# UNIT-I Introduction and features

**Introduction about Object Oriented Programming**

Object Oriented Programming is a concept in which problem is divided into number of entities called objects.Now what is an object. Object is a run time entity and built data and function around this entity. So, we can say that object is a combination of data and functions or methods.Data of an object can be accessed only by the functions associated with it.

**Features:**

**Simple and familiar:**
The Java language is easy to learn. Java code is easy to read and write.Java is similar to C/C++ but it removes the drawbacks and complexities of C/C++ like pointers and multiple inheritances. So if you have background in C/C++, you will find Java familiar and easy to learn.

**Object-Oriented:**

Java is a fully object-oriented programming language. It has all OOP features such as abstraction, encapsulation, inheritance and polymorphism.

**Secure:**
The Java platform is designed with security features built into the language and runtime system such as static type-checking at compile time and runtime checking . You never hear about viruses attacking Java applications.

**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.

**Multithreaded:**

The Java platform is designed with multithreading capabilities built into the language. That means you can build applications with many concurrent threads of activity, resulting in highly interactive and responsive applications.

**Platform Independence:**

Java code is compiled into intermediate format (bytecode), which can be executed on any systems for which Java virtual machine is ported. That means you can write a Java program once and run it on Windows, Mac, Linux

## 1.1 Difference between Procedure Oriented Programming and object oriented programming

| Sr. No. | Procedure Oriented Programming | object oriented programming |
|---|---|---|
| 1. | Problem is divided into parts called functions. | Problem is divided into parts called Objects. |
| 2. | It follows Top Down Approach | It follows Bottom up Approach |
| 3. | Here, emphasis is not only on data but on procedures as well. | In this approach, emphasis is on data rather than procedure. |
| 4. | Data can move freely from function to function. | objects can move and communicate with each other through member functions. |
| 5. | No proper way for hiding of data means less secure | Data hiding is possible. |
| 6. | Operator overloading is not possible. | Operator overloading is possible in form of operator overloading and function overloading. |
| 7. | Examples are:C,VB, FORTRAN etc. | Examples are: C++,JAVA,VB.NET etc. |

## 1.2 Object Oriented Concepts:

1. **Objects and Classes**: Objects are basic unit of OOP. They can represent a person,place or any item that a program may handle.Object is an instance of a class which have data members that are used to perform various functions associated with it.
   Class can be assumed as data type and object as a variable of that data type. Once a class has been defined we can create a number of objects of that class.A class is a collection of objects of similar type .We can say that Red,Blue, Pink are members of class colour. When a program is executed, different objects can interact with each other by sending messages.Foreg. A person and account are objects of banking system .If a person want to know the balance of account then person object may interact with account object by sending a message.

| Person | |
|---|---|
| Name | Acct_no |
| Basic pay | Bank name |
| Salary() | |
| Tax() | |

2. **Data Abstraction**:It is the concept of hiding internal details and elaborate the concept in simple term.for example: a method that multiply two integers.we get the result, the internal processing is hidden from the outer world.There are many ways to achieve abstraction in object oriented programming such as encapsulation and inheritance.
3. **Data Encapsulation**:The wrapping up of data and methods into a single unit is called as Encapsulation. The data is not accessed by the outside world .it can be accessed by only those methods which are wrapped in the same class.The protection of data from direct access is known as data hiding.
4. **Inheritance**:It is a concept in which objects of one class can acquire the properties of another class.It means we can derive a new class from the existing class and we can use data and methods of existing classs into new class. The new class have features of both the classes.Inheritance also provides the idea of reusability. It means we can add some additional features to an existing class without modifying it.
5. **Object reference**: A reference is an address where an object's variables and data are stored.
   When we assign an object to a variable ,we are not actually using objects.Even we are not using copies of the objects. Instead, we are using references to those objects.

```
import java.awt.Point;

class ReferencesTest {
  public static void main(String[] arguments) {
Point p1, p2;
  p1 = new Point(10, 10);
   p2 = p1;

   p1.x = 20;
   p1.y = 20;
System.out.println("Point1: " + p1.x + ", " + p1.y);
System.out.println("Point2: " + p2.x + ", " + p2.y);
 }
}
```

Here is this program's output:

```
Point1: 20, 20

Point2: 20, 20
```

The following takes place in the first part of this program:

- Line 5—Two Point variables are created.
- Line 6—A new Point object is assigned to p1.
- Line 7—The value of p1 is assigned to p2.

Lines 9–12 are the tricky part. The x and y variables of p1 are both set to 200, and then all variables of p1 and p2 are displayed onscreen.
You might expect p1 and p2 to have different values. However, the output shows this is not the case. As you can see, the x and y variables of p2 also were changed, even though nothing in the program explicitly changes them. This happens because line 7 creates a reference from p2 to p1, instead of creating p2 as a new object copied from p1.
p2 is a reference to the same object as p1.Either variable can be used to refer to the object or to change its variables.
If you wanted p1 and p2 to refer to separate objects, separate new Point() statements could be used on lines 6 and 7 to create separate objects, as shown in the following:
P1=new point(10,10)
P2=new point(10,10)

6. **Polymorphism**: It means more than one form.An operation can behave differently in different instances.It depends upon the type of data used in the operation. For e.g. In the operation of addition, if we take two integers then it will add these two integers. And if we add two strings then it will concatenate them.

## 1.3 Introduction of eclipse for developing programs in Java

Eclipse is a well-known and respected Integrated Development Environment (IDE) developed by the Eclipse Foundation. Eclipse is beneficial to programmers because it aids in the development process by providing the following key features:

- An easy to use graphical user interface that navigates through your code hierarchy.
- Syntax highlighting that displays source code in a color code format to improve readability

**Q. Very Short Questions Answers (2 Marks)**

**(a) Define class.**
**A:** A class is a template that contains data fields and methods to describe the behavior of its objects.

**(b) Define object.**
**A:** An object is an instance of class which stores its state in data and behavior via methods

**(c) What is instantiation?**
**A:** The process of creating an object of a class is called instantiation.

**(d) Define data members.**
**A:** The variables defined inside a class are called as data members.

**(e) Define a method.**
**A:** The function / procedure defined inside a class is called a method.

**(f) Define actual arguments.**
**A:** The arguments that appear in the method call are called as actual arguments.

**(g) Define formal arguments. .**
**A:** The arguments that appear in the method definition are called as formal arguments.

**(h) Define polymorphism.**
**A:** The technique of processing a message in more than one form is called polymorphism.

**(i) Define inheritance.**
**A:** The process of creating a new class from an already existing old class is called as inheritance.

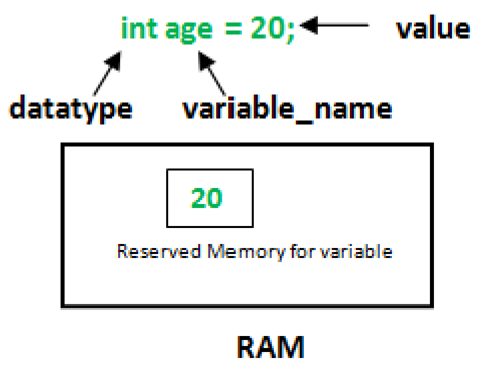
# Unit II-Language constructs

## 2.1.Variables ,types and type declarations:

### i) Variables:
A variable is the name given to a memory location. It is the basic unit of storage in a program.

- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.
- In Java, all the variables must be declared before use.

How to declare variables?
We can declare variables in java as follows:



**datatype**: Type of data that can be stored in this variable.
**variable_name**: Name given to the variable.
**value**: It is the initial value stored in the variable.
**Examples**:

```
float simpleInterest; //Declaring float variable

int time = 10, speed = 20; //Declaring and Initializing integer variable

char var = 'h'; // Declaring and Initializing character variable
```

### ii)Types of variables:

There are three types of variables in Java:

- Local Variables
- Instance Variables
- Static Variables

Let us now learn about each one of these variables in detail.

- **Local Variables**: A variable defined within a block or method or constructor is called local variable.

These variable are created when the block in entered or the function is called and destroyed after exiting from the block or when the call returns from the function.

The scope of these variables exists only within the block in which the variable is declared. i.e. we can access these variable only within that block.

```
publicclassStudentDetails
{
   publicvoidStudentAge()
   {   //local variable age
      intage = 0;
      age = age + 5;
      System.out.println("Student age is : "+ age);
   }

   publicstaticvoidmain(String args[])
   {
      StudentDetailsobj =newstudentDetails();
      obj.StudentAge();
   }
}
```

Output:

```
Student age is : 5
```

In the above program the variable age is local variable to the function StudentAge(). If we use the variable age outside StudentAge() function, the compiler will produce an error as shown in below program.

**Sample Program 2:**

```
publicclassStudentDetails
{
   publicvoidStudentAge()
   {   //local variable age
      intage = 0;
      age = age + 5;
   }

   publicstaticvoidmain(String args[])
   {
      //using local variable age outside it's scope
      System.out.println("Student age is : "+ age);
   }
}
```

**Output:**

```
error: cannot find symbol
```

" + age);

- **Instance Variables**: Instance variables are non-static variables and are declared in a class outside any method, constructor or block.

As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.

Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier then the default access specifier will be used.

**Sample Program:**

```
importjava.io.*;
classMarks
{
  //These variables are instance variables.
  //These variables are in a class and are not inside any function
  intengMarks;
  intmathsMarks;
  intphyMarks;
}

classMarksDemo
{
  publicstaticvoidmain(String args[])
  {   //first object
    Marks obj1 = newMarks();
    obj1.engMarks = 50;
    obj1.mathsMarks = 80;
    obj1.phyMarks = 90;

    //second object
    Marks obj2 = newMarks();
    obj2.engMarks = 80;
    obj2.mathsMarks = 60;
    obj2.phyMarks = 85;

    //displaying marks for first object
    System.out.println("Marks for first object:");
    System.out.println(obj1.engMarks);
    System.out.println(obj1.mathsMarks);
    System.out.println(obj1.phyMarks);

    //displaying marks for second object
    System.out.println("Marks for second object:");
    System.out.println(obj2.engMarks);
```

```
            System.out.println(obj2.mathsMarks);
            System.out.println(obj2.phyMarks);
        }
    }
```

Output:

```
Marks for first object:
50
80
90
Marks for second object:
80
60
85
```

As you can see in the above program the variables, engMarks , mathsMarks , phyMarksare instance variables. In case we have multiple objects as in the above program, each object will have its own copies of instance variables. It is clear from the above output that each object will have its own copy of instance variable.

