

- 3.1 Learning Objectives
- 3.2 Inheritance
- 3.3 Subclass
- 3.4 Subclass constructor
- 3.5 Hierarchical inheritance
- 3.6 Overriding methods
- 3.7 Final variables
- 3.8 Final methods
- 3.9 Final classes
- 3.10 Abstract Class
- 3.11 Multiple inheritance
- 3.12 The Object Class
- 3.13 Let us sum up
- 3.14 Check your Progress
- 3.15 Check your Progress: Possible Answers
- 3.16 Further Reading
- 3.17 Assignments
- 3.18 Case Study

---

## 3.1 LEARNING OBJECTIVE

---

After studying this unit student should be able to:

- Understand the inheritance and its types
- Implementation of various types of inheritance in java.
- Use of final keyword with variables, function and class.
- Use of abstract class and abstract function to implement polymorphism.
- Use of function overriding and its implementation
- Understand Object class and its functions.

---

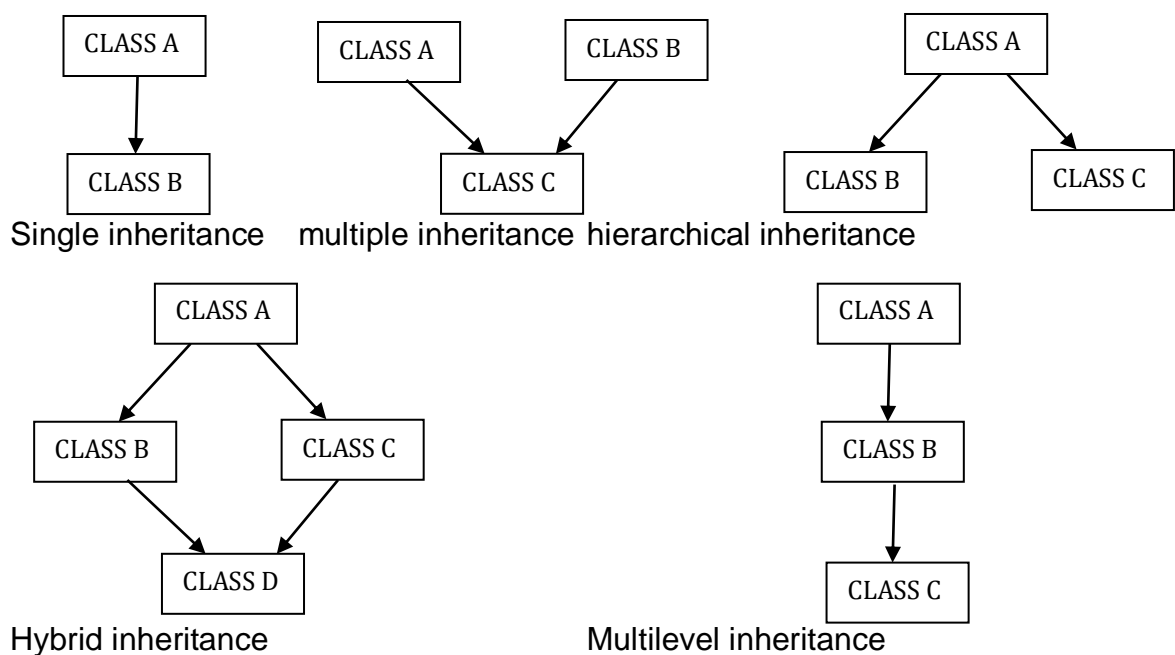
## 3.2 INHERITANCE

---

Using inheritance a class can inherit attributes and methods of the other class. It is like a child inherits the features of parents. It helps us to reuse an already available class, which is called reusability, an important feature of Object oriented programming.

The class which inherits the properties and method of existing class is called subclass or child class and the existing class is called super class or parent class.

The inheritance can be of various types. They are single inheritance, multiple inheritance, multilevel inheritance, hierarchical inheritance and hybrid inheritance.



**Figure-25 Pictorially view of type of inheritance**

- **Single inheritance** : Class B is inherited from Class A.
- **Multiple inheritance** : Class C is inherited from both Class A and Class B.
- **Hierarchical inheritance**: Class B and Class C are inherited from a single Class A.
- **Multilevel inheritance**: Class B inherited from Class A and Class C is inherited from Class B. here class C has properties and methods of Class A also through Class B.
- **Hybrid inheritance**: it is combination of two inheritance that are hierarchical and multiple inheritance

In java we can implement single inheritance, hierarchical inheritance and multilevel inheritance using class. For implementation of multiple and hybrid inheritance in java interfaces are used.

