UNIT-2

Inheritance

Passing on properties(attributes,methods) of one class to another class

Example

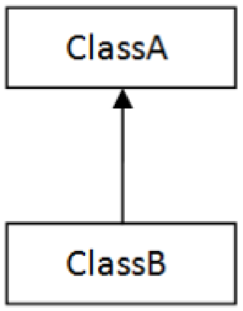
Each child(sub class) inherits few properties from his parents(super class) like

- attributes(DATA)
  - Caste
  - surname
  - Disease
- methods(behaviour)
  - The Way he walks
  - The Way he smiles
  - The Way he talks

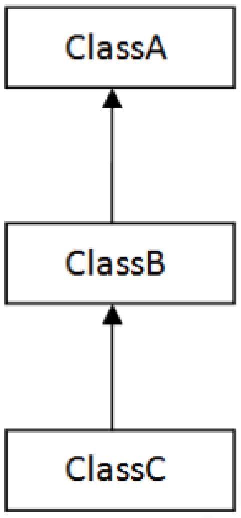
In java, one class can inherit variables and functions (properties) of another class

----

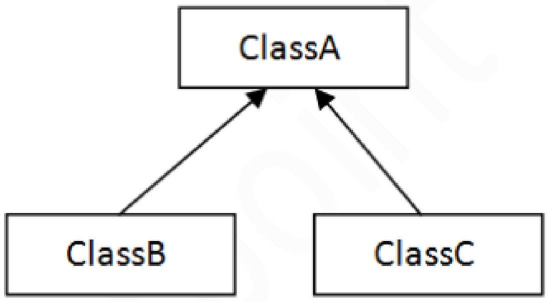
Types of inheritance



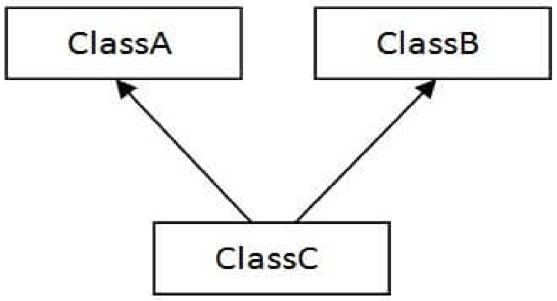
1) Single



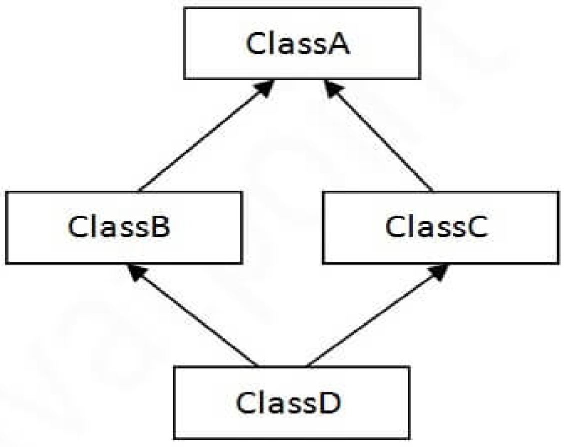
2) Multilevel



3) Hierarchical



4) Multiple



5) Hybrid

.

Single inheritance

```
class father
{
   int age;
   float height;
   String city;
}
class child extends father
{
}
class family
{
    public static void main(String args[])
    {
      child  vini=new child();
      vini.age=20;
      vini.height=5.2f;
      vini.city="hyd";
      System.out.println(vini.age+" "+vini.height+" "+vini.city);
    }
}
```

Output

```
20  5.2  hyd
```

Multi-level inheritance

```
class grand_father
{
    int age;
    float height;
    String city;
}
class father extends grand_father
{


}
class child extends father
{


}
class family
{
    public static void main(String args[])
    {
      child  vini=new child();
      vini.age=20;
      vini.height=5.2f;
      vini.city="hyd";
      System.out.println(vini.age+" "+vini.height+" "+vini.city);
    }
}
```

Output

```
20  5.2  hyd
```

----

note

The name of a method and the list of parameter types in the heading of the method definition is called the method signature.

eg

setDate(int, int, int)
setDate(String, int, int)
setDate(int)

