

Name	Width in Bits	Approximate Range
char	16	0 to 65,536

There are no negative chars in java.
Syntax: car c=’A’, c1=’X’, c2=’Z’;

Boolean

Java has a primitive type, called Boolean, for logical values. It can have only one of two possible values, true or false.

```
a=true;
b=false;
```

Example

```
public class DataTypes
{
     public static void main(String[] args)
    {
          byte b=12;
          short s=45;
          int i=3467;
          long l=2434355;
          float f=23.5f;
          double d=43545.7;
          char ch='A';
          boolean bo=true;
         System.out.println("b="+b+"s="+s+"i="+i+"l="+l);
         System.out.println("f="+f+ "d="+d);
          System.out.println("ch="+ch);
          System.out.println("bo="+bo);
     }
}
```

Variables

A variable is a named memory location used to store a data value. A variable can be defined as a container that holds a data value.

In java, we use the following syntax to create variables.

Syntax

```
data_type variable_name;
      (or)
data_type variable_name_1, variable_name_2,...;
     (or)
data_type variable_name = value;
     (or)
```

data_type variable_name_1 = value, variable_name_2 = value,.

Examples of variable declarations of various types.

- int a, b, c;   // declares three ints, a, b, and c.
- int d = 3, e, f = 5;   // declares three more ints, initializing // d and f.
- byte z = 22;   // initializes z.
- double pi = 3.14159;   // declares an approximation of pi.
- char x = 'x';

Arrays

- An array is a collection of similar data values with a single name.
- An array can also be defined as, a special type of variable that holds multiple values of the same data type at a time.
- In java, arrays are objects and they are created dynamically using **new** operator.
- Every array in java is organized using index values.
- The index value of an array starts with '0' and ends with 'zise-1'.
- We use the index value to access individual elements of an array.

In java, there are two types of arrays and they are as follows.

- One Dimensional Array
- Multi Dimensional Array

One Dimensional Array

Creating an array

In the java programming language, an array must be created using new operator and with a specific size. The size must be an integer value but not a byte, short, or long. We use the following syntax to create an array.

Syntax

```
data_type array_name[ ] = new data_type[size];
                        (or)
data_type[ ] array_name = new data_type[size];
```

Example

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

```
    {
      int list[] = new int[5];
      list[0] = 10;
      System.out.println("Value at index 0 - " + list[0]);
      System.out.println("Length of the array - " + list.length);

    }
}
```

In java, an array can also be initialized at the time of its declaration. When an array is initialized at the time of its declaration, it need not specify the size of the array and use of the new operator. Here, the size is automatically decided based on the number of values that are initialized.

Example

```
int list[ ] = {10, 20, 30, 40, 50};
```

Multidimensional Array

- In java, we can create an array with multiple dimensions. We can create 2-dimensional, 3-dimensional, or any dimensional array.
- In Java, multidimensional arrays are arrays of arrays.
- To create a multidimensional array variable, specify each additional index using another set of square brackets.

Syntax

```
data_type array_name[ ][ ] = new data_type[rows][columns];
                    (or)
data_type[ ][ ] array_name = new data_type[rows][columns];
```

When an array is initialized at the time of declaration, it need not specify the size of the array and use of the new operator. Here, the size is automatically decided based on the number of values that are initialized.

Example

```
class TwoDArray
 {
    public static void main(String args[])
    {
        int twoD[][]= new int[4][5];
        int i, j, k = 0;
```

```
         for(i=0; i<4; i++)
         for(j=0; j<5; j++)
        {
            twoD[i][j] = k;
            k++;
        }
        for(i=0; i<4; i++)
        {
            for(j=0; j<5; j++)
            System.out.print(twoD[i][j] + " ");
            System.out.println();
        }
      }
    }
```

Operators

An operator is a symbol used to perform arithmetic and logical operations. Java provides a rich set of operators.

In java, operators are clasiffied into the following four types.

- Arithmetic Operators
- Bitwise Operators
- Relational (or) Comparision Operators
- Logical Operators

Arithmetic Operators

The following table lists the arithmetic operators:

Operator	Result
+	Addition
–	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
–=	Subtraction assignment
*=	Multiplication assignment

/=	Division assignment
%=	Modulus assignment
– –	Decrement

Example

```
class BasicMath
 {
      public static void main(String args[])
      {
             System.out.println("Integer Arithmetic");
             int a = 1 + 1;
             int b = a * 3;
             int c = b / 4;
             int d = c - a;
             int e = -d;
             System.out.println("a = " + a);
             System.out.println("b = " + b);
             System.out.println("c = " + c);
             System.out.println("d = " + d);
             System.out.println("e = " + e);
     }
}
```

Bitwise Operators

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

Example

```
public class BitwiseOperators
{
      public static void main(String[] args)
     {
              int a = 25, b = 20;
              System.out.println(a + " & " + b + " = " + (a & b));
              System.out.println(a + " | " + b + " = " + (a | b));
              System.out.println(a + " ^ " + b + " = " + (a ^ b));
              System.out.println(a + ">>" + 2 + " = " + (a>>2));
              System.out.println(a + "<<" + 2 + " = " + (a<<2));
      }

 }
```