---

### 3.3 SUB CLASS

---

In java for implementing inheritance extends keyword is used. The syntax is as below,

```
class X
{
}
class Y extends X
{
}
}
```

Here X is a super class or parent class and Y is a sub class or child class. Class Y inherits class X.

#### Example,

```
class A
{
    int a;
    A() {a = 0; }
    A( int x ) { a = x; }
    void printA() { System.out.println(" a = " + a); }
}
class B extends A
{
    int b;
    B() { a = 0; b = 0; }
    B(int x, int y) { a = x; b = y; }
```

```

    void printB()
    {
        printA();
        System.out.println(" b = " + b);
    }
}

```

The above example shows single inheritance.

---

## 3.4 SUBCLASS CONSTRUCTOR

---

In above example, the class B has its own constructor in which it initializes the value of parameters of both class B (child) as well as class A (parent). We can also call constructor of parent class in child class for that super keyword is used. The super keyword is used to store reference of parent class object in child class. The method and properties of parent class can be accessed using super keyword in child class.

**For example:**

```

class A
{
    int a;
    A() {a = 0; }
    A( int x ) { a = x; }
    void printA() { System.out.println(" a = " + a); }
}
class B extends A
{
    int b;
    B() {super(); b = 0; }
    B(int x, int y) { super(x); b = y; }
    void printB()
    {
        super.printA();
        System.out.println(" b = " + b);
    }
}

```

**Example:**

```

class Person
{
    String name;
    String address;
    int phno;
}

```

```

Person(){ name = ""; address = ""; phno = 0;}
Person(String n, String a, int p){ name = n; address = a; phno = p;}
void printP()
{
System.out.println("Name : " + name);
System.out.println("Address : " + address);
System.out.println("Phone Number : " + phno);
}
}

class Student extends Person
{
    int rollNumber;
    String course;
    Student(){ super(); rollNumber = 0; course = "";}
    Student(String n, String a, int p,int r, String c)
    {
        super(n,a,p);
        rollNumber = r;
        course = c;
    }
    void printS()
    {
        printP();
        System.out.println("Roll number : " + rollNumber);
        System.out.println("Course : " + course);
    }
}

public class ExSimple
{
    public static void main(String args[])
    {
        Student s1=new Student("Aryan","Surat",34567890,12,"Computer");
        s1.printS();
    }
}

```

```

C:\ajava\oopj>java ExSimple
Name : Aryan
Address : Surat
Phone Number : 34567890
Roll number : 12
Course : Computer

```

Figure-26 Output of program

---

## 3.5 HIERARCHICAL INHERITANCE

---

This inheritance can be implemented using extends key word in java. In hierarchical inheritance more than one child can be inherited from the same parent class.

For example,

```
class Parent
{
}

class child1 extends Parent
{
}

class child2 extends Parent
{
}
```

Here, class Parent is the super class/parent class, which has two children class Child1 and Child2. Child1 and Child2 are also called sibling as they have same parent.

### **Example:**

```
class A
{
    int a;
    A() {a = 0; }
    A( int x ) { a = x; }
    void printA() { System.out.println(" a = " + a); }
}

class B extends A
{
    int b;
    B() {super(); b = 0; }
    B(int x, int y) { super(x); b = y; }
    void printB()
    {
        super.printA();
    }
}
```

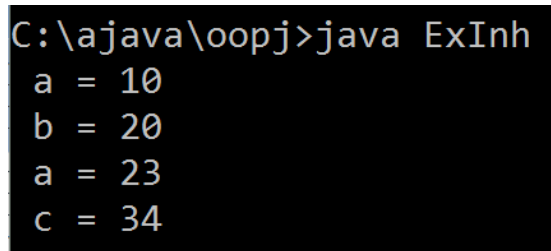
```

        System.out.println(" b = " + b);
    }
}

class C extends A
{
    int c;
    C() {super(); c = 0; }
    C(int x, int z) { super(x); c = z; }
    void printB()
    {
        super.printA();
        System.out.println(" c = " + c);
    }
}

public class ExInh
{
    public static void main(String args[])
    {
        B b1 = new B(10,20);
        C c1 = new C(23,34);
        b1.printB();
        c1.printC();
    }
}

```



```

C:\ajava\oopj>java ExInh
a = 10
b = 20
a = 23
c = 34

```

Figure-27 Output of program

---

## 3.6 OVERRIDING METHODS

---

When a child class inherits a parent class, we can redefine a method of parent class in child class. This concept is called method overriding.