- **Static Variables**: Static variables are also known as Class variables.

These variables are declared similarly as instance variables, the difference is that static variables are declared using the static keyword within a class outside any method constructor or block.

Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.

Static variables are created at start of program execution and destroyed automatically when execution ends.

To access static variables, we need not to create any object of that class, we can simply access the variable as:

class_name.variable_name;

**Sample Program:**

```
importjava.io.*;
classEmp {

   // static variable salary
   publicstaticdoublesalary;
   publicstaticString name = "Harsh";
}

publicclassEmpDemo
{
    publicstaticvoidmain(String args[]) {

     //accessing static variable without object
     Emp.salary = 1000;
```

```
        System.out.println(Emp.name + "'s average salary:"+ Emp.salary);
      }


    }
```
output:

```
Harsh's average salary:1000.0
```

**Instance variable Vs Static variable**

Each object will have its own copy of instance variable whereas We can only have one copy of a static variable per class irrespective of how many objects we create.

Changes made in an instance variable using one object will not be reflected in other objects as each object has its own copy of instance variable. In case of static, changes will be reflected in other objects as static variables are common to all object of a class.

We can access instance variables through object references and Static Variables can be accessed directly using class name.

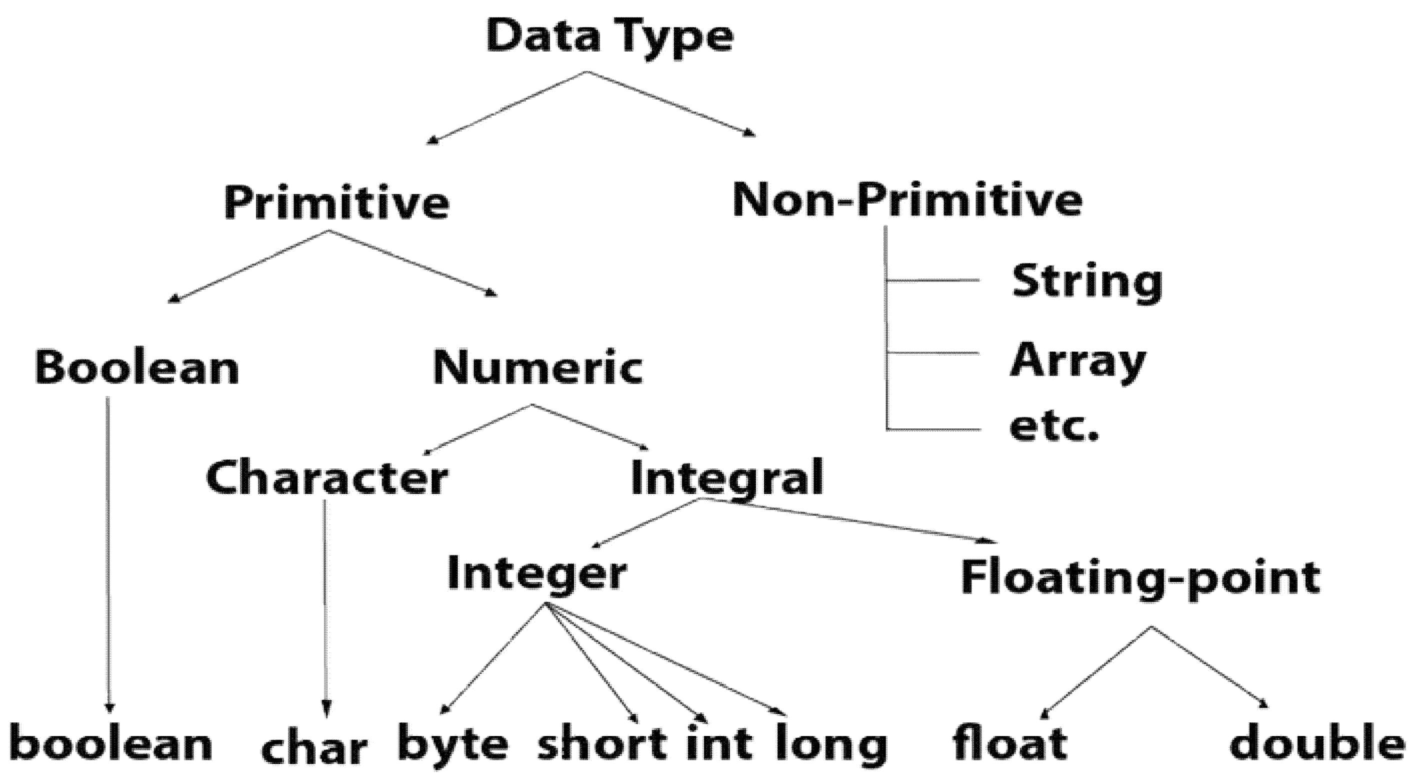
**Syntax for static and instance variables**:

```
class Example
   {
       static int a; //static variable
int b;        //instance variable
   }
```

## 2.2 Data Types in Java

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.



| Data Type | Default Value | Default size |
|---|---|---|
| boolean | false | 1 bit |
| char | '\u0000' | 2 byte |
| byte | 0 | 1 byte |
| short | 0 | 2 byte |
| int | 0 | 4 byte |
| long | 0L | 8 byte |
| float | 0.0f | 4 byte |
| double | 0.0d | 8 byte |

**Boolean Data Type**

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

**Example:** Boolean one = false

**Byte Data Type**

The byte data type is an example of primitive data type. It isan 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

**Example:** byte a = 10, byte b = -20

**Short Data Type**

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

**Example:** short s = 10000, short r = -5000

**Int Data Type**

The int data type is a 32-bit signed two's complement integer. Its value-range lies between -2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is -2,147,483,648and maximum value is 2,147,483,647. 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 = 100000, int b = -200000

**Long Data Type**

The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. 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 = 100000L, long b = -200000L

**Float Data Type**

The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

**Example:** float f1 = 234.5f

**Double Data Type**

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

**Example:** double d1 = 12.3

**Char Data Type**

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

**Example:** char letterA = 'A'

## 2.3. Operators

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups −

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

**The Arithmetic Operators**

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators −

Assume integer variable A holds 10 and variable B holds 20, then −

Show Examples

| Operator | Description | Example |
|---|---|---|

| | | |
|---|---|---|
| + (Addition) | Adds values on either side of the operator. | A + B will give 30 |
| - (Subtraction) | Subtracts right-hand operand from left-hand operand. | A - B will give -10 |
| * (Multiplication) | Multiplies values on either side of the operator. | A * B will give 200 |
| / (Division) | Divides left-hand operand by right-hand operand. | B / A will give 2 |
| % (Modulus) | Divides left-hand operand by right-hand operand and returns remainder. | B % A will give 0 |
| ++ (Increment) | Increases the value of operand by 1. | B++ gives 21 |
| -- (Decrement) | Decreases the value of operand by 1. | B-- gives 19 |

**The Relational Operators**
There are following relational operators supported by Java language.
Assume variable A holds 10 and variable B holds 20, then −
Show Examples

| Operator | Description | Example |
|---|---|---|
| == (equal to) | Checks if the values of two operands are equal or not, if yes then condition becomes true. | (A == B) is not true. |
| != (not equal to) | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. | (A != B) is true. |
| > (greater than) | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | (A > B) is not true. |
| < (less than) | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. | (A < B) is true. |
| >= (greater than or equal to) | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (A >= B) is not true. |
| <= (less than or | Checks if the value of left operand is less than or equal to | (A <= B) is true. |

| | | |
|---|---|---|
| equal to) | the value of right operand, if yes then condition becomes true. | |

**The Bitwise Operators**

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.
Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60 and b = 13; now in binary format they will be as follows −
a = 0011 1100
b = 0000 1101
-----------------
a&b = 0000 1100
a|b = 0011 1101
a^b = 0011 0001
~a  = 1100 0011
The following table lists the bitwise operators −
Assume integer variable A holds 60 and variable B holds 13 then −
Show Examples

| Operator | Description | Example |
|---|---|---|
| & (bitwise and) | Binary AND Operator copies a bit to the result if it exists in both operands. | (A & B) will give 12 which is 0000 1100 |
| \| (bitwise or) | Binary OR Operator copies a bit if it exists in either operand. | (A \| B) will give 61 which is 0011 1101 |
| ^ (bitwise XOR) | Binary XOR Operator copies the bit if it is set in one operand but not both. | (A ^ B) will give 49 which is 0011 0001 |
| ~ (bitwise compliment) | Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. | (~A ) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number. |
| << (left shift) | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. | A << 2 will give 240 which is 1111 0000 |
| >> (right shift) | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. | A >> 2 will give 15 which is 1111 |
| >>> (zero fill right shift) | Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand | A >>>2 will give 15 which is 0000 1111 |

| | | |
|---|---|---|
| | and shifted values are filled up with zeros. | |

**The Logical Operators**

The following table lists the logical operators –

Assume Boolean variables A holds true and variable B holds false, then −

| Operator | Description | Example |
|---|---|---|
| && (logical and) | Called Logical AND operator. If both the operands are non-zero, then the condition becomes true. | (A && B) is false |
| \|\| (logical or) | Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true. | (A \|\| B) is true |
| ! (logical not) | Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. | !(A && B) is true |

