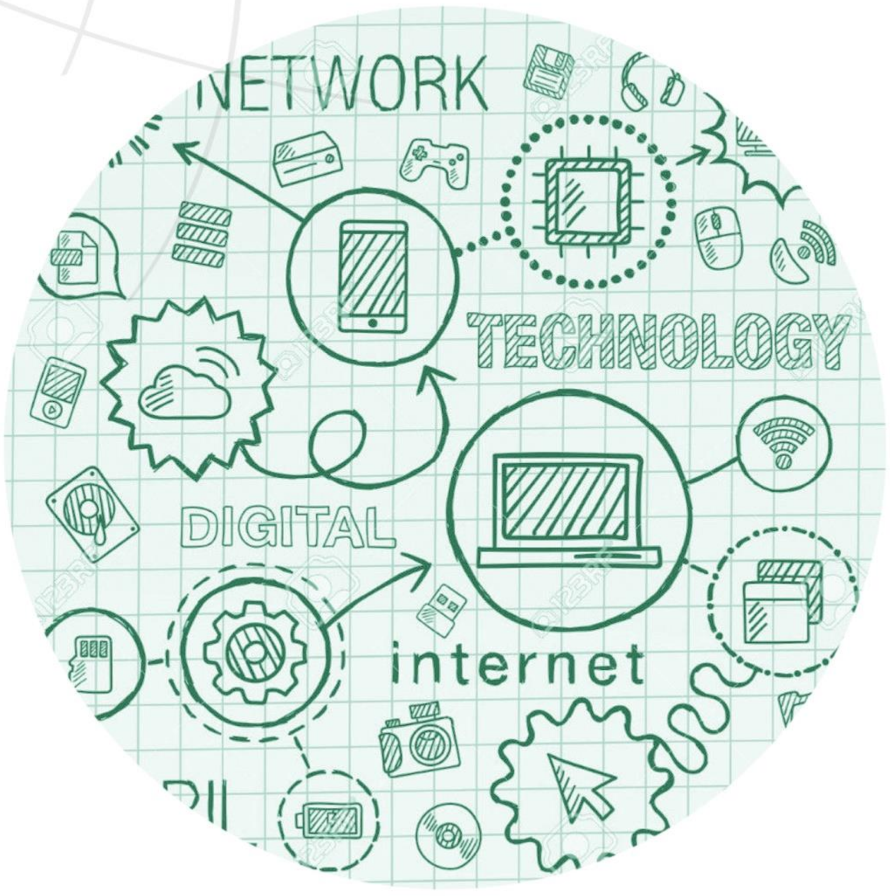
myperfectice

COMPUTER SCIENCE AND ENGINEERING



Quick Guide On Java OOPS Concepts

www.myperfectice.com

Java OOPS Concepts

Java - Inheritance

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

Extends Keyword

extends is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

Syntax

```
class Super {
  .....
  .....
}
class Sub extends Super {
  .....
               .....
}
```

Example

```
class Calculation {
  int z;
  public void addition(int x, int y) {
    z = x + y;
    System.out.println("The sum of the given numbers:"+z);
  }
  public void Subtraction(int x, int y) {
    z = x - y;
    System.out.println("The difference between the given numbers:"+z);
  }
}
public class My_Calculation extends Calculation {
  public void multiplication(int x, int y) {
    z = x * y;
    System.out.println("The product of the given numbers:"+z);
  }
  public static void main(String args[]) {
    int a = 20, b = 10;
    My_Calculation demo = new My_Calculation();
    demo.addition(a, b);
    demo.Subtraction(a, b);
    demo.multiplication(a, b);
  }
}
```

After executing the program, it will produce the following result –

Output

```
The sum of the given numbers: 30
The difference between the given numbers: 10
The product of the given numbers: 200
```

Note – 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.

The super keyword

The **super** keyword is similar to this keyword. Following are the scenarios where the super keyword is used.

- It is used to differentiate the members of superclass from the members of subclass, if they have same names.
- It is used to invoke the superclass constructor from subclass.