Method overriding

```
class father
{
    void working()
    {
       System.out.println("i am doctor");
    }
}
class child extends father
{
   void working()
    {
       System.out.println("i am singer");
    }

}

class family
{
    public static void main(String args[])
    {
      child  vini=new child();
      vini.working();
    }
}
```

Output

```
i am singer
```

----

using "super" keword to access members(variables,methods) of super class

```
class india
{
   void bowling()
   {
      System.out.println("50");
   }
}
class pak extends india
{
   void playing()
   {
      super.bowling();
      System.out.println("11");
   }
}
class cricket
{
    public static void main(String args[])
    {
      pak odi=new pak();
      odi.playing();

    }
}
```

Output

```
50
11
```

```
class father
{
    void working()
    {
       System.out.println("i am doctor");
    }
}
class child extends father
{
   void working()
    {

       System.out.println("i am singer");
        super.working();
    }

}

class family
{
    public static void main(String args[])
    {
      child  vini=new child();
      vini.working();
    }
}
```

Output

```
i am singer
i am doctor
```

```
class father
{
    double salary=60000;
    void show_sal()
    {
        System.out.println(salary);
    }
}
class child extends father
{
    double salary=35000;
    void show_sal()
    {
        System.out.println(salary);
        super.show_sal();
    }


}

public class family
```

```
{
    public static void main(String args[])
    {
      child  vini=new child();
      vini.show_sal();
    }
}
```

Output

```
35000.0
60000.0
```

constructor and inheritance

```
class gf
{
    gf()
    {
       System.out.println("gf");
    }
}
class f extends gf
{
    f()
    {
       System.out.println("f");
    }
}
class c extends f
{
    c()
    {
       System.out.println("c");
    }
}
public class family
{
    public static void main(String args[])
    {
      c micky=new c();
    }
}
```

output

```
gf
f
c
```

```
class f
{
    f(String s)
    {
       System.out.println(s);
    }
}
class c extends f
{
    c()
    {
       super("sam");
       System.out.println("c");
    }
}
 class family
{
    public static void main(String args[])
    {
      c micky=new c();
    }
}
```

output

```
sam
c
```

"Object" class

- ★ The Object class is the parent class of all the classes in java by default.
- ★ In other words, it is the topmost class of java.
- ★ Object class is present in java.lang package.
- ★ Every class in Java is directly or indirectly derived from the Object class.
- ★ If a Class does not extend any other class then it is direct child class of Object

methods in 'Object' class are

hashCode()

toString()

equals()

getClass()

clone()

examples

```
class a
{

}
class object
{
	public static void main(String ar[])
	{
		a a1=new a();
		a a2=new a();
		a a3=new a();
		System.out.println(a1.hashCode());
		System.out.println(a2.hashCode());
		System.out.println(a3.hashCode());
	}
}
```

output

```
31168322

17225372

5433634
```

```
class a
{
        int i=10,j=20;
}
class object
{
        public static void main(String ar[])
        {
                a a1=new a();
                a a2=new a();
                System.out.println(a1.equals(a2));
                System.out.println(a1.hashCode()+" "+a2.hashCode());
                a1=a2;
                System.out.println(a1.hashCode()+" "+a2.hashCode());
                System.out.println(a1.equals(a2));
        }
}
```

output

```
false
31168322 17225372
17225372 17225372
true
```

```
class a
{

}
class object
{
        public static void main(String ar[])
        {
                a a1=new a();
                System.out.println(a1);

        }
}

/*
output

a@1db9742
```

```
*/
-------------------------------------------------
class a
{

}
class object
{
	public static void main(String ar[])
	{
		a a1=new a();
		System.out.println(a1.toString());

	}
}

/*
output

a@1db9742

*/
-------------------------------------------------------
class a
{
	public String toString()
	{
		return "Hi,i am a";
	}
}
class object
{
	public static void main(String ar[])
	{
		a a1=new a();
		System.out.println(a1);

	}
}

/*
output

Hi,i am a

*/
```