**The Assignment Operators**

| Operator | Description | Example |
|---|---|---|
| = | Simple assignment operator. Assigns values from right side operands to left side operand. | C = A + B will assign value of A + B into C |
| += | Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand. | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand. | C -= A is equivalent to C = C – A |
| *= | Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand. | C *= A is equivalent to C = C * A |
| /= | Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand. | C /= A is equivalent to C = C / A |
| %= | Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand. | C %= A is equivalent |

| | | |
|---|---|---|
| | | to C = C % A |
| <<= | Left shift AND assignment operator. | C <<= 2 is same as C = C << 2 |
| >>= | Right shift AND assignment operator. | C >>= 2 is same as C = C >> 2 |
| &= | Bitwise AND assignment operator. | C &= 2 is same as C = C & 2 |
| ^= | bitwise exclusive OR and assignment operator. | C ^= 2 is same as C = C ^ 2 |
| \|= | bitwise inclusive OR and assignment operator. | C \|= 2 is same as C = C \| 2 |

## 2.4. Control flow statements.

1) **Decision-making statements or conditional expressions**: if-then, if-then-else, switch
2) **Looping statements** : for, while, do-while
3) **Branching statements** : break, continue, return

### 1) Decision making statements:

### a)if clause

The if-then statement is the most basic of all the control flow statements. It enables your program to execute a certain section of code depending on the state of variables, or values returned from methods.

```
if(isCar)
{
        System.out.println("I am a Car");
}
```

If isCar test evaluates to false, control jumps to the end of the if-then statement. In Java, the opening and closing braces are optional.But this is applicable only if the block of code to be executed is just a single statement:

```
if(isCar)
        System.out.println("I am a Car");
```

But as a good practice, it is advisable to put the brackets around the statements, even if there is only one statement to execute.This is because, in the begining you may start with one statement and later during the development phase you may add more statements. During this a common mistake would be forgetting to add the newly required braces which compiler cannot catch.

### b)The if-then-else Statement

The if-then-else statement provides a alternate path of execution when an "if" clause evaluates to false.

```
if(isCar)
{
        System.out.println("I am a Car");
}
else
{
        System.out.println("I am a Truck");
}
```

## c)Chaining if Statements

You can chain if-else statements, and create a decision tree sort of thing.

```
if(vehicle="Car")
{
        System.out.println("I am a Car");
}
else if(vehicle="Truck")
{
        System.out.println("I am a Truck");
}
else
{
        System.out.println("I am a Bike");
}
```

In some other example, there could be a chance that it can satisfy more than one expression in the statements. But in Java, once a condition is satisfied and appropriate statements are executed then the remaining conditions are skipped.

## d)Switch statement

Another way to control the flow of your programs with decision-making statements is by a switch statement. A switch statement gives you the option to test for a range of values for your variables. If you think your if-else-if statements is long and complex, you can use switch statement instead.

```
class SwitchDemo {
  public static void main(String[] args) {
    int age = 3;
    String yourAge;
    switch (age) {
      case 1:  System.out.println("You are one yr old");
           break;
      case 2:  System.out.println("You are two yr old");
           break;
      case 3:  System.out.println("You are three yr old");
           break;
      default: System.out.println("You are more than three yr old");
           break;
```

```
        }
    }
}
```

The body of a switch statement is known as a switch block. The switch statement evaluates its expression within the brackets, then executes all statements that follow the matching case label. Again there might be more than one cases being matched but switch will choose the first immidiate matching case ignoring the others.

break statement is necessary.Because without it, statements in switch blocks fall through. Lets take a look at below program:

```
class SwitchDemo {
    public static void main(String[] args) {
        int age = 2;
        String yourAge;
        switch (age) {
            case 1:  System.out.println("You are one yr old");
            case 2:  System.out.println("You are two yr old");
            case 3:  System.out.println("You are three yr old");
            default: System.out.println("You are more than three yr old");
                                                 break;
        }
    }
}
```

Output will be:

```
You are two yr old
You are three yr old
You are more than three yr old
```

If you notice the output, all statementsafter matching case label(in our example case 2) are executed in sequence until a break statement if found.Though technically, the final break is not required because execution flow falls out of the switch statement.

```

: 5
```

## 2.5 looping statements

### a)while loop

The while loop executes a set of statements while a certain conditions is true. In Java there are 2 variations of while loop: while and do-while loop. Here is a simple while loop example:

```
int counter = 0;
while(counter < 5) {
    System.out.println("Inside the while loop, counting: " + counter);
    counter++;
}
```

This above while loop checks whether counter value is less than 5 to check if the statements inside while loop should be executed or not. If the counter value is less than 5, the while loop body is executed one more time else execution continues at the next statement after the while loop.

You can implement an infinite loop using the while statement, like below:

```
while (true){
    // some java statements
}
```

## b) do while loop:

```
int counter = 0;
do {
    System.out.println("Inside the while loop, counting: " + counter);
    counter++;
} while(counter < 5)
```

Notice the condition check is moved to end of while body in do-while construct. Which means statements inside the do while loop body is always executed at least once, and is then executed repeatedly while the while loop condition is true.

This is the main difference between java while and do while loop, that the statements inside the do while loop is always executed at least once before the while loop condition is tested.

## c) For statements

The Java for loop repeats the execution of a set of Java statements. A for loop executes a block of code as long as some condition is true.

```
for (initialization; termination condition; increment/decrement) {
    //java statement(s)
}
```

The initialization expression initializes the loop and is executed only once at the begining when the loop begins. The termination condition is evaluated every loop and if it evaluates to false, the loop is terminated. And lastly increment/decrement expression is executed after each iteration through the loop

Lets print the numbers 1 through 5 to console output.

```
for(int i=1; i<=5; i++){
          System.out.println("Printing using for loop. Count is: " + i);
 }
```

Output will be:

```
Printing using for loop. Count is: 1
Printing using for loop. Count is: 2
Printing using for loop. Count is: 3
Printing using for loop. Count is: 4
Printing using for loop. Count is: 5
```

You can create an infinte loop using for loop also:

```
for(; ; ){
        //java statement(s)
 }
```

There is another version of for loop in java, which is sometimes referred to as the enhanced for statement. This can be used to make your loops more compact and easy to read. It is generally used to iterate through Collections and arrays.

```
String[] people = {"Vivek","Kavya","Aryan"};
```

```
for (String person : people) {
        System.out.println("Hi, I am " + person);
}
```

## iii)Branching statements

### a)The break Statement

The break statement comes in two forms: labeled and unlabeled. Unlabeled is the most commonly used one in java. You would have already seen break statement in switch case. Similarly you can use unlabeled break to terminate a for, while, or do-while loop.

```
for(int i=1; i<=5; i++){
        if(i==3)
        {
                System.out.println("Breaking the for loop.");
                break;
        }
        System.out.println("Printing using for loop. Count is: " + i);
}
```

Output will be:

```
Printing using for loop. Count is: 1
Printing using for loop. Count is: 2
Breaking the for loop.
```

Similarly you can use it in while loop also

```
int counter = 0;
while(counter < 5) {
        if(counter==3)
        {
                System.out.println("Breaking the for loop.");
                break;
        }
    System.out.println("Inside the while loop, counting: " + counter);
    counter++;
}
```

Remember an unlabeled break statement terminates the innermost switch, for, while, or do-while statement.

### b)The continue Statement

The continue statement skips the current iteration of a for, while , or do-while loop. Continue statement is just similar to the break statement in a way that a break statement is used to pass program control immediately after the end of a loop and the continue statement is used to force program control back to the top of a loop. Meaning it skips one execution of a loop’s body.

```
while (someCondition) {
    if (someOtherCondition) {
        continue;
    }
    // Do something
}
```

Lets take an example:

```
int[] nums = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
for (int i = 0; i < nums.length; i++) {
        if (nums[i] % 2 != 0) {
                continue;
        }
        System.out.println(nums[i] + " is even");
}
```

In the above code we check for even numbers, if it is not then we skip that present loop by using `Continue`. So the output will be:

```
0 is even
2 is even
4 is even
6 is even
8 is even
```

## 2.6. Input using Scanner Class  and output statements

Scanner is a class in java.util package used for obtaining the input of the primitive types like int, double etc. and strings. It is the easiest way to read input in a Java program, though not very efficient if you want an input method for scenarios where time is a constraint like in competitive programming.

To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream. We may pass an object of class File if we want to read input from a file.

To read numerical values of a certain data type XYZ, the function to use is nextXYZ(). For example, to read a value of type short, we can use nextShort()

To read strings, we use nextLine().

To read a single character, we use next().charAt(0). next() function returns the next token/word in the input as a string and charAt(0) funtion returns the first character in that string.

Let us look at the code snippet to read data of various data types.

**// Java program to read data of various types using Scanner class**.

```
import java.util.Scanner;
public class ScannerDemo1
{
    public static void main(String[] args)
    {
        // Declare the object and initialize with
        // predefined standard input object
        Scanner sc = new Scanner(System.in);

        // String input
        String name = sc.nextLine();

        // Character input
        char gender = sc.next().charAt(0);

        // Numerical data input
        // byte, short and float can be read
        // using similar-named functions.
        int age = sc.nextInt();
        long mobileNo = sc.nextLong();
        double cgpa = sc.nextDouble();

        // Print the values to check if input was correctly obtained.
        System.out.println("Name: "+name);
        System.out.println("Gender: "+gender);
        System.out.println("Age: "+age);
        System.out.println("Mobile Number: "+mobileNo);
        System.out.println("CGPA: "+cgpa);
    }
}
```

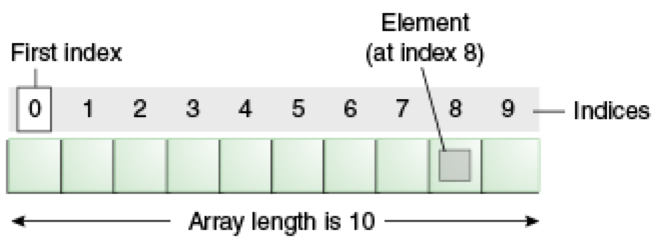
```
Input :
Geek
F
40
9876543210
9.9
Output :
Name: Geek
Gender: F
Age: 40
Mobile Number: 9876543210
CGPA: 9.9
```

## 2.7. Java Array

Normally, an array is a collection of similar type of elements that have a contiguous memory location.