Relational Operators

Operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Example

```
public class RelationalOperators
{
      public static void main(String[] args)
      {
              boolean a;
              a = 10<5;
              System.out.println("10 < 5 is " + a);
              a = 10>5;
              System.out.println("10 > 5 is " + a);
              a = 10<=5;
              System.out.println("10 <= 5 is " + a);
              a = 10>=5;
              System.out.println("10 >= 5 is " + a);
              a = 10==5;
```

```
                System.out.println("10 == 5 is " + a);
                a = 10!=5;
                System.out.println("10 != 5 is " + a);
        }
   }
```

Logical Operators

Operator	Result
&	Logical AND
|	Logical OR
^	Logical XOR (exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

Example

```
  public class LogicalOperators
 {
        public static void main(String[] args)
       {
                int x = 10, y = 20, z = 0;
                boolean a = true;
                a = x>y && (z=x+y)>15;
                System.out.println("a = " + a + ", and z = " + z);
                a = x>y & (z=x+y)>15;
                System.out.println("a = " + a + ", and z = " + z);
        }
 }
```

Expressions

- In any programming language, if we want to perform any calculation or to frame any condition etc., we use a set of symbols to perform the task. These set of symbols makes an expression.

  In the java programming language, an expression is defined as follows..
- An expression is a collection of operators and operands that represents a specific value.
- In the above definition, an operator is a symbol that performs tasks like arithmetic operations, logical operations, and conditional operations, etc.

Expression Types

In the java programming language, expressions are divided into THREE types. They are as follows.

- **Infix Expression**
- **Postfix Expression**
- **Prefix Expression**

The above classification is based on the operator position in the expression.

Infix Expression

The expression in which the operator is used between operands is called infix expression. The infix expression has the following general structure.

Example

a+b

Postfix Expression

The expression in which the operator is used after operands is called postfix expression. The postfix expression has the following general structure.

Example

ab+

Prefix Expression

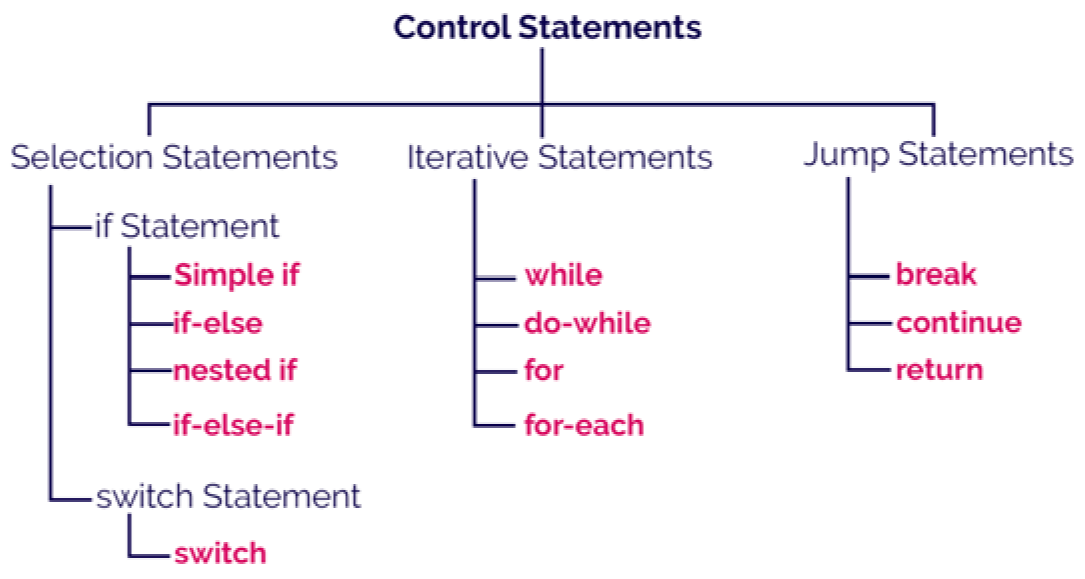
The expression in which the operator is used before operands is called a prefix expression. The prefix expression has the following general structure.

Example

+ab

Control Statements

In java, the default execution flow of a program is a sequential order. But the sequential order of execution flow may not be suitable for all situations. Sometimes, we may want to jump from line to another line, we may want to skip a part of the program, or sometimes we may want to execute a part of the program again and again. To solve this problem, java provides control statements.



Types of Control Statements

In java, the control statements are classified as follows.

1. Selection Control Statements (Decision Making Statements)
2. Iterative Control Statements (Looping Statements)
3. Jump Statements

1.Selection Control Statements

In java, the selection statements are also known as decision making statements or branching statements. The selection statements are used to select a part of the program to be executed based on a condition.

Java provides the following selection statements.

- if statement
- if-else statement
- if-elif statement
- nested if statement
- switch statement

if statement in java

- In java, we use the if statement to test a condition and decide the execution of a block of statements based on that condition result.
- The if statement checks, the given condition then decides the execution of a block of statements. If the condition is True, then the block of statements is executed and if it is False, then the block of statements is ignored.