For example,

```

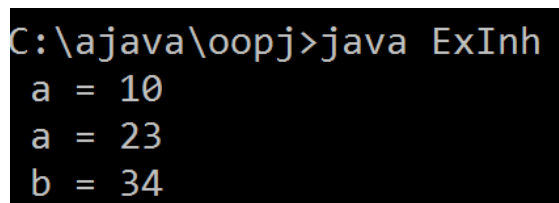
class A
{
    int a;
    A() {a = 0; }
}

```

```

    A( int x ) { a = x; }
    void printData() { System.out.println(" a = " + a); }
}
class B extends A
{
    int b;
    B() {super(); b = 0; }
    B(int x, int y) { super(x); b = y; }
    void printData() //the method of parent class is redefined
    {
        System.out.println(" a = " + a);
        System.out.println(" b = " + b);
    }
}
public class ExInh
{
    public static void main(String args[])
    {
        A a1 = new A(10);
        B b1 = new B(23,34);
        a1. printData();
        b1. printData();
    }
}

```



```

C:\ajava\oopj>java ExInh
a = 10
a = 23
b = 34

```

**Figure-28 Output of program**

In above example printData() method of parent class A is override in child class B. when we call printData method using object of child class, the method of child class will be called. When we call same method using object of parent class, the parent class printData method will be called.

We can also call child class method using reference of parent class. That means when parent class refer parent object it will call parent class's method. And when parent class refers child class object, it will call child class's method.

It is decided at run time which method will be called using reference of parent class. This concept is called dynamic binding or dynamic method dispatch.

For example,



```

class B extends A
{
    int b;
    B() {super(); b = 0; }
    B(int x, int y) { super(x); b = y; }
    void printData() //the method of parent class is redefined
    {
        System.out.println(" a = " + a);
        System.out.println(" b = " + b);
    }
}

```

```

public class ExInh
{
    public static void main(String args[])
    {
        A a1=new A(10);
        B b1=new B(23,34);
        b1.printData();
        a1=b1;
        b1.printData();
    }
}

```

In this example in main method b1.printData() method is called twice. Both time it will run different method. This is due to runtime binding of object with class. It decides at run time which method will be called. This can also be an example of polymorphism.

---

### 3.7 FINAL VARIABLE

---

In java, when a variable is declared as final, it is constant. We have to assign value to this variable while declaring them final. We cannot change value of final variables in our program. Final variables are same as constant variables of C++ and C.

For example,

```
final int N=50;
```

Using final variable in java program

```

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

```

```

    final int x=80;
    int[] a=new int[x];    //we can use x but can not modify it
    x=90; //this gives compilation error as x is constant
    }
}

```

```

C:\ajava\oopj>javac ExInh.java
ExInh.java:8: error: cannot assign a value to final variable x
x=90;    //this gives compilation error as x is constant
^
1 error

```

Figure-29 Output of program

---

### 3.8 FINAL METHOD

---

We can also declare method of a class final. If any method of class defined final it cannot be override/redefine in its child class. Final methods of parent calss can not be overridden in child class. The final methods cannot be changed outside the class.

For example,

```

class A
{
    int a;
    A() {a = 0; }
    A( int x ) { a = x; }
    final void printData() { System.out.println(" a = " + a); } // can not override
}
class B extends A
{
    int b;
    B() {super(); b = 0; }
    B(int x, int y) { super(x); b = y; }
    void printB()
    {
        Super.printData();
        System.out.println(" b = " + b);
    }
}

```

However, if we override the method declared final it gives us a compilation error.

For example,

```

class A
{
    int a;

```

```

    A(){ a=0;}
    A(int x){ a=x;}
    final void printA()
    {
        System.out.println(" a = " + a);
    }
}
class B extends A
{
    int b;
    B(){ super(); b=0;}
    B(int x, int y) { super(x); b=y;}
    int printA(){ return a+b;}
}
public class ExFinal
{
    public static void main(String args[])
    {
        B b1=new B(10,20);
        System.out.println(b1.printA());
    }
}

```

```

C:\ajava\oopj>javac ExFinal.java
ExFinal.java:17: error: printA() in B cannot override printA() in A
int printA(){ return a+b;}
    ^
    overridden method is final
1 error

```

Figure-30 Output of program

---

## 3.9 FINAL CLASS

---

In java class can also be final. The final class restrict them from inheritance. We cannot inherit a class if it is declared as final.

```

final class A
{
}