Java array is an object which contains elements of a similar data type. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in java is index-based, the first element of the array is stored at the 0 index.



**Advantages**

- Code Optimization: It makes the code optimized, we can retrieve or sort the data efficiently.
- Random access: We can get any data located at an index position.

**Disadvantages**

- Size Limit: We can store only the fixed size of elements in the array. It doesn't grow its size at runtime.

**Types of Array in java**

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

---

Single Dimensional Array in Java

**Syntax to Declare an Array in Java**

```
dataType[] arr; (or)
dataType []arr; (or)
dataType arr[];
```

**Instantiation of an Array in Java**

```
arrayRefVar=new datatype[size];
```

Example of Java Array

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

```
//Java Program to illustrate how to declare, instantiate, initialize
//and traverse the Java array.
class Testarray{
public static void main(String args[]){
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
//traversing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}}
```

Output:

```
10
20
70
40
50
```

---

Multidimensional Array in Java

In such case, data is stored in row and column based index (also known as matrix form).

**Syntax to Declare Multidimensional Array in Java**

```
dataType[][] arrayRefVar; (or)
dataType [][]arrayRefVar; (or)
dataType arrayRefVar[][]; (or)
dataType []arrayRefVar[];
```

**Example to instantiate Multidimensional Array in Java**

```
int[][] arr=new int[3][3];//3 row and 3 column
```

**Example to initialize Multidimensional Array in Java**

```
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
arr[1][1]=5;
arr[1][2]=6;
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;
```

**Example of Multidimensional Java Array**

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

```
//Java Program to illustrate the use of multidimensional array
class Testarray3{
public static void main(String args[]){
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
//printing 2D array
for(int i=0;i<3;i++){
 for(int j=0;j<3;j++){
   System.out.print(arr[i][j]+" ");
 }
 System.out.println();
}
}}
```

Test it Now

Output:

```
1 2 3
2 4 5
4 4 5
```

**Addition of 2 Matrices in Java**

Let's see a simple example that adds two matrices.

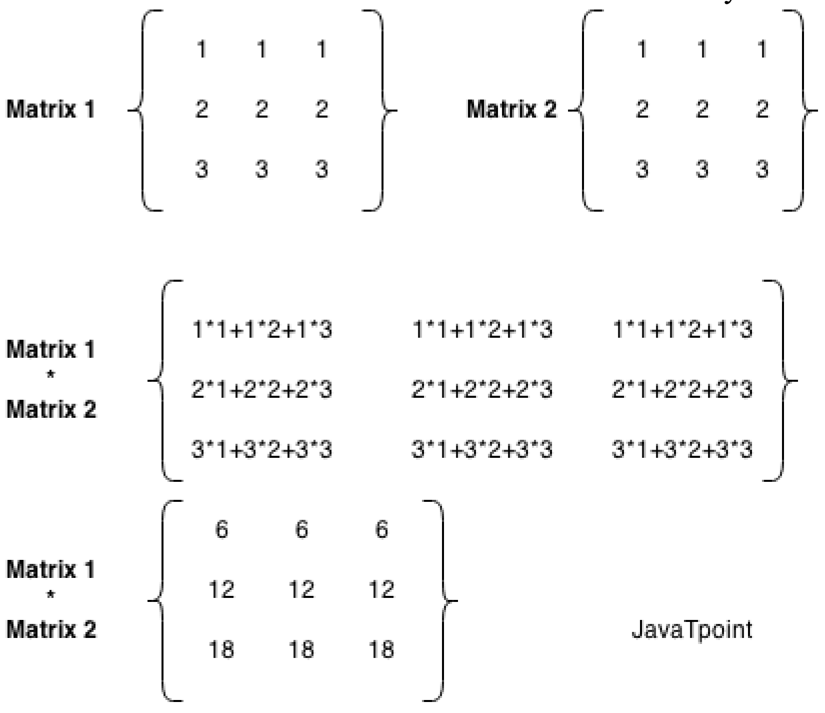
```
//Java Program to demonstrate the addition of two matrices in Java
class Testarray5{
public static void main(String args[]){
//creating two matrices
int a[][]={{1,3,4},{3,4,5}};
int b[][]={{1,3,4},{3,4,5}};

//creating another matrix to store the sum of two matrices
int c[][]=new int[2][3];

//adding and printing addition of 2 matrices
for(int i=0;i<2;i++){
for(int j=0;j<3;j++){
c[i][j]=a[i][j]+b[i][j];
System.out.print(c[i][j]+" ");
}
System.out.println();//new line
}

}}
```
Test it Now

Output:

```
2 6 8
6 8 10
```

**Multiplication of 2 Matrices in Java**

In the case of matrix multiplication, a one-row element of the first matrix is multiplied by all the columns of the second matrix which can be understood by the image given below.



Let's see a simple example to multiply two matrices of 3 rows and 3 columns.

```
//Java Program to multiply two matrices
public class MatrixMultiplicationExample{
public static void main(String args[]){
//creating two matrices
int a[][]={{1,1,1},{2,2,2},{3,3,3}};
int b[][]={{1,1,1},{2,2,2},{3,3,3}};

//creating another matrix to store the multiplication of two matrices
int c[][]=new int[3][3];  //3 rows and 3 columns

//multiplying and printing multiplication of 2 matrices
for(int i=0;i<3;i++){
for(int j=0;j<3;j++){
c[i][j]=0;
for(int k=0;k<3;k++)
{
c[i][j]+=a[i][k]*b[k][j];
}//end of k loop
System.out.print(c[i][j]+" ");  //printing matrix element
}//end of j loop
System.out.println();//new line
}
}}
```

Test it Now

Output:

```
6 6 6
12 12 12
18 18 18
```

## 2.8.Methods in java

A Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.println() method, for example, the system actually executes several statements in order to display a message on the console.

Now you will learn how to create your own methods with or without return values, invoke a method with or without parameters, and apply method abstraction in the program design.

**Creating Method**

Considering the following example to explain the syntax of a method −

Syntax

```
public static int methodName(int a, int b) {
   // body
}
```

Here,

- public static − modifier
- int − return type
- methodName − name of the method
- a, b − formal parameters
- int a, int b − list of parameters

Method definition consists of a method header and a method body. The same is shown in the following syntax −

**Syntax**

```
modifier returnType nameOfMethod (Parameter List) {
   // method body
}
```

The syntax shown above includes −

- modifier − It defines the access type of the method and it is optional to use.
- returnType − Method may return a value.

- nameOfMethod − This is the method name. The method signature consists of the method name and the parameter list.
- Parameter List − The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.
- method body − The method body defines what the method does with the statements.

**Example**

Here is the source code of the above defined method called min(). This method takes two parameters num1 and num2 and returns the maximum between the two −

```
public static int minFunction(int n1, int n2) {

  int min;

  if (n1 > n2)

    min = n2;

  else

    min = n1;

  return min;

}
```

**Method Calling**

For using a method, it should be called. There are two ways in which a method is called i.e., method returns a value or returning nothing (no return value).

The process of method calling is simple. When a program invokes a method, the program control gets transferred to the called method. This called method then returns control to the caller in two conditions, when −

- the return statement is executed.
- it reaches the method ending closing brace.

The methods returning void is considered as call to a statement. Lets consider an example −

```
System.out.println("This is tutorialspoint.com!");
```

The method returning value can be understood by the following example −

```
int result = sum(6, 9);
```

Following is the example to demonstrate how to define a method and how to call it −

Example

```
public class ExampleMinNumber {

    public static void main(String[] args) {

    int a = 11;

    int b = 6;

    int c = minFunction(a, b);

    System.out.println("Minimum Value = " + c);
```

```
  }

  /** returns the minimum of two numbers */

  public static int minFunction(int n1, int n2) {

    int min;

    if (n1 > n2)

      min = n2;

    else

      min = n1;

    return min;

  }

}
```

This will produce the following result −

Output

```
Minimum value = 6
```

The void Keyword

The void keyword allows us to create methods which do not return a value. Here, in the following example we're considering a void method methodRankPoints. This method is a void method, which does not return any value. Call to a void method must be a statement i.e. methodRankPoints(255.7);. It is a Java statement which ends with a semicolon as shown in the following example.

Example

```
public class ExampleVoid {

  public static void main(String[] args) {

    methodRankPoints(255.7);

  }

  public static void methodRankPoints(double points) {

    if (points >= 202.5) {

      System.out.println("Rank:A1");

    }else if (points >= 122.4) {

      System.out.println("Rank:A2");

    }else {

      System.out.println("Rank:A3");

    }

  }

}
```

This will produce the following result −

Output

```
Rank:A1
```

**Passing Parameters by Value**

While working under calling process, arguments is to be passed. These should be in the same order as their respective parameters in the method specification. Parameters can be passed by value or by reference.

Passing Parameters by Value means calling a method with a parameter. Through this, the argument value is passed to the parameter.

Example

The following program shows an example of passing parameter by value. The values of the arguments remains the same even after the method invocation.

```
public class swappingExample {

  public static void main(String[] args) {

    int a = 30;

    int b = 45;

    System.out.println("Before swapping, a = " + a + " and b = " + b);

    // Invoke the swap method

    swapFunction(a, b);

    System.out.println("\n**Now, Before and After swapping values will be same here**:");

    System.out.println("After swapping, a = " + a + " and b is " + b);

  }

  public static void swapFunction(int a, int b) {

    System.out.println("Before swapping(Inside), a = " + a + " b = " + b);

        // Swap n1 with n2

    int c = a;

    a = b;

    b = c;

    System.out.println("After swapping(Inside), a = " + a + " b = " + b);

  }

}
```

This will produce the following result −

Output

```
Before swapping, a = 30 and b = 45

Before swapping(Inside), a = 30 b = 45
```

After swapping(Inside), a = 45 b = 30

**Now, Before and After swapping values will be same here**:

After swapping, a = 30 and b is 45

**Very Short Questions Answers (2 Marks)**

**Q: What is token?**
**A:** The smallest individual entity of a Java program is called a token.

**Q: What is keyword?**
**A:** A keyword is a special word whose meaning is predefined by the Java compiler.

**Q: What is Identifier?**
**A:** Identifiers are the name associated with various programming elements like variables, arrays, classes, interfaces, etc.