Syntax

```
if(condtion)
{
   if-block of statements;
}
statement after if-block;
```

Example

```
import java.util.Scanner;
public class IfStatementTest
{
        public static void main(String[] args)
       {
                Scanner read = new Scanner(System.in);
                System.out.print("Enter any number: ");
                int num = read.nextInt();
                if((num % 5) == 0)
               {
                        System.out.println("We are inside the if-block!");
                        System.out.println("Given number is divisible by 5!!");
                }
                System.out.println("We are outside the if-block!!!");
        }
}
```

In the above execution, the number 12 is not divisible by 5. So, the condition becomes False and the condition is evaluated to False. Then the if statement ignores the execution of its block of statements.

if-else statement in java

- In java, we use the if-else statement to test a condition and pick the execution of a block of statements out of two blocks based on that condition result.
- The if-else statement checks the given condition then decides which block of statements to be executed based on the condition result.
- If the condition is True, then the true block of statements is executed and if it is False, then the false block of statements is executed.

Syntax

```
if(condtion)
{
   true-block of statements;
}
else
{
   false-block of statements;
}
statement after if-block;
```

Example

```
import java.util.Scanner;
public class IfElseStatementTest
{
        public static void main(String[] args)
       {
               Scanner read = new Scanner(System.in);
               System.out.print("Enter any number: ");
               int num = read.nextInt();
               if((num % 2) == 0)
             {
                       System.out.println("We are inside the true-block!");
                       System.out.println("Given number is EVEN number!!");
               }
               else
             {
                       System.out.println("We are inside the false-block!");
                       System.out.println("Given number is ODD number!!");
               }
               System.out.println("We are outside the if-block!!!");
        }
}
```

Nested if statement in java

Writing an if statement inside another if-statement is called nested if statement.

Syntax

```
if(condition_1)
{
   if(condition_2)
   {
      inner if-block of statements;
      ...
   }
   ...
}
```

Example

```
import java.util.Scanner;
public class NestedIfStatementTest
{
          public static void main(String[] args)
         {
                    Scanner read = new Scanner(System.in);
                    System.out.print("Enter any number: ");
                    int num = read.nextInt();
                    if (num < 100)
                  {
                              System.out.println("\nGiven number is below 100");
                              if (num % 2 == 0)
                                        System.out.println("And it is EVEN");
                              else
                                        System.out.println("And it is ODD");
                     }
                   else
                              System.out.println("Given number is not below 100");
                    System.out.println("\nWe are outside the if-block!!!");
          }
}
```

if-else if statement in java

Writing an if-statement inside else of an if statement is called if-else-if statement.

Syntax

```
if(condition_1)
{
   condition_1 true-block;
   ...
}
else if(condition_2)
{
   condition_2 true-block;
   condition_1 false-block too;
   ...
}
```

Example

```
import java.util.Scanner;
public class IfElseIfStatementTest
{
        public static void main(String[] args)
       {
                int num1, num2, num3;
                Scanner read = new Scanner(System.in);
                System.out.print("Enter any three numbers: ");
                num1 = read.nextInt();
                num2 = read.nextInt();
                num3 = read.nextInt();
                if( num1>=num2 && num1>=num3)
                        System.out.println("\nThe largest number is " + num1) ;
               else if (num2>=num1 && num2>=num3)
                       System.out.println("\nThe largest number is " + num2) ;

                  else
                       System.out.println("\nThe largest number is " + num3) ;
                System.out.println("\nWe are outside the if-block!!!");
        }
}
```

Switch

- Using the switch statement, one can select only one option from more number of options very easily.
- In the switch statement, we provide a value that is to be compared with a value associated with each option. Whenever the given value matches the value associated with an option, the execution starts from that option.
- In the switch statement, every option is defined as a **case**.

Syntax:

```
switch (expression)
{
        case value1: // statement sequence
                break;
        case value2: // statement sequence
               break;
          ....
         case valueN:
}
```

Example

```
class SampleSwitch
{
     public static void main(String args[])
     {
         for(int i=0; i<6; i++)
         {
            switch(i)
            {
              case 0:System.out.println("i is zero.");
                     break;
              case 1:System.out.println("i is one.");
                     break;
              case 2:System.out.println("i is two.");
              default:System.out.println("i is greater than 3.");
            }
          }
        }
     }
```

2.Iteration Statements

- The java programming language provides a set of iterative statements that are used to execute a statement or a block of statements repeatedly as long as the given condition is true.
- The iterative statements are also known as looping statements or repetitive statements. Java provides the following iterative statements.
    1. while statement
    2. do-while statement
    3. for statement
    4. for-each statement

while statement in java

The while statement is used to execute a single statement or block of statements repeatedly as long as the given condition is TRUE. The while statement is also known as Entry control looping statement.

Syntax

```
 while(condition)
{
     // body of loop
 }
```

Example

```
public class WhileTest
{
      public static void main(String[] args)
     {
              int num = 1;
              while(num <= 10)
             {
                      System.out.println(num);
                      num++;
              }
              System.out.println("Statement after while!");
      }
}
```

do-while statement in java

- The do-while statement is used to execute a single statement or block of statements repeatedly as long as given the condition is TRUE.
- The do-while statement is also known as the Exit control looping statement.

Syntax

```
do
{
    // body of loop
 } while (condition);
```

Example

```
public class DoWhileTest
{
    public static void main(String[] args)
   {
            int num = 1;
            do
          {
                    System.out.println(num);
                    num++;
            }while(num <= 10);
            System.out.println("Statement after do-while!");
     }
}
```

for statement in java

The for statement is used to execute a single statement or a block of statements repeatedly as long as the given condition is TRUE.