```

We can not create any class B which inherits class A. For example,

```

final class A
{
    int a;
    A(){ a=0;}
}

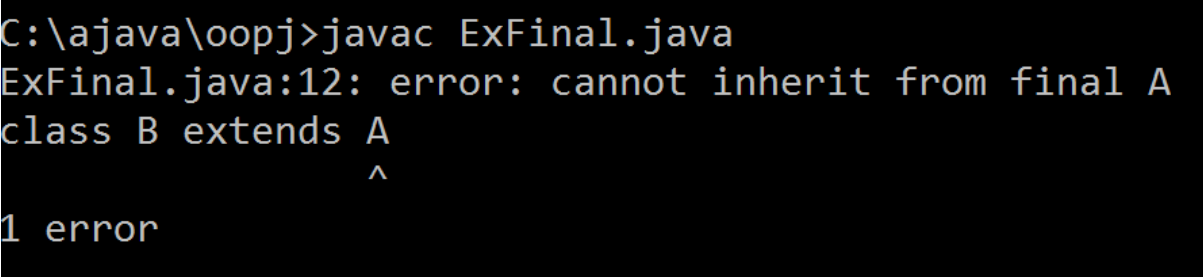
```

```

        A(int x){ a=x;}
        void printA()
        {
            System.out.println(" a = " + a);
        }
    }
class B extends A
{
    int b;
    B(){ super(); b=0;}
    B(int x, int y) { super(x); b=y;}
}
public class ExFinal
{
    public static void main(String args[])
    {
        B b1=new B(10,20);
        b1.printA();
    }
}

```

This program gives compilation error because we try to inherit final class A in this program.



```

C:\ajava\oopj>javac ExFinal.java
ExFinal.java:12: error: cannot inherit from final A
class B extends A
                ^
1 error

```

Figure-31 Output of program

---

### 3.10 ABSTRACT CLASS

---

Abstract class is used to implement abstraction which is an important OOP concept. It is used to create a class with partial implementation. The subclass of abstract class must complete the implementation left in abstract class.

In java, abstract class can be created using abstract keyword. The abstract class is a class which has at least one method declared as an abstract method.

Abstract methods are the methods of a class which are declared in class and have no definition. These methods must be defined in the child classes which are inherited from that abstract class.

We cannot create an object of abstract class. The abstract class can define constructor, non abstract methods, static methods as well as final methods.

Abstract class enforces inheritance. To use abstract class we have to create a class which inherits an abstract class. We can access the method of subclass using the parent class reference.

For example,

```
abstract class A
{
    int a;
    A() {a = 0; }
    A( int x ) { a = x; }
    abstract void printData();
}
class B extends A
{
    int b;
    B() {super(); b = 0; }
    B(int x, int y) { super(x); b = y; }
    void printData()    //definition of abstract method of parent class A
    {
        System.out.println(" a = " + a);
        System.out.println(" b = " + b);
    }
}
```

**Example:**

```
abstract class Shape
{
    double ar;
    double peri;
    Shape()
    {
        ar = 0.0;
        peri = 0.0;
    }
    final double PI=3.14;
    abstract void area();
    abstract void perimeter();
    void printArea()
    {
        System.out.println("Area : " + ar);
    }
    void printPerimeter()
    {
        System.out.println("Perimeter : " + peri);
    }
}
```

```

    }
}

class Circle extends Shape
{
    int r;
    Circle(){ r = 0; }
    Circle(int r){ this.r = r; }
    void area()
    {
        ar = PI*r*r;
    }
    void perimeter()
    {
        peri = 2*PI*r;
    }
}

class Square extends Shape
{
    int s;
    Square(){ s = 0; }
    Square(int s){ this.s = s; }
    void area()
    {
        ar = s * s;
    }
    void perimeter()
    {
        peri = 4 * s;
    }
}

public class ExInh1
{
    public static void main(String args[])
    {
        Shape c1=new Circle(2);
        c1.area();
        c1.printArea();
        c1.perimeter();
        c1.printPerimeter();
        c1=new Square(2);
        c1.area();
        c1.printArea();
        c1.perimeter();
        c1.printPerimeter();
    }
}

```

```
}  
}
```

```
C:\ajava\oopj>java ExInh1  
Area : 12.56  
Perimeter : 12.56  
Area : 4.0  
Perimeter : 8.0
```

**Figure-32 Output of program**

In the above example in main method, we are using reference of Shape class to call methods of child class. When reference of Shape (c1) refers to circle object(line 1) it calls method of Circle class. Same reference can also be used to call the method of Square class. You can see in the main method, line number 2,3,4,5 and line 7,8,9,10 are same in syntax but the line 2,3,4 and 5 calls method of Circle class where as late four lines calls methods of square class. Thus same line code can be executed differently which is an implementation of polymorphism concept of OOP.

---

### **3.11 MULTIPLE INHERITANCE**

---

The multiple inheritance cannot be implemented in java using class. We have to use interface. For creating interface we need to use interface keyword. Interface are created with declaration of methods and constant variables in it. All the methods of interface are either abstract or final. All variables in interface are final and static. We cannot create an instance of interface. We need to implement it in its child class. Interface can extend other interface. A class can implement one or more interface using implement keyword. By default, all the method in interface are abstract and all the variables are final and static. The method in interface must be declared public.