Example

```
class Super_class {
  int num = 20;
  // display method of superclass
  public void display() {
    System.out.println("This is the display method of superclass");
  }
}
public class Sub_class extends Super_class {
  int num = 10;

  // display method of sub class
  public void display() {
    System.out.println("This is the display method of subclass");
  }
  public void my_method() {
    // Instantiating subclass
    Sub_class sub = new Sub_class();
    // Invoking the display() method of sub class
    sub.display();
    // Invoking the display() method of superclass
    super.display();
    // printing the value of variable num of subclass
    System.out.println("value of the variable named num in sub class:"+ sub.num);
    // printing the value of variable num of superclass
    System.out.println("value of the variable named num in super class:"+ super.num);
  }
  public static void main(String args[]) {
    Sub_class obj = new Sub_class();
    obj.my_method();
  }
}
```

IS-A Relationship

IS-A is a way of saying: This object is a type of that object. Let us see how the extends keyword is used to achieve inheritance.

```
public class Animal {
}
public class Mammal extends Animal {
}
public class Reptile extends Animal {
}
public class Dog extends Mammal {
}
```

Now, based on the above example, in Object-Oriented terms, the following are true:

- Animal is the superclass of Mammal class.
- Animal is the superclass of Reptile class.
- Mammal and Reptile are subclasses of Animal class.
- Dog is the subclass of both Mammal and Animal classes.

Now, if we consider the IS-A relationship, we can say:

- Mammal IS-A Animal
- Reptile IS-A Animal
- Dog IS-A Mammal
- Hence: Dog IS-A Animal as well

The instanceof Keyword

Let us use the **instanceof** operator to check determine whether Mammal is actually an Animal, and dog is actually an Animal.

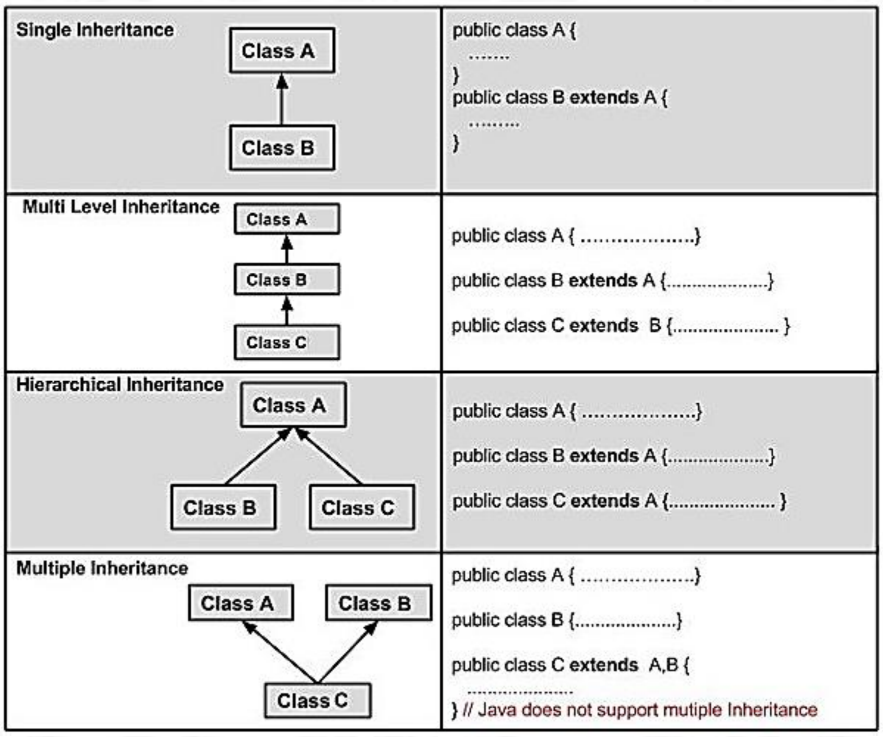
Example

Live Demo

```
interface Animal{}
class Mammal implements Animal{}
public class Dog extends Mammal {
  public static void main(String args[]) {
    Mammal m = new Mammal();
    Dog d = new Dog();
    System.out.println(m instanceof Animal);
    System.out.println(d instanceof Mammal);
    System.out.println(d instanceof Animal);
  }
}
```

Types of Inheritance

There are various types of inheritance as demonstrated below.



```
public class A {
  .......
}
public class B extends A {
  .........
}
```

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

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

```
public class A { ...................}
public class B {...................}
public class C extends  A,B {
  ....................
} // Java does not support mutiple Inheritance
```

A very important fact to remember is that Java does not support multiple inheritance. This means that a class cannot extend more than one class. Therefore following is illegal –

Example

```
public class extends Animal, Mammal{}
```

Java - Polymorphism

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.