Syntax

```
for(initialization; condition; inc/dec)
{
     // body
}
```

If only one statement is being repeated, there is no need for the curly braces.

In for-statement, the execution begins with the **initialization** statement. After the initialization statement, it executes **Condition**. If the condition is evaluated to true, then the block of statements executed otherwise it terminates the for-statement. After the block of statements execution, the **modification** statement gets executed, followed by condition again.

Example

```
public class ForTest
{
        public static void main(String[] args)
       {
                for(int i = 0; i < 10; i++)
               {
                        System.out.println("i = " + i);
                }
                System.out.println("Statement after for!");
        }
}
```

3.Jump Statements

The java programming language supports jump statements that used to transfer execution control from one line to another line.
The java programming language provides the following jump statements.

1. break statement
2. continue statement

break

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

Example

```
class BreakStatement
{
       public static void main(String args[] )
       {
              int i;
              i=1;
              while(true)
              {
                     if(i >10)
                     break;
```

```
                System.out.print(i+" ");
                i++;
            }
        }
}
```

Continue

This command skips the whole body of the loop and executes the loop with the next iteration. On finding continue command, control leaves the rest of the statements in the loop and goes back to the top of the loop to execute it with the next iteration (value).

Example

```
/* Print Number from 1 to 10 Except 5 */
class NumberExcept
{
        public static void main(String args[] )
        {
                int i;
                for(i=1;i<=10;i++)
                {
                        if(i==5) continue;
                        System.out.print(i +" ");
                }
        }
}
```

Introducing Classes and Methods

- classes usually consist of two things: instance variables and methods.
- Java is an object-oriented programming language, so everything in java program must be based on the object concept.
- In a java programming language, the class concept defines the skeleton of an object.
- The java class is a template of an object.
- The class defines the blueprint of an object.
- Every class in java forms a new data type.
- Once a class got created, we can generate as many objects as we want.
- Every class defines the properties and behaviors of an object.
- All the objects of a class have the same properties and behaviors that were defined in the class.

Every class of java programming language has the following characteristics.

- **Identity** - It is the name given to the class.
- **State** - Represents data values that are associated with an object.
- **Behavior** - Represents actions can be performed by an object.

Creating a Class

In java, we use the keyword class to create a class. A class in java contains properties as variables and behaviors as methods.

Syntax

```
Class ClassName
{
   data members declaration;
   methods defination;
}
```

- The ClassName must begin with an alphabet, and the Upper-case letter is preferred.
- The ClassName must follow all naming rules.

Example

Here is a class called Box that defines three instance variables: width, height, and depth.

```
class Box
 {
      double width;
      double height;
      double depth;
      void volume()
      {
          …………………
      }
}
```

Creating an Object

- In java, an object is an instance of a class. When an object of a class is created, the class is said to be instantiated.
- All the objects that are created using a single class have the same properties and methods.
- But the value of properties is different for every object.

Syntax

```
ClassName objectName = new ClassName( );
```

- The objectName must begin with an alphabet, and a Lower-case letter is preferred.
- The objectName must follow all naming rules.

Example

To actually create a Box object, you will use a statement like the following:

```
Box  mybox = new Box();
```

The new operator dynamically allocates memory for an object.

Example

```
class Box
{
        double width;
        double height;
        double depth;
}
class BoxDemo
{
     public static void main(String args[])
     {
              Box mybox = new Box();
              double vol;
              mybox.width = 10;
              mybox.height = 20;
              mybox.depth = 15;
              vol = mybox.width * mybox.height * mybox.depth;
              System.out.println("Volume is " + vol);
     }
}
```

Methods

- A method is a block of statements under a name that gets executes only when it is called.
- Every method is used to perform a specific task. The major advantage of methods is code re-usability (define the code once, and use it many times).
- In a java programming language, a method defined as a behavior of an object. That means, every method in java must belong to a class.

- Every method in java must be declared inside a class.

Every method declaration has the following characteristics.

- **returnType** - Specifies the data type of a return value.
- **name** - Specifies a unique name to identify it.
- **parameters** - The data values it may accept or recieve.
- **{ }** - Defienes the block belongs to the method.

Creating a method

A method is created inside the class

Syntax

```
class ClassName
{
    returnType methodName( parameters )
    {
                // body of method
    }
}
```

Calling a method

- In java, a method call precedes with the object name of the class to which it belongs and a dot operator.
- It may call directly if the method defined with the static modifier.
- Every method call must be made, as to the method name with parentheses (), and it must terminate with a semicolon.

Syntax

```
objectName.methodName(actualArguments );
```

Example

```
//Adding a Method to the Box Class
Class Box
{
    double width, height, depth;
    void volume()
    {
    System.out.print("Volume is ");
    System.out.println(width * height * depth);
```

```
        }
    }
    class BoxDemo3
    {
        public static void main(String args[])
        {
            Box mybox1 = new Box();
            Box mybox2 = new Box();
            mybox1.width = 10;
            mybox1.height = 20;
            mybox1.depth = 15;
            mybox2.width = 3;
            mybox2.height = 6;
            mybox2.depth = 9;
            mybox1.volume();
            mybox2.volume();
        }
}
```

String Handling

- A string is a sequence of characters surrounded by double quotations. In a java programming language, a string is the object of a built-in class **String**.
- The string created using the **String** class can be extended. It allows us to add more characters after its definition, and also it can be modified.