**Q: What is Constant?**
**A:** The entities which do not change during the execution of a program are called as constants.

**Q: What is an operator?**
**A:** An operator is the symbol that performs operation on data items to give results.

**Q: List primitive Java types?**
**A:** The eight primitive types are byte, char, short, int, long, float, double, and boolean.

**Q: What is the default value of boolean datatype in Java?**
**A:** The default value of the boolean type is false.

**Q: What is the default value of byte datatype in Java?**
**A:** Default value of byte datatype is 0.

**Q: What is the default value of float and double datatype in Java?**
**A:** Default value of float and double datatype in different as compared to C/C++. For float its 0.0f and for double it's 0.0d

**Q: When a byte datatype is used?**
**A:** This data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an int.

**Q: Variables used in a switch statement can be used with which datatypes?**
**A:** Variables used in a switch statement can only be a byte, short, int, or char.

**Q: Explain the following line used under Java Program:**
**public static void main (String args[ ])**
**A:** The following shows the explanation individually:

- public: it is the access specifier.
- static: it allows main() to be called without instantiating a particular instance of a class.
- void: it tells the compiler that no value is returned by main().
- main(): this method is called at the beginning of a Java program.

String args[ ]: args parameter

# UNIT-III Classes and Objects

## 3.1 Creation and accessing class members:

### a) Class:

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

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

**Syntax of class:**

```
class <class_name>{
   field;
   method;
 }
```

**b) Object:**

An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

- State: represents the data (value) of an object.
- Behavior:represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class. An object in Java is the physical as well as logical entity whereas a class in Java is a logical entity only.

**Syntax of Object:**

ClassName ReferenceVariable = new ClassName();

**Example:**

In this example, we have created a Student class which has two data members id and name. We are creating the object of the Student class by new keyword and printing the object's value.

Here, we are creating a main() method inside the class.

*File: Student.java*

```
//Java Program to illustrate how to define a class and fields
//Defining a Student class.
class Student{
 //defining fields
 int  roll=5;//field or data member or instance variable
 String name="abc";
 //creating main method inside the Student class
 public static void main(String args[]){
  //Creating an object or instance
  Student s1=new Student();//creating an object of Student
  //Printing values of the object
  System.out.println(s1.roll);//accessing member through reference variable
  System.out.println(s1.name);
 }
}
```

Output:

```
5
abc
```

## 3.2 Private Vs Public Vs Protected Vs Default

Java provides a number of access modifiers to set access levels for classes, variables, methods, and constructors. The four access levels are −

- Visible to the package, the default. No modifiers are needed.
- Visible to the class only (private).
- Visible to the world (public).
- Visible to the package and all subclasses (protected).

**a)Default Access Modifier - No Keyword**

Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.

A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.

**Example 1.**

```
//save by A.java
package pack;
class A{
  void msg(){System.out.println("Hello");}
}
```

**Example 2.**

```
//save by B.java
package mypack;
```

```
import pack.*;
class B{
  public static void main(String args[]){
   A obj = new A();//Compile Time Error
   obj.msg();//Compile Time Error
  }
}
```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

**b) Private Access Modifier - Private**

Methods, variables, and constructors that are declared private can only be accessed within the declared class itself.

Private access modifier is the most restrictive access level. Class and interfaces cannot be private.

Variables that are declared private can be accessed outside the class, if public getter methods are present in the class.

Using the private modifier is the main way that an object encapsulates itself and hides data from the outside world.

**Simple example of private access modifier**

In this example, we have created two classes A and Simple. A class contains private data member and private m
private members from outside the class, so there is compile time error.

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}

public class Simple{
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }
```

**c) Public Access Modifier - Public**

A class, method, constructor, interface, etc. declared public can be accessed from any other class. Therefore, fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.

However, if the public class we are trying to access is in a different package, then the public class still needs to be imported. Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.

Example of public access modifier

```
//save by A.java

package pack;
public class A{
public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java

package mypack;
import pack.*;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Output:Hello

**d) Protected Access Modifier - Protected**

Variables, methods, and constructors, which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.

The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.

Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

Example of protected access modifier

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

```
//save by A.java
package pack;
```

```
public class A{
protected void msg(){System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
import pack.*;

class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  }
}
```

```
Output:Hello
```

## 3.3 Constructors

A constructor initializes an object when it is created. It has the same name as its class and is syntactically similar to a method. However, constructors have no explicit return type.

Typically, you will use a constructor to give initial values to the instance variables defined by the class, or to perform any other start-up procedures required to create a fully formed object.

All classes have constructors, whether you define one or not, because Java automatically provides a default constructor that initializes all member variables to zero. However, once you define your own constructor, the default constructor is no longer used.

**Syntax**

```
class ClassName {
   ClassName() {
   }
}
```

**Java allows two types of constructors namely −**

- No argument Constructors
- Parameterized Constructors

**No argument Constructors**

As the name specifies the no argument constructors of Java does not accept any parameters instead, using these constructors the instance variables of a method will be initialized with fixed values for all objects.

Example

```
Public class MyClass {
  Int num;
  MyClass() {
    num = 100;
  }
}
```

You would call constructor to initialize objects as follows

```
public class ConsDemo {
  public static void main(String args[]) {
    MyClass t1 = new MyClass();
    MyClass t2 = new MyClass();
    System.out.println(t1.num + " " + t2.num);
  }
}
```

This would produce the following result

```
100 100
```

**Parameterized Constructors**

Most often, you will need a constructor that accepts one or more parameters. Parameters are added to a constructor in the same way that they are added to a method, just declare them inside the parentheses after the constructor's name.

Example

Here is a simple example that uses a constructor −

```
// A simple constructor.
class MyClass {
  int x;

  // Following is the constructor
  MyClass(int i ) {
    x = i;
  }
}
```

You would call constructor to initialize objects as follows −

```
public class ConsDemo {
  public static void main(String args[]) {
    MyClass t1 = new MyClass( 10 );
```

```
      MyClass t2 = new MyClass( 20 );

      System.out.println(t1.x + " " + t2.x);

   }

}
```

This would produce the following result − 10 20

## 3.4 Object and object Reference:

when we work with objects, it's important to understand references.

A *reference* is an address that indicates where an object's variables and methods are stored.

We  aren't actually using objects when you assign an object to a variable or pass an object to a method as an argument. We aren't even using copies of the objects. Instead, we're using references to those objects.

**Example ReferencesTest.java**

```
 1: import java.awt.Point;

 2:

 3: class ReferencesTest {

 4:   public static void main(String[] arguments) {

 5:     Point pt1, pt2;

 6:     pt1 = new Point(100, 100);

 7:     pt2 = pt1;

 8:

 9:     pt1.x = 200;

10:     pt1.y = 200;

11:     System.out.println("Point1: " + pt1.x + ", " + pt1.y);

12:     System.out.println("Point2: " + pt2.x + ", " + pt2.y);

13:   }

14: }
```

Here is this program's output:

```
Point1: 200, 200

Point2: 200, 200
```

The following takes place in the first part of this program:

- Line 5—Two Point variables are created.
- Line 6—A new Point object is assigned to pt1.
- Line 7—The value of pt1 is assigned to pt2.

Lines 9–12 are the tricky part. The x and y variables of pt1 are both set to 200, and then all variables ofpt1 and pt2 are displayed onscreen.

You might expect pt1 and pt2 to have different values. However, the output shows this is not the case. As you can see, the x and y variables of pt2 also were changed, even though nothing in the program explicitly changes them. This happens because line 7 creates a reference from pt2 to pt1, instead of creating pt2 as a new object copied from pt1.

pt2 is a reference to the same object as pt1; this is shown in figure 1.. Either variable can be used to refer to the object or to change its variables.

If you wanted pt1 and pt2 to refer to separate objects, separate new Point() statements could be used on lines 6 and 7 to create separate objects, as shown in the following:

pt1 = new Point(100, 100);

pt2 = new Point(100, 100);

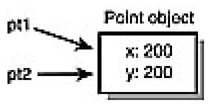


Figure 1 **References to objects.**

References in Java become particularly important when arguments are passed to methods.

**Q.1 Very Short Questions Answers (2 Marks)**

**(a) Define class.**
**A:** A class is a template that contains data fields and methods to describe the behavior of its objects.

**(b) Define object.**
**A:** An object is an instance of class which stores its state in data and behavior via methods

**(c) What is instantiation?**
**A:** The process of creating an object of a class is called instantiation.

**(d) Define data members.**
**A:** The variables defined inside a class are called as data members.

**(e) Define a method.**
**A:** The function / procedure defined inside a class is called a method.

**(f) Define actual arguments.**
**A:** The arguments that appear in the method call are called as actual arguments.

**(g) Define formal arguments. .**
**A:** The arguments that appear in the method definition are called as formal arguments.

# UNIT -IV Inheritance

## 4.1 Definition of inheritance

Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.
**Important terminology:**

- **Super Class:** The class whose features are inherited is known as super class(or a base class or a parent class).
- **Sub Class:** The class that inherits the other class is known as sub class(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.