For example multiple inheritance can be implemented as below,

```
interface A  
{  
    int x = 5;  
    public void getData();  
    public void printData();  
}
```

```

interface B
{
    int y = 2;
    public void getD();
    public void printD();
}

class C implements A,B
{
    int [] data;
    C () { int [] data = new int[ x + y ]; }
    public void getData()
    {
        for( int i = 0; i < x ; i++)
            data[i] = 10 * ( i + 1 );
    }
    public void printData()
    {
        for( int i = 0; i < x ; i++)
            System.out.println( data[i] );
    }
    public void getD()
    {
        for( int i = x; i < x+y ; i++)
            data[i] = 10 * ( i + 1 );
    }
    public void printD()
    {
        for( int i = x; i < x+y ; i++)
            System.out.println( data[i] );
    }
}

```

### Example:

```

interface Shape
{
    double PI=3.14;
    public double area();
    public double perimeter();
    public void printData();
}

class Circle implements Shape
{
    int r;
    Circle(){ r = 0; }
    Circle(int r){ this.r = r; }
}

```



```

    public double area()
    {
        return PI*r*r;
    }
    public double perimeter()
    {
        return 2*PI*r;
    }
    public void printData()
    {
        System.out.println("Area : " + area());
        System.out.println("Perimeter : " + perimeter());
    }
}

class Square implements Shape
{
    int s;
    Square(){ s = 0; }
    Square(int s){ this.s = s; }
    public double area()
    {
        return s*s*1.0;
    }
    public double perimeter()
    {
        return 4.0*s;
    }
    public void printData()
    {
        System.out.println("Area : " + area());
        System.out.println("Perimeter : " + perimeter());
    }
}

public class ExInf
{
    public static void main(String args[])
    {
        Shape c1=new Circle(5);
        c1.printData();
    }
}

```

```

C:\ajava\oopj>java ExInf
Area : 78.5
Perimeter : 31.400000000000002

```

Figure-33 Output of program

---

## 3.12 THE OBJECT CLASS

---

Object class is available in java.lang package in java. Any class created in java, automatically derived from Object class. Hence methods of Object class available in all classes of java. The Object class is root of all class.

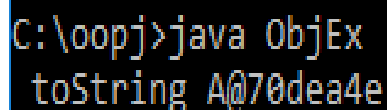
Some of the methods of Object class:

- **String toString():**

It converts an object into String. It returns a string consists of name of class, '@' and hashCode of the object. We can customize the output of toString() function by overriding it in our class.

Example,

```
class A
{
int a;
A() { a = 0; }
A( int x ) { a = x; }
}
public class ObjEx
{
public static void main(String args[])
{
A x1 = new A( 5 );
System.out.println( " toString " + x1.toString() );
}
}
```



```
C:\oopj>java ObjEx
toString A@70dea4e
```

Figure-34 Output of program

- **Example of overriding toString()**

```
class A
{
int a;
A() { a = 0; }
A( int x ) { a = x; }
public String toString()
```

```

    {
    return " Object of Class A ";
    }
    }
    public class ObjEx
    {
    public static void main(String args[])
    {
    A x1 = new A( 5 );
    System.out.println( " toString " + x1.toString() );
    }
    }

```

```

C:\oopj>java ObjEx
toString Object of Class A

```

Figure-35 Output of program

- **int hashCode():**

it is used to get hashvalue of object which can be used to search for object. Hashcode is unique for each object.

Example,

```

public class ObjEx
{
public static void main(String args[])
{
String s = new String(" Hello ");
Class c = s.getClass();
System.out.println ( " class of object s is :" + c.getName() );

}
}

```

```

C:\oopj>java ObjEx
Hashcode of object s is :493245902

```

Figure-36 Output of program

- **boolean equals(Object obj):**

compare object obj with this object and returns true if equal else false.

For example,

```

class A
{
int a;

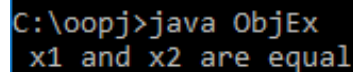
```

```

A() { a = 0; }
A( int x ) { a = x; }
}
public class ObjEx
{
public static void main(String args[])
{
A x1 = new A( 5 );
A x2 = x1;
if ( x1.equals(x2))
System.out.println( " x1 and x2 are equal" );
else
System.out.println( " x1 and x2 are not equal" );

}
}

```



```

C:\oopj>java ObjEx
x1 and x2 are equal

```

Figure-37 Output of program

- **Class getClass():**

Returns Class object of this object. The Class object has method name getName() which returns name of class which of the type of this object.