Example

Let us look at an example.

```
public class Animal{}
public class Deer extends Animal
Deer d = new Deer();
Animal a = d;
Object o = d;
```

All the reference variables d, a, o refer to the same Deer object in the heap.

Virtual Methods

In this section, we will see how the behavior of overridden methods in Java allows you to take advantage of polymorphism when designing your classes.

Every non-static method in JAVA is by default *virtual method* except final and private methods**.** The methods which cannot be inherited for polymorphic behaviour is not a virtual method.

Abstract class is nothing but the pure virtual method equivalent to C++ in Java.

Java - Abstraction

A class which contains the abstract keyword in its declaration is known as abstract class.

- Abstract classes may or may not contain *abstract methods*, i.e., methods without body (public void get();)
- But, if a class has at least one abstract method, then the class must be declared abstract.
- If a class is declared abstract, it cannot be instantiated.
- To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.
- If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

Example

This section provides you an example of the abstract class. To create an abstract class, just use the **abstract** keyword before the class keyword, in the class declaration.

```
/* File name : Employee.java */
public abstract class Employee {
  private String name;
  private String address;
  private int number;
  public Employee(String name, String address, int number) {
    System.out.println("Constructing an Employee");
    this.name = name;
    this.address = address;
    this.number = number;
  }
  public double computePay() {
```

```
 System.out.println("Inside Employee computePay");
    return 0.0;
  }
  public void mailCheck() {
     System.out.println("Mailing a check to " + this.name + " " + this.address);
  }
  public String toString() {
     return name + " " + address + " " + number;
  }
  public String getName() {
     return name;
  }
  public String getAddress() {
     return address;
  }
  public void setAddress(String newAddress) {
     address = newAddress;
  }
  public int getNumber() {
     return number;
  }
}
```

Now you can try to instantiate the Employee class in the following way –

```
/* File name : AbstractDemo.java */
public class AbstractDemo {

  public static void main(String [] args) {
     /* Following is not allowed and would raise error */
     Employee e = new Employee("George W.", "Houston, TX", 43);
     System.out.println("\n Call mailCheck using Employee reference--");
     e.mailCheck();
  }
}
```

When you compile the above class, it gives you the following error –

```
Employee.java:46: Employee is abstract; cannot be instantiated
     Employee e = new Employee("George W.", "Houston, TX", 43);
               ^
1 error
```

Abstract Methods

If you want a class to contain a particular method but you want the actual implementation of that method to be determined by child classes, you can declare the method in the parent class as an abstract.

- abstract keyword is used to declare the method as abstract.
- You have to place the abstract keyword before the method name in the method declaration.
- An abstract method contains a method signature, but no method body
- Instead of curly braces, an abstract method will have a semoi colon (;) at the end.

Following is an example of the abstract method.

Example

```
public abstract class Employee {
  private String name;
  private String address;
```

```
 private int number;
  public abstract double computePay();
  // Remainder of class definition
}
```

Declaring a method as abstract has two consequence:

- The class containing it must be declared as abstract.
- Any class inheriting the current class must either override the abstract method or declare itself as abstract.

Java - Encapsulation

Encapsulation is one of the four fundamental OOP concepts. The other three are inheritance, polymorphism, and abstraction.

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit.

To achieve encapsulation in Java:

- Declare the variables of a class as private.
- Provide public setter and getter methods to modify and view the variables values.

Example

Following is an example that demonstrates how to achieve Encapsulation in Java:

```
/* File name : EncapTest.java */
public class EncapTest {
  private String name;
  private String idNum;
  private int age;
  public int getAge() {
    return age;
  }
  public String getName() {
    return name;
  }
  public String getIdNum() {
    return idNum;
  }
  public void setAge( int newAge) {
    age = newAge;
  }
  public void setName(String newName) {
    name = newName;
  }
  public void setIdNum( String newId) {
    idNum = newId;
  }
}
```

The public setABC() and getABC() methods are the access points of the instance variables of the EncapTest class

Benefits of Encapsulation

- The fields of a class can be made read-only or write-only.
- A class can have total control over what is stored in its fields.

MyPerfectice Points:

- When a class inherits the properties of an existing class it is called inheritance. Inheritance ensures reusability of code.
- Even if a class has only one abstract method, then the class must be declared abstract.
- The class inheriting an abstract class must override its abstract methods and define them or that class should be declared abstract itself.
- Wrapping of data and functions in a single unit is called data encapsulation.