"final" keyword

```
final class father
{

}
class child extends father
{

}
class MyClass
{
    public static void main(String args[])
    {
        System.out.println("hi");
    }
}
```

output

```
error:cannot inherit from final father
```

```
class father
{
    final void working()
    {
        System.out.println(" i am doctor");
    }
}
class child extends father
{
    void working()
    {
        System.out.println(" i am singer");
    }
}
public class MyClass
{
    public static void main(String args[])
    {
        System.out.println("hi");
    }
}
```

output

```
error: working() in child cannot override working() in father
```

```
class IT_A
```

```
{
    final int branch_code=12;
}
public class family
{
    public static void main(String args[])
    {
      IT_A  sasi=new IT_A();
      sasi.branch_code=34;//error
    }
}
```

Output

```
error: cannot assign a value to final variable branch_code
```

----

note

abstract means

not in detail
summary
hiding something

abstract class

A class which is declared with the abstract keyword is known as an abstract class in Java.
It can have abstract and non-abstract methods (method with the body).

An abstract class must be declared with an abstract keyword.
It can have abstract and non-abstract methods.
It cannot be instantiated.
It can have constructors and static methods also.

What is abstract method?

A method which is declared as abstract and does not have implementation is known as an abstract method.

```
abstract   void     working ();//no method body
```

```
 abstract class human
{
    abstract void working();
}
class father extends human
{
    void working()
    {
        System.out.println("I am an engineer");
    }
}
public class family
```

```
{
    public static void main(String args[])
    {
      father raj=new father();
      raj.working();
    }
}
```

Output

```
I am an engineer
```

Objects cant be created for abstract classes

```
abstract class human
{
    abstract void working();
    String color="white";
}

public class family
{
    public static void main(String args[])
    {
      human bob=new human();//error

    }
}
```

Output

```
error: human is abstract; cannot be instantiated
```

```
abstract class shape
{
    abstract void draw();
}
class circle extends shape
{
    void draw()
    {
        System.out.println(" i am circle");
    }
}
public class cricket
{
    public static void main(String args[])
```

```
   {
     circle c=new circle();
     c.draw();
   }
}
```

Output

```
i am circle
```

Packages

A package in Java is used to group related classes. Think of it as a folder.

Packages are divided into two categories:

- Built-in Packages (pre defined)
- User-defined Packages (create your own packages)

example

java.net

used to make two computers to communicate with each other

java.sql

used to connect to database with java program

java.lang

Contains language support classes(e.g classed which defines primitive data types, math operations). This package is automatically imported.

java.lang.Math

Here 'Math' is a predefined class

It has predefined mathematical functions(ceil,floor,sqrt,pow etc)

java.io

Contains classes for supporting input / output operations.

Advantage of package

- ★ Used to group set of similar type of classes
- ★ To avoid naming collisions
- ★ Provides access protection to class members

Access specifiers used in package

private

public

protected

default

private

private members of a class are accessible by members of the same class only.They are not accessible outside of the class

eg   1

```
class cse
{
   private int code=05;
}
class MyClass
{
   public static void main(String args[])
   {

     cse bob=new cse();
     System.out.println(bob.code);//error cant access code
   }
}
```

Output

```
MyClass.java:11: error: code has private access in cse
     System.out.println(bob.code);
                          ^
```

private members are not inheritable

```
class cse
{
   private int code=05;
}
class eee extends cse
{


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

     eee hema=new eee();
     System.out.println(hema.code);//error,code cant be inherited to eee
   }
}
```

Output

```
MyClass.java:16: error: code has private access in cse
     System.out.println(hema.code);
                            ^
```

Interfaces

IT IS A BLOCK OF CODE LIKE CLASS WHICH CONTAINS ONLY

ABSTRACT METHODS
CONSTANT VARIABLES(FINAL)

Interface methods are by default abstract and public

Interface attributes are by default public, static and final

Interfaces specify what a class must do and not how

If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must be declared abstract

syntax

```
interface <interface_name>
{

    // declare constant fields
```

```
            // declare methods that abstract  by default.

        }


        eg

//1 dollar=73rupees
//1 euro =89 rupees
//1 kuwait dinar=241 rupees
import java.util.Scanner;
interface currency
{
        double rupees_to_dollar(double r);
         double rupees_to_euro(double r);
        double rupees_to_dinar(double r);
}
class convert implements currency
{

        public double rupees_to_dollar(double r)
        {
                return (r/73);
        }
        public double rupees_to_euro(double r)
        {
                return (r/89);
        }
        public double rupees_to_dinar(double r)
        {
                return (r/241);
        }
}
class money
{
        public static void main(String coin[])
        {

                double rupee,doll,euro,dinar;
                convert c=new convert();
                Scanner take=new Scanner(System.in);
                System.out.println("\nEnter some indian rupees:");
                rupee=take.nextDouble();
                doll=c.rupees_to_dollar(rupee);
                euro=c.rupees_to_euro(rupee);
                dinar=c.rupees_to_dinar(rupee);
                System.out.println(rupee+" rupees =  "+doll+" dollars");
                System.out.println(rupee+" rupees =  "+euro+" euros");
                System.out.println(rupee+" rupees =  "+dinar+" dinars");
        }
}
```