For example:

```

A a = new A(15);
Class c = a.getClass();
// print A as class name
System.out.println( " class of object s is :" + c.getName());

```

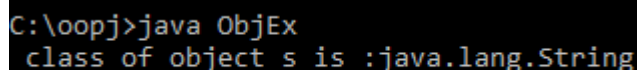
Example,

```

public class ObjEx
{
public static void main(String args[])
{
String s = new String(" Hello ");
Class c = s.getClass();
System.out.println( " class of object s is :" + c.getName());

}
}

```



```

C:\oopj>java ObjEx
class of object s is :java.lang.String

```

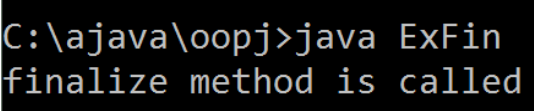
Figure-38 Output of program

- **finalize():**

This method is called before call of garbage collector in java. this method is called for each object once.

for example,

```
class A
{
int a;
A(){ a = 0; }
A(int x) { a = x; }
protected void finalize()
{
System.out.println(" finalize method is called " );
}
}
public class ExFin
{
public static void main(String args[])
{
A a1=new A(4);
a1 = null;
System.gc();
}
}
```



```
C:\ajava\oopj>java ExFin
finalize method is called
```

Figure-39 Output of program

- **Object clone():**

This method returns an object that is same as this object. For using clone() function the class must implements Cloneable interface and implements a function name clone in it. Also the method which calling clone function must handle the CloneNotSupportedException. We will discuss more about Exception in unit 6 of this book.

For example,

```
class A implements Cloneable
{
int a;
A() { a = 0; }
A( int x ) { a = x; }
```

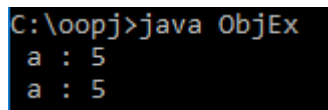
```

    public Object clone() throws CloneNotSupportedException
    {
        return super.clone();
    }
    public void printData()
    {
        System.out.println(" a : " + a );
    }

}
public class ObjEx
{
    public static void main(String args[]) throws CloneNotSupportedException
    {
        A x1 = new A( 5 );
        A x2 = ( A ) x1.clone();
        x1.printData();
        x2.printData();

    }
}

```



```

C:\oopj>java ObjEx
a : 5
a : 5

```

Figure-40 Output of program

---

### 3.13 LET US SUM UP

---

**Inheritance:** it is an important object oriented programming features in which we can reuse existing class by adding new features and methods in it.

**Subclass:** in inheritance the class which derives the existing class is called subclass

**Super class:** in inheritance class from which a subclass is derived is called super class

**super keyword:** in child class, super is a reference to object of parent class. We can access parent class properties and method using super key word.

**method overriding:** The function of parent class and be redefined in child class, this is called method overriding.

**final variable:** it is used to define constant variables in java.

**final function:** it is a function of parent class which cannot be overridden in child class.

**final class:** the final class cannot be inherited. We cannot create a child of final class.

**abstract class:** Abstract class is a class which has at least one abstract method declared in it. This class cannot be instantiated. We have to inherit this class to use it.

**abstract function:** these methods have only signature in class. The subclass which inherits the parent class must define all the abstract methods in it.

**Interface:** it must have only static final variables and abstract and final methods in it. It supports multiple inheritance in java.

**Object class:** Object class is available in java.lang package library. It is the parent of each class created in java program

---

## 3.14 CHECK YOUR PROGRESS

---

➤ True-False with reason

1. extends keyword is used to inherit a class.
2. implements keyword is used to inherit a class.
3. abstract class cannot be inherited.
4. Final class cannot be inherited.
5. Final method cannot be overloaded.
6. Interface and class are same.
7. Object class is parent of each class created in java.
8. Interface can have at least one abstract function in it.
9. All the variable declared in interface are final and static.
10. All variables declared in abstract class are final.
11. Method overriding is writing more than one method in a class with same name.
12. Super is a reference to object which is accessing the variable or method of class.
13. We can call constructor of parent class using super keyword.
14. Multilevel inheritance is not supported in java using class.
15. Multiple inheritance is possible using interface.

➤ Compare the followings

1. Class and interface
2. Abstract class and interface
3. Method overloading and method overriding
4. Constructor and finalize method
5. Final class and abstract class.
6. Final variable and static variable
7. Final method and abstract method

➤ MCQ.