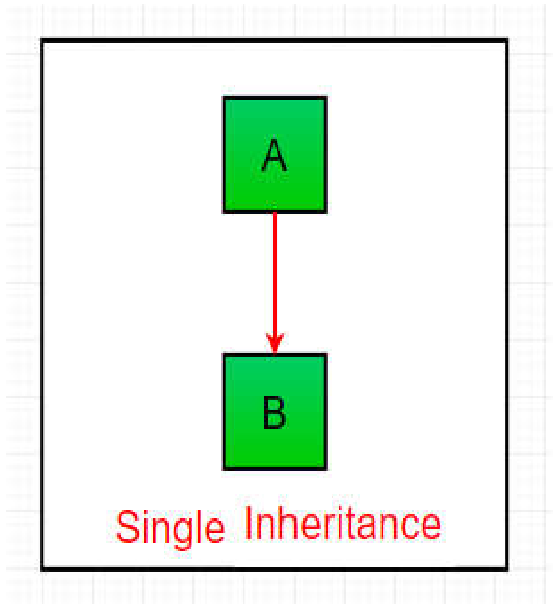
- **Reusability:** Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class
- The keyword used for inheritance is **extends**.
  Syntax :

```
• class derived-class extends base-class
• {
•   //methods and fields
• }
```

**Types of Inheritance in Java**

Below are the different types of inheritance which is supported by Java.

1. **Single Inheritance :** In single inheritance, subclasses inherit the features of one superclass. In image below, the class A serves as a base class for the derived class B.



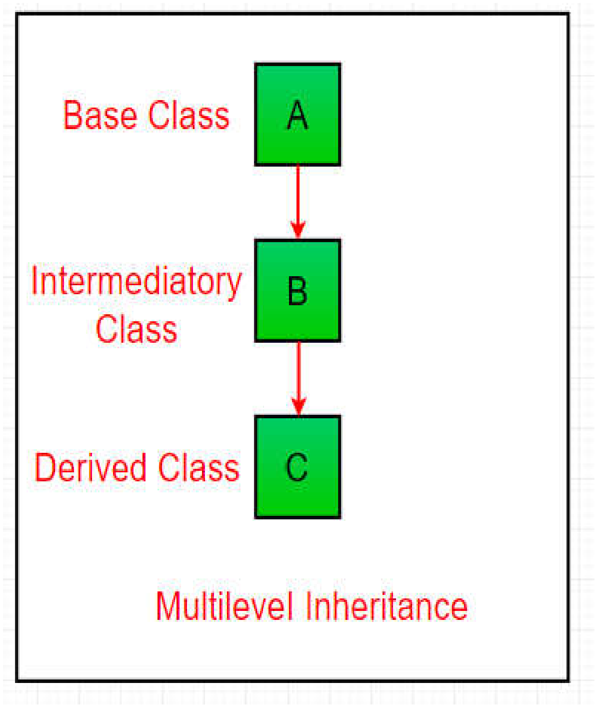
```
//Java program to illustrate the

// concept of single inheritance

import java.util.*;

import java.lang.*;

import java.io.*;


class one

{

   public void print_geek()

   {

      System.out.println("Geeks");

   }

}
```

```
class two extends one
{
    public void print_for()
    {
        System.out.println("for");
    }
}
// Driver class
public class Main
{
    public static void main(String[] args)
    {
        two g = new two();
        g.print_geek();
        g.print_for();
        g.print_geek();
    }
}
```

**Output:**

```
Geeks
for
Geeks
```

2. **Multilevel Inheritance :** In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In below image, the class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C. In Java, a class cannot directly access thegrandparent's members.



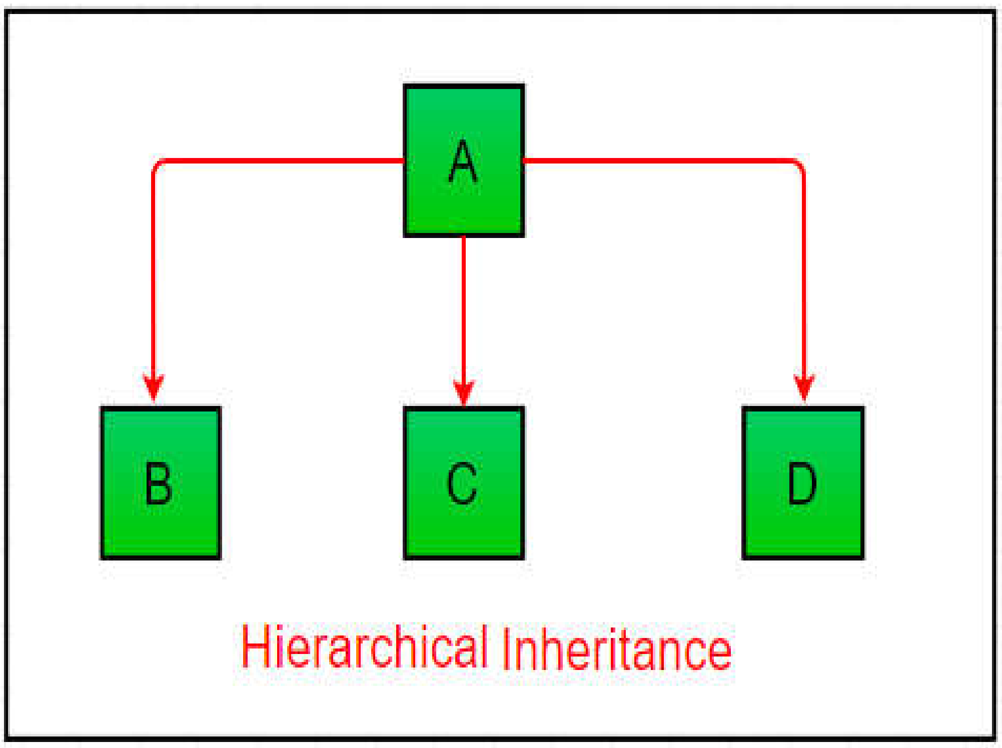
```
// Java program to illustrate the
// concept of Multilevel inheritance
import java.util.*;
import java.lang.*;
import java.io.*;

class one
{
    public void print_geek()
    {
        System.out.println("Geeks");
    }
}

class two extends one
{
    public void print_for()
    {
        System.out.println("for");
    }
}
```

```
class three extends two
{
    public void print_geek()
    {
        System.out.println("Geeks");
    }
}


// Drived class
public class Main
{
    public static void main(String[] args)
    {
        three g = new three();
        g.print_geek();
        g.print_for();
        g.print_geek();
    }
}
```

**Output:**

```
Geeks
for
Geeks
```

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



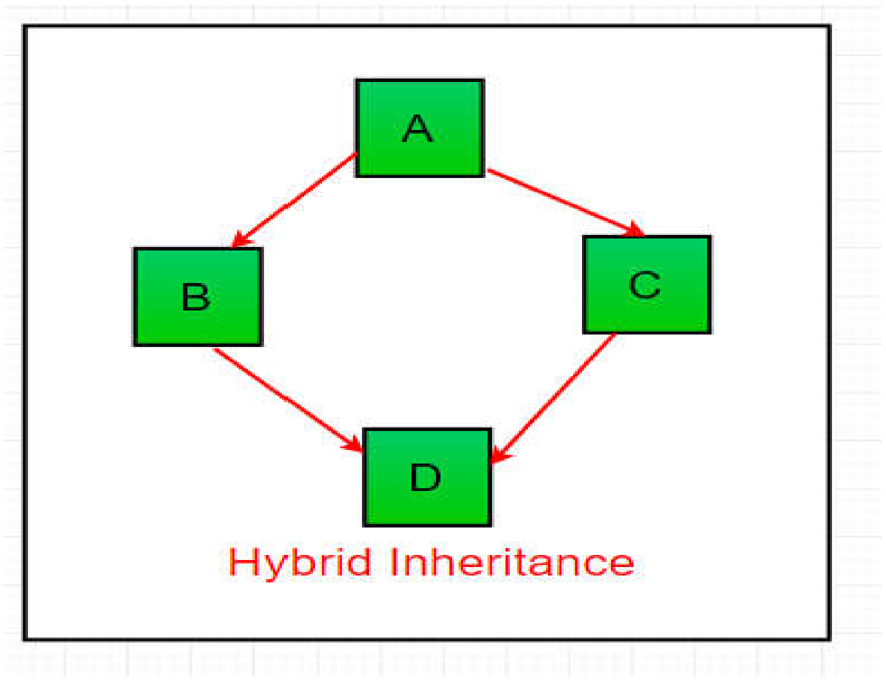
```
// Java program to illustrate the
// concept of Hierarchical inheritance
import java.util.*;
import java.lang.*;
import java.io.*;

class one
{
    public void print_geek()
    {
        System.out.println("Geeks");
    }
}

class two extends one
{
    public void print_for()
    {
        System.out.println("for");
    }
}
 class three extends one
```

```
{
    /*............*/
}


// Drived class
public class Main
{
    public static void main(String[] args)
    {
        three g = new three();
        g.print_geek();
        two t = new two();
        t.print_for();
        g.print_geek();
    }
}
```

Output:

```
Geeks
for
Geeks
```

**4. Hybrid Inheritance**(Through Interfaces) : It is a mix of two or more of the above types of inheritance. Since java doesn't support multiple inheritance with classes, the hybrid inheritance is also not possible with classes. In java, we can achieve hybrid inheritance only through Interfaces.



Important facts about inheritance in Java

- Default superclass: Except Object class, which has no superclass, every class has one and only one direct superclass (single inheritance). In the absence of any other explicit superclass, every class is implicitly a subclass of Object class.
- Superclass can only be one: A superclass can have any number of subclasses. But a subclass can have only one superclass. This is because Java does not support multiple inheritance with classes. Although with interfaces, multiple inheritance is supported by java.
- Inheriting Constructors: A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass.
- Private member inheritance: A subclass does not inherit the private members of its parent class. However, if the superclass has public or protected methods(like getters and setters) for accessing its private fields, these can also be used by the subclass.

**Very Short Questions Answers**

**1. Define inheritance.**
**A:** The process of creating a new class from an already existing old class is called as inheritance.