Example

```
String siteName = "javaprogramming";
siteName = "javaprogramminglanguage";
```

String handling methods

In java programming language, the String class contains various methods that can be used to handle string data values.

The following table depicts all built-in methods of String class in java.

S.No	Method	Description
1	charAt(int)	Finds the character at given index
2	length()	Finds the length of given string
3	compareTo(String)	Compares two strings

4	compareToIgnoreCase(String)	Compares two strings, ignoring case
5	concat(String)	Concatenates the object string with argument string.
6	contains(String)	Checks whether a string contains sub-string
7	contentEquals(String)	Checks whether two strings are same
8	equals(String)	Checks whether two strings are same
9	equalsIgnoreCase(String)	Checks whether two strings are same, ignoring case
10	startsWith(String)	Checks whether a string starts with the specified string
11	isEmpty()	Checks whether a string is empty or not
12	replace(String, String)	Replaces the first string with second string
13	replaceAll(String, String)	Replaces the first string with second string at all occurrences.
14	substring(int, int)	Extracts a sub-string from specified start and end index values
15	toLowerCase()	Converts a string to lower case letters
16	toUpperCase()	Converts a string to upper case letters
17	trim()	Removes whitespace from both ends
18	toString(int)	Converts the value to a String object

Example

```
public class JavaStringExample
{
  public static void main(String[] args)
       {
          String title = "Java Programming";
          String siteName = "String Handling Methods";
          System.out.println("Length of title: " + title.length());
          System.out.println("Char at index 3: " + title.charAt(3));
          System.out.println("Index of 'T': " + title.indexOf('T'));
          System.out.println("Empty: " + title.isEmpty());
          System.out.println("Equals: " + siteName.equals(title));
          System.out.println("Sub-string: " + siteName.substring(9, 14));
          System.out.println("Upper case: " + siteName.toUpperCase());
       }
}
```

Inheritance in Java

- The process of obtaining the data members and methods from one class to another class is known as **inheritance**. It is one of the fundamental features of object-oriented programming.
- In the inheritance the class which is giving data members and methods is known as base or super or parent class.
- The class which is taking the data members and methods is known as sub or derived or child class.
- The data members and methods of a class are known as features.

Why use Inheritance ?

- For Method Overriding (used for Runtime Polymorphism).
- It's main uses are to enable polymorphism and to be able to reuse code for different classes by putting it in a common super class
- For code Re-usability

Syntax

```
class Subclass-Name extends Superclass-Name
{
      //methods and fields
}
```

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

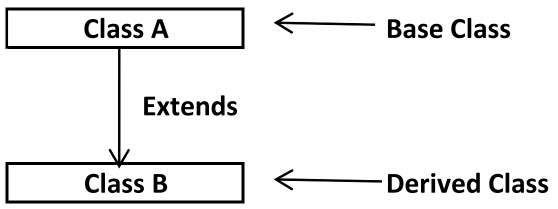
Tpyes of Inheritance

Based on number of ways inheriting the feature of base class into derived class we have five types of inheritance they are:

1. Single inheritance
2. Multiple inheritance
3. Hierarchical inheritance
4. Multilevel inheritance
5. Hybrid inheritance

1.Single inheritance

In single inheritance there exists single base class and single derived class.

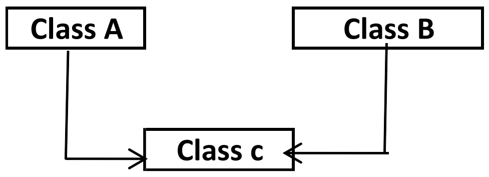


Example

```
class Faculty
{
  float salary=30000;
}
class Science extends Faculty
{
     float bonous=2000;
     public static void main(String args[])
    {
        Science obj=new Science();
        System.out.println("Salary is:"+obj.salary);
        System.out.println("Bonous is:"+obj.bonous);
    }
}
```

2.Multiple inheritance

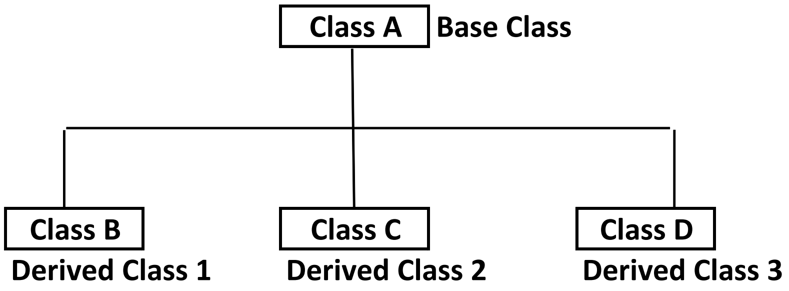
In multiple inheritance there exist multiple classes and singel derived class.



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

3.Hierarchical Inheritance

In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass. In the below image, class A serves as a base class for the derived class B, C and D.