1) In following Java Program which show method is called in main()?

```
class Base {
    public void show() {
        System.out.println("Base::show() called");
    }
}

class Derived extends Base {
    public void show() {
        System.out.println("Derived::show() called");
    }
}

public class Main {
    public static void main(String[] args) {
        Base b = new Derived();
        b.show();
    }
}
```

(a) show method of Derived Class

(b) show method of Base Class

2) In following Java Program which show method is called in main()?



```

class Base {
    final public void show() {
        System.out.println("Base::show() called");
    }
}

```

```

class Derived extends Base {
    public void show() {
        System.out.println("Derived::show() called");
    }
}

```

```

class Main {
    public static void main(String[] args) {
        Base b = new Derived();
        b.show();
    }
}

```

- (a) show method of Derived      (b) show method of Base  
(c) compile time error          (d) run time error

3) . ..... helps to extend the functionality of an existing by adding more methods to the subclass.

- a) Mutual Exclusion              b) Inheritance  
c) Package                        d) Interface

4) An ..... is an incomplete class that requires further specification.

- a) abstract class                b) final class  
c) static class                  d) super class

5) A class can be declared as ..... if you do not want the class to be sub-classed.

- a) abstract                        b) final



d) Computer is a superclass, AppleComputer is a subclass of Computer, and IBMComputer is a subclass of AppleComputer

11) Which of these is correct way of inheriting class A by class B?

a) class B + class A {}

b) class B inherits class A {}

c) class B extends A {}

d) class B extends class A {}

12) What is the output of this program?

```
class A
{
    int i;
    void display()
    {
        System.out.println(i);
    }
}
class B extends A
{
    int j;
    void display()
    {
        System.out.println(j);
    }
}
class inheritance_demo
{
    public static void main(String args[])
    {
        B obj = new B();
        obj.i=1;
        obj.j=2;
        obj.display();
    }
}
```

- a) 0
- b) 1
- c) 2
- d) Compilation Error

13) What is the output of this program?

```
class A
{
    int i;
}
class B extends A
{
    int j;
    void display()
    {
        super.i = j + 1;
        System.out.println(j + " " + i);
    }
}
class inheritance
{
    public static void main(String args[])
    {
        B obj = new B();
        obj.i = 1;
        obj.j = 2;
        obj.display();
    }
}
```

- a) 2 2
- b) 3 3
- c) 2 3
- d) 3 2

14) What is the output of this program?

```

class A
{
    public int i;
    public int j;
    A()
    {
        i = 1;
        j = 2;
    }
}
class B extends A
{
    int a;
    B()
    {
        super();
    }
}
class super_use
{
    public static void main(String args[])
    {
        B obj = new B();
        System.out.println(obj.i + " " + obj.j)
    }
}

```

a) 1 2

b) 2 1

c) Runtime Error

d) Compilation Error

---

### 3.15 CHECK YOUR PROGRESS: POSSIBLE ANSWERS

---

➤ True-False with reason

1. True

2. False. implements keyword is used to inherit an interface.
3. False. abstract class has to be inherited.
4. True
5. False. Final methods cannot be overridden.
6. False. Class does not support multiple inheritance where as interface does.
7. True
8. False. Interface has all abstract functions in it.
9. True
10. False. At least one method in abstract class must be abstract.
11. False. Method overriding is writing a definition of a parent class's method in subclass.
12. False. Super is a reference to object which is accessing the variable or method of parent class
13. True.
14. False. Multilevel inheritance is supported in java using class.
15. True.

➤ Compare the followings

1. Class v/s interface

<b>Class</b>	<b>Interface</b>
Class can have member variables and functions	Interface can have final and static member variables and abstract or final methods.
It does not support multiple inheritance	It supports multiple inheritance
It can be instantiated	It can not directly be instantiated

2. Abstract class v/s Interface

<b>Abstract Class</b>	<b>Interface</b>
It must have at least one abstract method	It has abstract or final methods.

It does not support multiple inheritance	It supports multiple inheritance
All member variables are not final.	All member variables must be final

### 3. Method overloading v/sMethod overriding

<b>Method overloading</b>	<b>Method overriding</b>
Writing method with same name and different arguments in a class	Writing a method which is defined in parent class again in child class with new definition.
Method overloading is not compulsory	Abstract methods must be override

### 4. Constructor v/sFinalize method

<b>Constructor</b>	<b>Finalizemethod</b>
It is a function in a class which has same name as class name.	It is a protected function of Object class which can be called at the end of the program.
It is called when object is created	It is called when object are destroyed by garbage collector.
It is used to initialize the object	It is used to run some code when object is deleted.

### 5. Final class v/sAbstract class.

<b>Final class</b>	<b>Abstract class</b>
The restricts inheritance	They enforce inheritance
The method of this class can not be abstract	At least one method of this class must be abstract.
final keyword is used	abstract keyword is used.