Output

```
Enter some indian rupees:
100000
100000.0 rupees =  1369.86301369863 dollars
100000.0 rupees =  1123.5955056179776 euros
100000.0 rupees =  414.9377593360996 dinars
```

Why do we use interface ?

- It is used to achieve total abstraction.

```
interface student
{
   void study();
   void sleeps();
   void play();
}

class itguys implements student
{
   public void study()
   {
      System.out.println(" 1 hour per day");
   }
   public void sleeps()
   {
     System.out.println("10 hours per day");
   }
   public void play()
   {
      System.out.println("3 hours per day");
   }

}
public class money
{
         public static void main(String coin[])
         {

         itguys madu=new itguys();
         itguys harika=new itguys();
         madu.sleeps();
         harika.sleeps();
```

```
	}
}
```

Output

```
10 hours per day
10 hours per day
```

- Since java does not support multiple inheritance in case of class, but by using interface it can achieve multiple inheritance .

eg

```
interface father
{
	float height=5.8f;
}
interface mother
{

	String colour="white";
}
class child implements father,mother
{


}
class family
{
	public static void main(String coin[])
	{

		child vani=new child();
		System.out.println(vani.height+ "  "+vani.colour);
	}
}
```

Output

```
5.8  white
```

- We can’t create instance(interface can’t be instantiated) of interface but we can make reference of it that refers to the Object of its implementing class.
- A class can implement more than one interface.
- An interface can extends another interface or interfaces (more than one interface) .
- A class that implements interface must implements all the methods in interface.

```
interface itguys
{
          void sleeps(String p);
          void plays(String q);
          void study(String r);
}
class it implements itguys
{
         public void sleeps(String p)
         {
                  System.out.println("I sleep on "+p);
         }
         public void plays(String q)
         {
                  System.out.println("I like playing "+q);
         }
         public void study(String r)
         {
                  System.out.println("i like studying "+ r);

         }
}
class btech
{
         public static void main(String divya[])
         {
                  it sasi=new it();
                  it kavya=new it();
                  sasi.plays("tennis");
                  kavya.plays("koko");
          }
}
```

- All the methods are public and abstract. And all the fields(variables) are public, static, and final.
- It is used to achieve multiple inheritance.

```
interface father
{
          double height=5.8;
}
interface mother
{
          String  color="white";
}
interface child extends father,mother
{


}
class it implements child
{


}
class btech
{
          public static void main(String divya[])
          {

                    it kavya=new it();
                    System.out.println(kavya.color);
                    System.out.println(kavya.height);
          }
}
```

Dynamic method dispatch(run time polymorphism)

- Method overriding is one of the ways in which Java supports Runtime Polymorphism.
- Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.
- When an overridden method is called through a superclass reference, Java determines which version(superclass/subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time.
- At run-time, it depends on 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
- A superclass reference variable can refer to a subclass object. This is also known as upcasting. Java uses this fact to resolve calls to overridden methods at run time.

Therefore, if a superclass contains a method that is overridden by a subclass, then when different types of objects are referred to through a superclass reference variable, different versions of the method are executed.

Here is an example

```
class svc
{
   void lunch_time()
   {
      System.out.println("12-2");
   }
}
class fy extends svc
{
   void lunch_time()
   {
      System.out.println("12-1");
   }
}
class sy extends svc
{
    void lunch_time()
    {
        System.out.println("1-2");
    }
}
public class MyClass
{
public static void main(String args[])
    {
      svc svec;//reference variable
      fy vini=new fy();
      svec=vini;
      svec.lunch_time();
      sy sasi=new sy();
      svec=sasi;
      svec.lunch_time();
    }
}
```

Output

```
12-1

1-2
```