**2. Define super class.**
**A:** The class that is inherited by other classes is super class, also known as base class or parent class.

**3. Define sub class.**
**A:** The class that inherits the properties of the super class is sub class, also known as derived class or child class.

**4. The ___________ keyword is used to define a new class.**
**A:** extends

**5. Define single inheritance.**
**A:** The process of creating a new class from a single super class is called single inheritance.

**6. Define hierarchical inheritance.**
**A:** The process of creating two or more new classes from a single base is called hierarchical inheritance.

**7. Define multiple inheritance.**
**A:** The process of creating a new class from two or more super classes is called multiple inheritance.

**8. Define multilevel inheritance.**
**A:** Multilevel inheritance is a type of inheritance in which one class is inherited from another class, which in turn is inherited from some other class. .

**9. Does Java support multiple inheritance?**
Ans. Java doesn't support multiple inheritance.

**10. Define reusability.**
**A:** Reusability means taking an existing class and using it in a new programming situation.

**11. Java supports multiple inheritance. (True/False)**
**A:** False

**12. What do you mean by access modifiers?**
**A:** The keywords which control the accessibility of class members are called as access modifiers.

**13. List three access modifiers available in Java?**
**A:** public, private and protected

**14. Define public access modifier?**
**A:** The access specifier which is responsible for making a class member accessible everywhere in a program is called as public access modifier.

**15. Define private access modifier?**
**A:** The access specifier which is responsible for making a class member accessible for all classes in the same package and subclasses in other packages also is called as private access modifier.

**16. Define protected access modifier?**
**A:** The access specifier which is responsible for making a class member accessible within a class and subclasses everywhere in a program is called as protected access modifier.

**17. Define private protected access modifier?**
**A:** The access specifier which is responsible for making a class member accessible to all the subclasses irrespective of the package to which it belong is called as private protected access modifier.

**18. What is default access modifier?**
**A:** If no access modifier is specified, by default, the member of a class is visible only in the same package.

# UNIT -V Polymorphism

## 5.1 Method & Constructor Overloading in Java

### 5.1.1 Method Overloading

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

**Advantage of method overloading**

Method overloading *increases the readability of the program*.

**Different ways to overload the method**

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

**1) Method Overloading: changing no. of arguments**
In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

In this example, we are creating static methods so that we don't need to create instance for calling methods.

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
```

Output:

```
22

33
```

---

**2) Method Overloading: changing data type of arguments**

In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b){return a+b;}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

Output:

```
22

24.9
```

**5.1.2 Constructor Overloading** looks more like a method but without return type. Moreover, the name of the constructor and the class name should be the same. The advantage of constructors over methods is that they are **called implicitly** whenever an object is created. In case of methods, they must be called explicitly. To create an object, the constructor must be called. Constructor gives properties to an object at the time of creation itself (else, it takes some method calls with extra code to do the same job). Programmer uses constructor for initializing variables, instantiating objects and setting colors.

**Default Constructor – No Argument Constructor**

A constructor without parameters is called as "**default constructor**" or "**no-args constructor**". It is called default because if the programmer does not write himself, Java creates one and supplies.

```
public class Demo
{
 public Demo()
 {
  System.out.println("From default constructor");
 }
 public static void main(String args[])
 {
  Demo d1 = new Demo();
  Demo d2 = new Demo();
 }
}
```

output:

```
From default constructor
From default constructor
```

*public Demo()*

"public" is the access specifier and "**Demo()**" is the constructor. Notice, it does not have return type and the name is that of the class name.  Notice, it does not have return type and the name is that of the class name.

*Demo d1 = new Demo();*

In the above statement, **d1** is an object of Demo class. To create the object, the constructor "Demo()" is called. Like this, any number of objects can be created like **d2** and for each object the constructor is called.

**Constructor Overloading**

Just like method overloading, constructors also can be overloaded. Same constructor declared with different parameters in the same class is known as constructor overloading. Compiler differentiates which constructor is to be called depending upon the number of parameters and their sequence of data types.

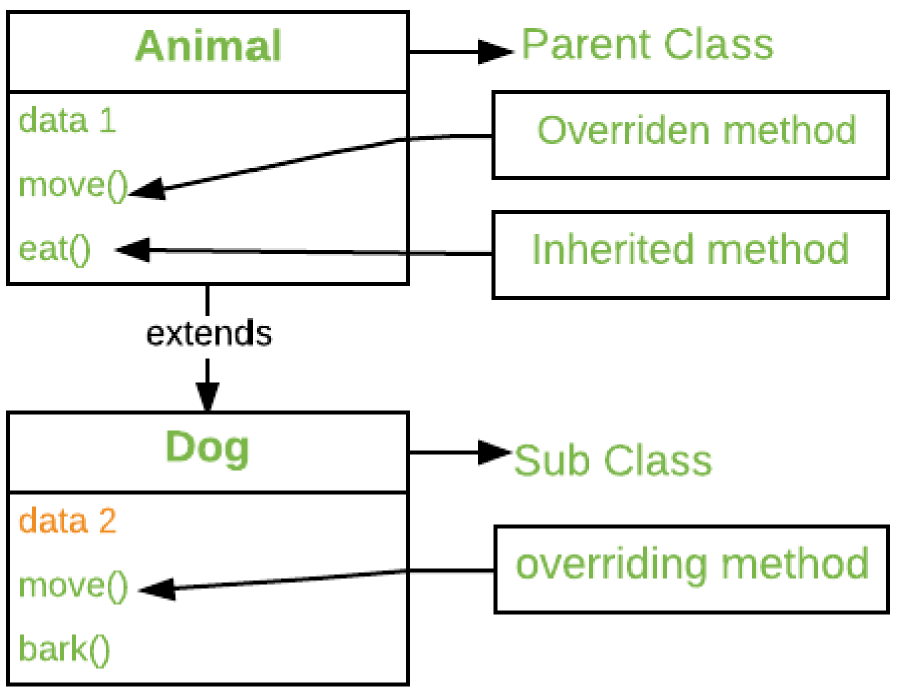
```
public class Perimeter
{
 public Perimeter()                                   // I
 {
  System.out.println("From default");
}
public Perimeter(int x)                              // II
{
  System.out.println("Circle perimeter: " + 2*Math.PI*x);
}
public Perimeter(int x, int y)                        // III
{
  System.out.println("Rectangle perimeter: " +2*(x+y));
}
public static void main(String args[])
{
  Perimeter p1 = new Perimeter();                // I
  Perimeter p2 = new Perimeter(10);              // II
  Perimeter p3 = new Perimeter(10, 20);          // III
}
}
```

Output:

```
From default
Circle perimeter: 62.83185307179586
Rectangle perimeter:60
```

## 5.2 MethodOverriding in Java

In any object-oriented programming language, Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to *override* the method in the super-class.



Method overriding is one of the way by which java achieve Run Time Polymorphism.The version of a method that is executed will be determined by the object that is used to invoke it. If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed.In other words, *it is the type of the object being referred to* (not the type of the reference variable) that determines which version of an overridden method will be executed.

## 5.3 Up-casting and Down casting:

Converting a subclass type to a superclass type is known as up casting.

**Example:**

```
class Super {
  void Sample() {
    System.out.println("method of super class");
  }
}
public class Sub extends Super {
  void Sample() {
    System.out.println("method of sub class");
  }
    public static void main(String args[]) {
    Super obj =(Super) new Sub(); obj.Sample();
  }
}
```

**Down-casting:** Converting a superclass type to a subclass type is known as downcasting.