### 6. Final variable v/sStatic variable

<b>Final variable</b>	<b>Static variable</b>
They used to defined constant in java	They used to define class variable in java
They are not shared among all objects of class	They are shared among all objects of class.
They can be accessed using object name	They can be accessed using class name

#### 7. Final method v/s abstract method

<b>Final methods</b>	<b>Static methods</b>
They are the method in parent class which can not be redefine in child class	They are the methods of class which can access only static members of class.
They can be accessed using object name	They can be accessed using class name

#### ➤ MCQ.

- |      |       |       |
|------|-------|-------|
| 1) a | 6) b  | 11) c |
| 2)c  | 7) a  | 12) c |
| 3) b | 8) c  | 13) a |
| 4) a | 9) c  | 14) a |
| 5) b | 10) a |       |

---

### 3.16 FURTHER READING

---

1. Java Inheritance (Subclass and Superclass) - W3Schools [https://www.w3schools.com/java/java\\_inheritance.asp](https://www.w3schools.com/java/java_inheritance.asp)
2. Inheritance in Java OOPs with Example - Guru99 <https://www.guru99.com/java-class-inheritance.html>
3. "Java 2: The Complete Reference" by Herbert Schildt, McGraw Hill Publications.



---

### 3.17 ASSIGNMENTS

---

- Write java program for following
- 1) Create a class to find out the Area and perimeter of rectangle.
  - 2) Create a class quadrilateral and create two methods each for calculating area & perimeter of the quadrilateral with one & two parameters respectively Check number is even or odd.
  - 3) Define a class student with the following specifications:  
Private members of the class:  
Admission Number - An Integer  
Name - string of 20 characters  
Class - Integer  
Roll Number - Integer  
Public members of the class:  
getdata() - To input the data  
showdata() - To display the data  
Write a program to define an array of 10 objects of this class, input the data in this array and then display this list.
  - 4) A class STUDENT has 3 data members:  
Name, Roll Number, Marks of 5 subjects, Stream  
and member functions to input and display data. It also has a function member to assign stream on the basis of the table given below:  

Average Marks	Stream
96% or more	Computer Science
91% - 95%	Electronics
86% - 90%	Mechanical
81% - 85%	Electrical
75% - 80%	Chemical
71% - 75%	Civil

  
Declare a structure STUDENT and define the member functions.  
Write a program to define a structure STUDENT and input the marks of n ( $n \leq 20$ ) students and for each student allot the stream.

- 5) Define a POINT class for two-dimensional points (x, y). Include constructors, a negate() function to transform the point into its negative, a norm() function to return the point's distance from the origin (0,0), and a print() function besides the functions to input and display the coordinates of the point. Use this class in a menu driven program to perform various operations on a point.
- 6) Write a program implement a class 'Complex' of complex numbers. The class should be include member functions to add and subtract two complex numbers.
- 7) Write a Program to implement a sphere class with appropriate members and member function to find the surface area and the volume. (Surface =  $4 \pi r^2$  and Volume =  $\frac{4}{3} \pi r^3$  ).
- 8) Write a program to implement an Account Class with member functions to Compute Interest, Show Balance, Withdraw and Deposit amount from the Account.

---

### 3.18 CASE STUDY

---

File Player.java contains a class that holds information about an athlete: name, team, and uniform number. File ComparePlayers.java contains a skeletal program that uses the Player class to read in information about two baseball players and determine whether or not they are the same player.

1. Fill in the missing code in ComparePlayers so that it reads in two players and prints "Same player" if they are the same, "Different players" if they are different. Use the equals method, which Player inherits from the Object class, to determine whether two players are the same. Are the results what you expect?

2. The problem above is that as defined in the Object class, equals does an address comparison. It says that two objects are the same if they live at the same memory location, that is, if the variables that hold references to them are aliases. The two Player objects in this program are not aliases, so even if they contain exactly the same information they will be "not equal." To make equals compare the actual information in the object, you can override it with a definition specific to the class. It

might make sense to say that two players are "equal" (the same player) if they are on the same team and have the same uniform number. Use this strategy to define an equals method for the Player class. Your method should take a Player object and return true if it is equal to the current object, false otherwise. Test your ComparePlayers program using your modified Player class. It should give the results you would expect.

```
import java.util.Scanner;

public class Player
{
    private String name;
    private String team;
    private int jerseyNumber;

    public void readPlayer()
    { Scanner scan = new Scanner(System.in);
      System.out.print("Name: ");
      name = scan.nextLine();
      System.out.print("Team: ");
      team = scan.nextLine();
      System.out.print("Jersey number: ");
      jerseyNumber = Scan.nextInt();
    }

}

import java.util.Scanner;
public class ComparePlayers
{
    public static void main(String[] args)
    {
        Player player1 = new Player();
        Player player2 = new Player();
```

```
Scanner scan = new Scanner();  
    // Read player 1  
    // Read player 2  
    // compare player1 and player2  
  
    } }
```