Example

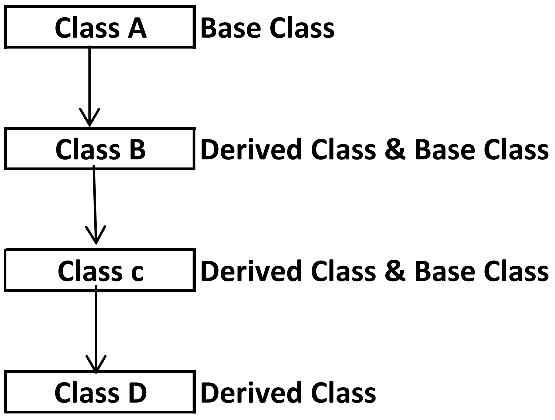
```
class A
{
   public void print_A()
  {
     System.out.println("Class A");
   }
}
class B extends A
{
   public void print_B()
   {
      System.out.println("Class B");
   }
}
class C extends A
{
   public void print_C()
   {
    System.out.println("Class C");
   }
 }
class D extends A
{
   public void print_D()
   {
     System.out.println("Class D");
   }
 }
public class Test
{
   public static void main(String[] args)
```

```
        {
           B obj_B = new B();
           obj_B.print_A();
           obj_B.print_B();
           C obj_C = new C();
           obj_C.print_A();
           obj_C.print_C();
           D obj_D = new D();
           obj_D.print_A();
           obj_D.print_D();
        }
     }
```

4.Multilevel inheritances in Java

In Multilevel inheritances there exists single base class, single derived class and multiple intermediate base classes.

Single base class + single derived class + multiple intermediate base classes.



Hence all the above three inheritance types are supported by both classes and interfaces.

Example

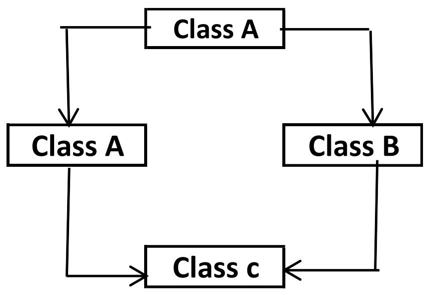
```
class Faculty
{
   float total_sal=0, salary=30000;
}
class HRA extends Faculty
{
   float hra=3000;
}
```

```
class DA extends HRA
{
   float da=2000;
}
class Science extends DA
{
       float bonous=2000;
       public static void main(String args[])
      {
          Science obj=new Science();
          obj.total_sal=obj.salary+obj.hra+obj.da+obj.bonous;
          System.out.println("Total Salary is:"+obj.total_sal);
      }
}
```

5.Hybrid inheritance

Combination of any inheritance type

In the combination if one of the combination is multiple inheritance then the inherited combination is not supported by java through the classes concept but it can be supported through the concept of interface.



Access Control(Member Access)

Java supports rich set of access specifiers, they are:

1.public
2.private
3.protected
4.default access level.

1.public:

When a member of a class is modified by the public specifier, then that member can be accessed by any other code.

Eg: public int a;

2.private:

When a member of a class is specified as private, then that member can only be accessed by other members of its class.

Eg: private int b;

3.protected:

- This is applied only when inheritance is used.
- When the member of a class is declared as protected, then that member can be used by all the classes of same package but by only subclasses which belong to different packages.

Eg: protected int c;

4.default:

When no access specifier is used, then by default the member of a class is public but within its own package i.e. this member can be used by all the classes that belongs to same package.

Table: class member access

Access Specifiers / **Accessibility Location**	**private**	**public**	**protected**	**default**
Same class	yes	yes	yes	yes
Same package subclass	No	yes	yes	yes
Same package non-subclass	No	yes	yes	yes
Different package subclass	No	yes	yes	No
Different package non-subclass	No	yes	No	No

Example

```
class ParentClass
{
        int a = 10;
        public int b = 20;
        protected int c = 30;
        private int d = 40;
        void showData()
       {
                System.out.println("Inside ParentClass");
                System.out.println("a = " + a);
                System.out.println("b = " + b);
                System.out.println("c = " + c);
                System.out.println("d = " + d);
        }
}
class ChildClass extends ParentClass
{
        void accessData()
       {
                System.out.println("Inside ChildClass");
                System.out.println("a = " + a);
                System.out.println("b = " + b);
                System.out.println("c = " + c);
                //System.out.println("d = " + d);     // private member can't be accessed
        }
}
public class AccessModifiersExample
{
        public static void main(String[] args)
       {
                ChildClass obj = new ChildClass();
                obj.showData();
                obj.accessData();
        }
}
```

Constructors

- Constructor in Java is a special member method which will be called automatically by the JVM whenever an object is created for placing user defined values in place of default values.
- In a single word constructor is a special member method which will be called automatically whenever object is created.
- The purpose of constructor is to initialize an object called object initialization. Initialization is a process of assigning user defined values at the time of allocation of memory space.

Syntax

```
ClassName()
{
.......
.......
}
```

Types of constructors

Based on creating objects in Java constructor are classified in two types. They are

1.Default or no argument Constructor

2.Parameterized constructor.

1.Default Constructor

- A constructor is said to be default constructor if and only if it never take any parameters.
- If any class does not contain at least one user defined constructor than the system will create a default constructor at the time of compilation it is known as system defined default constructor.