```
class Super {
  void Sample() {
    System.out.println("method of super class");
  }
}
public class Sub extends Super {
  void Sample() {
    System.out.println("method of sub class");
  }
  public static void main(String args[]) {
    Super obj = new Sub();
    Sub sub = (Sub) obj; sub.Sample();
  }
}
```

# UNIT-VI Abstract class & Interface

**6.1 Abstract class and interface**: Abstract class and interface in Java is similar to interface except that it can contain default method implementation. An abstract class can have an abstract method without body and it can have methods with implementation also.

abstract keyword is used to create a abstract class and method. Abstract class in java can't be instantiated. An abstract class is mostly used to provide a base for subclasses to extend and implement the abstract methods and override or use the implemented methods in abstract class.

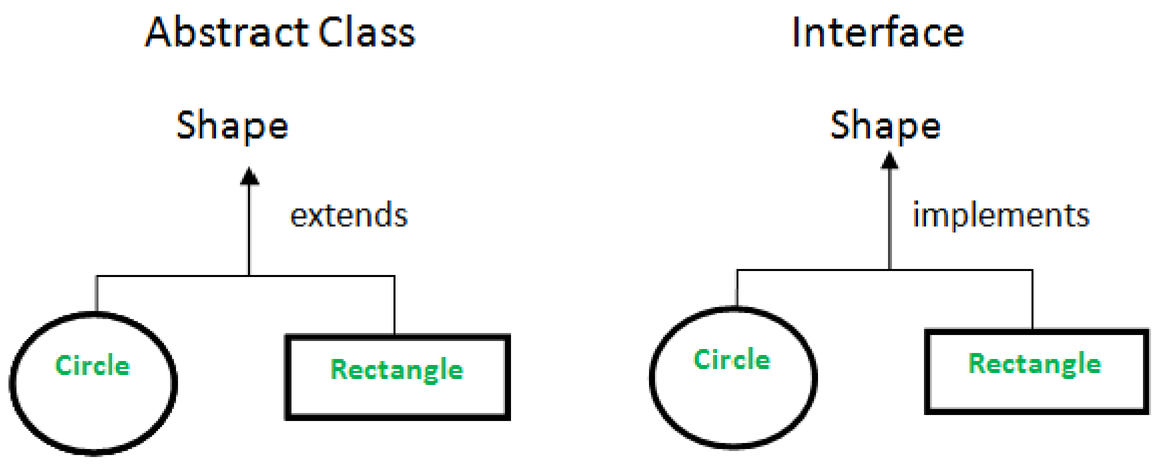
## 6.2 Difference between Abstract Class and Interface in Java

**Abstraction:** Hiding the internal implementation of the feature and only showing the functionality to the users. i.e. what it works (showing), how it works (hiding). Both abstract classand interface are used for abstraction.

**Abstract class vs Interface**

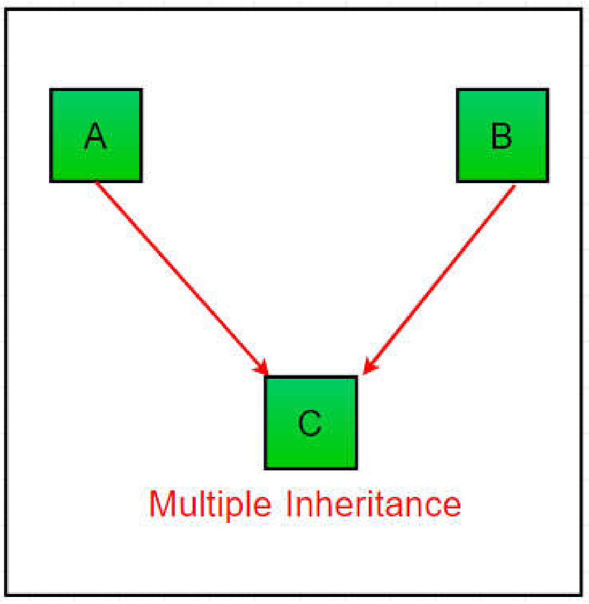
1. **Type of methods:** Interface can have only abstract methods. Abstract class can have abstract and non-abstract methods. From Java 8, it can have default and static methods also.
2. **Final Variables:** Variables declared in a Java interface are by default final. An abstract class may contain non-final variables.
3. **Type of variables:** Abstract class can have final, non-final, static and non-static variables. Interface has only static and final variables.
4. **Implementation:** Abstract class can provide the implementation of interface. Interface can't provide the implementation of abstract class.
5. **Inheritance vs Abstraction:** A Java interface can be implemented using keyword "implements" and abstract class can be extended using keyword "extends".

6. **Multiple implementation:** An interface can extend another Java interface only, an abstract class can extend another Java class and implement multiple Java interfaces.
7. **Accessibility of Data Members:** Members of a Java interface are public by default. A Java abstract class can have class members like private, protected, etc.



## 6.3 Implementation of multiple inheritance through interface

Multiple Inheritance (Through Interfaces) : In Multiple inheritance ,one class can have more than one superclass and inherit features from all parent classes. Please note that Java does not support multiple inheritance with classes. In java, we can achieve multiple inheritance only through Interfaces. In image below, Class C is derived from interface A and B.



filter_none

edit

play_arrow

brightness_4

```
// Java program to illustrate the

// concept of Multiple inheritance

import java.util.*;
```

```
}

import java.lang.*;
import java.io.*;
  interface one
{
    public void print_geek();
}

interface two
{
    public void print_for();
}

interface three extends one,two
{
    public void print_geek();
}
class child implements three
{
    @Override
    public void print_geek() {
        System.out.println("Geeks");
    }

    public void print_for()
    {
        System.out.println("for");
    }
}

// Drived class
public class Main
{
    public static void main(String[] args)
```

```
        {
            child c = new child();
            c.print_geek();
            c.print_for();
            c.print_geek();
        }
    }
```

Output:

```
Geeks
for
Geeks
```

**Q.1 Very Short Questions Answers (2 Marks)**

**(a) What is an Interface?**
**A:** An interface is a collection of only final variables and abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

**(b) How does an interface differ from a class?**
**A:** Unlike a class, an interface contains only final variables and method declarations.

**(c) Give some features of Interface?**
**A:** Some important features of interface are:
- Interface cannot be instantiated
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.

**(d) Define Packages in Java?**
**A:** A Package can be defined as a grouping of related types of classes, interfaces, enumerations and annotations providing access protection and name space management.

**(e) Define API.**
**A:** API (Application Programming Interface) is the standard packages available in Java which contain all the standard classes.

**(f) Why Packages are used?**
**A:** Packages are used to create a separate namespace for group of classes and interfaces. Packages are also used to organize related classes and interfaces into a single API unit and to control accessibility to these classes in Java in-order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations, etc., easier.

**(g) How do you access a particular class or package?**
**A:** An import statement is used to access a particular class or package in a program.

**(h) Which package is imported in Java automatically?**
**A:** java.lang

**(i) How do you create a user-defined package?**
**A:** A package statement is used to create a user-defined package.

**(j) An interface is defined using the keyword _____________.**
**A:** interface

**(k) The variables in an interface are by default ________ and _________.**
**A:** static, final

**(l) The methods are by default ________ in an interface.**
**A:** abstract

**(m) An ___________ statement is used to access a particular class or package in a program.**
**A:** import

**(n) A ____________ keyword is used to create a user-defined package.**
**A:** package .

**(o) The keyword _________ is used by one or more classes to implement the interface in the class definition.**
**A:** implements

**(p) An interface can inherit another interface by using _________ keyword.**
**A:** extends

**(q) API stands for _________ .**
**A:** Application Programming Interface

# UNIT-VII Exception Handling

**7.1 Definition of Exception Handling:** An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at run time, that disrupts the normal flow of the program's instructions**.**

### Error vs Exception

Error: An Error indicates serious problem that a reasonable application should not try to catch.
Exception: Exception indicates conditions that a reasonable application might try to catch.

### Advantage of Exception Handling

The core advantage of exception handling is to maintain the normal flow of the application. An exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

1. statement 1;
2. statement 2;
3. statement 3;
4. statement 4;
5. statement 5;//exception occurs
6. statement 6;
7. statement 7;
8. statement 8;
9. statement 9;
10. statement 10;

Suppose there are 10 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 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

## 7.2 Implementation of keywords try,catch,finally, throw and throws

**1.try**: The try block contains set of statements where an exception can occur.

```
try

{

   // statement(s) that might cause exception

}
```

**2.catch** : Catch block is used to handle the uncertain condition of try block. A try block is always followed by a catch block, which handles the exception that occurs in associated try block.

```
catch

{

  // statement(s) that handle an exception

  // examples, closing a connection, closing

  // file, exiting the process after writing

  // details to a log file.

}
```

**3.throw**: Throw keyword is used to transfer control from try block to catch block.

**4.throws**: Throws keyword is used for exception handling without try & catch block. It specifies the exceptions that a method can throw to the caller and does not handle itself.

**5.finally**: It is executed after catch block. We basically use it to put some common code when there are multiple catch blocks.

**/ Java program to demonstrate working of try,**

**// catch and finally**

```
 class Division {

   public static void main(String[] args)

   {

       int a = 10, b = 5, c = 5, result;

       try {

          result = a / (b - c);

          System.out.println("result" + result);

       }

        catch (ArithmeticException e) {

          System.out.println("Exception caught:Division by zero");

       }

        finally {
            System.out.println("I am in final block");
        }
    }
}
```

**Output:**

```
Exception caught:Division by zero
I am in final block
```

**An example of throws keyword:**

```
// Java program to demonstrate working of throws
class ThrowsExecp {
      // This method throws an exception
    // to be handled
    // by caller or caller
    // of caller and so on.
    static void fun() throws IllegalAccessException
    {
        System.out.println("Inside fun(). ");
        throw new IllegalAccessException("demo");
    }
      // This is a caller function
    public static void main(String args[])
    {
        try {
            fun();
        }
        catch (IllegalAccessException e) {
            System.out.println("caught in main.");
        }
    }
}
```

**Output:**

```
Inside fun().
caught in main.
```

## Very Short Questions Answers [2 Marks]

**(a) What do you mean by Exception handling in Java.**
**A:** When a Java program executes and an abnormal event may disrupt the normal flow of instruction. This is called an exception has occurred and the technique that handles this exception is called as exception handling in Java.

**(b) Explain compile time errors.**
**A:** Errors that occur during the compilation of a program are called as compile time errors.

**(c) Define exception.**
**A:** An abnormal event that occurs during program execution and disrupts the normal flow of instruction.

**(d) What happens if no exception handler is provided?**
**A:** If no exception handler is provided, Java's default exception handler is invoked.

**(e) What is Throwable?**
**A:** The Throwable is a class which is super class of all exception classes.

**(f) What is Exception class?**
**A:** The Exception class is a class that defines exceptions which are thrown by methods of standard Java class library or methods defined in user's program.

**(g) Write the name of five keywords that handle Java exceptions.**
**A:** Java exception handling mechanism mainly uses five keywords – try, catch, throw, throws and finally.

**(h) Define try-catch in exception handling mechanism.**
**A:** The try block contains a set of statements that may throw an exception. The catch block catches and handles the exception thrown by the statements within the try block.

**(i) Define throw keyword.**
**A:** The throw keyword is used to throw an exceptional object explicitly.

**(j) Write down the general syntax of throw statement.**
**A:** The throw is used to to throw an exception. The throw statement requires a single argument: a throwable object. Its syntax is as:

```
throw someThrowableObject;
```

**(k) Define throws keyword.**
**A:** The throws keyword restricts a method to throw a specific exception only.

**(l) What is the role of finally block?**
**A:** The finally block contains those statements which will always be executed irrespective of the occurrence of the exception,

**(m) Is it necessary that each try block must be followed by a catch block?**
**A:** It is not necessary that each try block must be followed by a catch block. It should be followed by either a catch block or a finally block.

**(n) If an exception is encountered and no matching catch block is found, the _____________ takes the control and displays the error message.**
**A:** The default exception handler

**(o) User-defined exceptions can be created simply by defining a subclass of _________ class and using the ________ keyword.**
**A:** Exception, throw

**(p) A program will continue to execute even after an exception has occurred. [True/False]**
**A:** False

**(q) All the classes inherit _________ built-in class.**
**A:** Exception.