Note: System defined default constructor is created by java compiler and does not have any statement in the body part. This constructor will be executed every time whenever an object is created if that class does not contain any user defined constructor.

Example

```
class Test
{
   int a, b;
   Test()
   {
        a=10;
        b=20;
```

```
            System.out.println("Value of a: "+a);
            System.out.println("Value of b: "+b);
    }
  }
 class TestDemo
 {
     public static void main(String args[])
     {
         Test t1=new Test();
     }
 }
```

2.Parameterized constructor

If any constructor contain list of variables in its signature is known as paremetrized constructor. A parameterized constructor is one which takes some parameters.

Example

```
class Test
 {
     int a, b;
     Test(int n1, int n2)
     {
          a=n1;
          b=n2;
          System.out.println("Value of a = "+a);
          System.out.println("Value of b = "+b);
    }
  }
  class TestDemo
  {
          public static void main(String args[])
           {
                 Test t1=new Test(10, 20);
           }
   }
```

Difference between Method and Constructor

	Method	Constructor
1	Method can be any user defined name	Constructor must be class name
2	Method should have return type	It should not have any return type (even void)
3	Method should be called explicitly either with object reference or class reference	It will be called automatically whenever object is created
4	Method is not provided by compiler in any case.	The java compiler provides a default constructor if we do not have any constructor.

Super Keyword

Super keyword in java is a reference variable that is used to refer parent class features.

Usage of Java super Keyword

1. Super keyword At Variable Level
2. Super keyword At Method Level

- Whenever the derived class is inherits the base class features, there is a possibility that base class features are similar to derived class features and JVM gets an ambiguity.
- In order to differentiate between base class features and derived class features must be preceded by super keyword.

Syntax

super.baseclass features.

1.Super Keyword at Variable Level

- Whenever the derived class inherit base class data members there is a possibility that base class data member are similar to derived class data member and JVM gets an ambiguity.
- In order to differentiate between the data member of base class and derived class, in the context of derived class the base class data members must be preceded by super keyword.

Syntax

super.baseclass datamember name

Example

```
class Employee
{
   int salary=10000;
}
class HR extends Employee
{
  int salary=20000;
  void display()
   {
        System.out.println(salary);
        System.out.println(super.salary);

   }
 }
class Supervarible
{
   public static void main(String[] args)
   {
     HR obj=new HR();
     obj.display();
   }
}
```

2.Super Keyword at Method Level

The **super keyword** can also be used to invoke or call parent class method. It should be use in case of method overriding. In other word **super keyword** use when base class method name and derived class method name have same name.

Example

```
class Student
{
      void message()
      {
      System.out.println("Good Morning Sir");
      }
}
```

```
  class Faculty extends Student
  {
        void message()
          {
          System.out.println("Good Morning Students");
          }
       void display()
        {
          message();
          super.message();
        }
  public static void main(String args[])
   {
  Faculty f=new Faculty();
  f.display();
   }
}
```

Using final Kewword

It is used to make a variable as a constant, Restrict method overriding, Restrict inheritance.

In java language final keyword can be used in following ways:

1. Final Keyword at Variable Level
2. Final Keyword at Method Level
3. Final Keyword at Class Level

1.Final at variable level

Final keyword is used to make a variable as a constant. This is similar to const in other language. A variable declared with the final keyword cannot be modified by the program after initialization. This is useful to universal constants, such as "PI".

Example

```
public class Circle
{
     public  static final double PI=3.14159;
     public static void main(String[] args)
     {
        System.out.println(PI);
     }
}
```

2.Final Keyword at method level

- It makes a method final, meaning that sub classes can not override this method. The compiler checks and gives an error if you try to override the method.
- When we want to restrict overriding, then make a method as a final.

Example

```
class Employee
{
 final void disp()
  {
   System.out.println("Hello Good Morning");
  }
}
class Developer extends Employee
{
   void disp()
   {
     System.out.println("How are you ?");
   }
 }
class FinalDemo
 {
    public static void main(String args[])
    {
        Developer obj=new Developer();
         obj.disp();
    }
 }
```

Output

It gives an error

3.Final Keyword at Class Level

It makes a class final, meaning that the class can not be inheriting by other classes. When we want to restrict inheritance then make class as a final.

Example

```
public final class A
{
   ......
```

```
        ......
      }
      public class B extends  A
      {
          // it gives an error, because we can not inherit final class
      }
```

Polymorphism

The polymorphism is the process of defining same method with different implementation. That means creating multiple methods with different behaviors.

In java, polymorphism implemented using method overloading and method overriding.

Ad hoc polymorphism(Method Overloading)

Whenever same method name is exiting multiple times in the same class with different number of parameter or different order of parameters or different types of parameters is known as **method overloading**.

Example

```
  class Addition
  {
      void sum(int a, int b)
      {
        System.out.println(a+b);
      }
      void sum(int a, int b, int c)
      {
        System.out.println(a+b+c);
      }
      void sum(int a, int b)
      {
        System.out.println(a+b);
      }
      void sum(float a, float b)
      {
        System.out.println(a+b);
      }
      public static void main(String args[])
      {
       Addition obj=new Addition();
        obj.sum(10, 20);
        obj.sum(10, 20, 30);
```

```
            obj.sum(10, 20);
            obj.sum(10.05, 15.20);
        }
    }
```

Pure polymorphism(Method Overriding)

- Whenever same method name is existing in both base class and derived class with same types of parameters or same order of parameters is known as **method Overriding**.
- In a java programming language, pure polymorphism carried out with a method overriding concept.

Note: Without Inheritance method overriding is not possible.

Example

```
 class Walking
{
      void walk()
      {
        System.out.println("Man walking fastly");
      }
 }
class Man extends walking
{
      void walk()
      {
        System.out.println("Man walking slowly");
      }
}
class OverridingDemo
{
      public static void main(String args[])
      {
          Walking obj = new Walking();
          obj.walk();
      }
}
```

Note:

- Whenever we are calling overridden method using derived class object reference the highest priority is given to current class (derived class). We can see in the above example high priority is derived class.
- super. (super dot) can be used to call base class overridden method in the derived class.

Abstract Classes and Methods

- An abstract class is a class that created using abstract keyword. In other words, a class prefixed with abstract keyword is known as an abstract class.
- In java, an abstract class may contain abstract methods (methods without implementation) and also non-abstract methods (methods with implementation).

Syntax

```
        abstract class className
        {
            ......
        }
```

Abstract method

An abstract method contains only declaration or prototype but it never contains body or definition. In order to make any undefined method as abstract whose declaration is must be predefined by abstract keyword.

Syntax

```
 abstract returntype methodName(List of formal parameter);
```

Example

```
import java.util.*;
abstract class Shape
{
        int length, breadth, radius;
        Scanner input = new Scanner(System.in);
        abstract void printArea();
}
class Rectangle extends Shape
{
        void printArea()
       {
                System.out.println("*** Finding the Area of Rectangle ***");
                System.out.print("Enter length and breadth: ");
                length = input.nextInt();
                breadth = input.nextInt();
                System.out.println("The area of Rectangle is: " + length * breadth);
        }
}
class Triangle extends Shape
{
```

```
            void printArea()
           {
                        System.out.println("\n*** Finding the Area of Triangle ***");
                        System.out.print("Enter Base And Height: ");
                        length = input.nextInt();
                        breadth = input.nextInt();
                        System.out.println("The area of Triangle is: " + (length * breadth) / 2);
            }
}
class Cricle extends Shape
{
            void printArea()
           {
                        System.out.println("\n*** Finding the Area of Cricle ***");
                        System.out.print("Enter Radius: ");
                        radius = input.nextInt();
                        System.out.println("The area of Cricle is: " + 3.14f * radius * radius);
            }
}
public class AbstractClassExample
{
            public static void main(String[] args)
           {
                        Rectangle rec = new Rectangle();
                        rec.printArea();
                        Triangle tri = new Triangle();
                        tri.printArea();
                        Cricle cri = new Cricle();
                        cri.printArea();
            }
}
```

Object Class

There is one special class defined in java library called, Object. All other classes are subclasses of Object. That is, Object is a superclass of all other classes.
Object defines the following methods, which means that they are available in every object.

Method	Purpose
Object clone(	Creates a new object that is the same as the object being cloned.
void finalize( )	Called before an unused object is recycled.
void notify( )	Resumes execution of a thread waiting on the invoking object.
void notifyAll( )	Resumes execution of all threads waiting on the invoking object.
String toString( )	Returns a string that describes the object.
void wait( )	Waits on another thread of execution.
void wait(long milliseconds)	
void wait(long milliseconds, int nanoseconds)	

Forms of Inheritance

- The inheritance concept used for the number of purposes in the java programming language. One of the main purposes is substitutability.
- The substitutability means that when a child class acquires properties from its parent class, the object of the parent class may be substituted with the child class object.
- For example, if B is a child class of A, anywhere we expect an instance of A we can use an instance of B.
- The substitutability can achieve using inheritance, whether using extends or implements keywords.

The following are the differnt forms of inheritance in java.

- Specialization
- Specification
- Construction
- Eextension
- Limitation
- Combination

Specialization

It is the most ideal form of inheritance. The subclass is a special case of the parent class. It holds the principle of substitutability.

Specification

This is another commonly used form of inheritance. In this form of inheritance, the parent class just specifies which methods should be available to the child class but doesn't implement them. The java provides concepts like abstract and interfaces to support this form of inheritance. It holds the principle of substitutability.

Construction

This is another form of inheritance where the child class may change the behavior defined by the parent class (overriding). It does not hold the principle of substitutability.

Eextension

This is another form of inheritance where the child class may add its new properties. It holds the principle of substitutability.

Limitation

This is another form of inheritance where the subclass restricts the inherited behavior. It does not hold the principle of substitutability.

Combination

This is another form of inheritance where the subclass inherits properties from multiple parent classes. Java does not support multiple inheritance type.

Benefits of Inheritance

- Software Reusability (among projects)
- Increased Reliability (resulting from reuse and sharing of well-tested code)
- Code Sharing (within a project)
- Consistency of Interface (among related objects)
- Software Components
- Rapid Prototyping (quickly assemble from pre-existing components)
- Polymorphism and Frameworks (high-level reusable components)
- Information Hiding

The Costs of Inheritance

- Execution Speed
- Program Size
- Message-Passing Overhead
- Program Complexity (in overuse of inheritance)