UNIT II

CLASSES OBJECT AND METHOD

INTRODUCTION:

- Java is a true object oriented language and therefore the structure of all java program starts with class.
- Class defines and creates objects and object use method to communicate between them.
- Class provides convenient method for packing together data item and function to work on them.
- In java the data items are called fields and function are called methods.
- Java functions include the basic OOP concepts such as Encapsulation, Inheritance and Polymorphism.

DEFINING A CLASS:

- A class is a user defines data type with a template that serves to define its property.
- Once the class has been created we create variable of that data type using object declaration.

SYNTAX:

class Sub_class name [Extends Superclass_name]

{

[field declaration;]

[methods declaration;]

}

FIELDS DECLARARION: We can define data fields inside the body of class definition. There variable are called instance variables because they are created whenever an object of classes is instantiated.

EX:
```
class  Addition
{
int a;
int b;
}
```

The class addition contains two instant variables i.e. a & b.

METHOD DECLARATION:

We must add methods for manipulating a data contained in class.

- Methods are declare inside the body of class but after the declaration of instance variables
- The general form of method declaration is

```
Type methodname(parameter-list)
{
Method_ body ;
}
```

- Method declaration have 4 basic part
  - ➔ Name of method
  - ➔ Type of value the method returns.
  - ➔ List of parameters.
  - ➔ Body of method.

EX:

```
class Rectangle
        {
        int length, width;
        void getdata(int x,int y)
        {
        length= x;
        width= y;
        }
        int rectarea( )
        {
        int area= length * width;
        return(area);
        }
        }
public class Compute
{
public static void main(String args[])
{
Rectangle r=new Rectangle();
```

```
r.getdata(10,20);
int ar=r.rectarea();
System.out.println("Area is="+ar);
}
}
```

To show different between local and instance:-

EX:

```
class Accure
{
int x;
int z;
void m( )
{
int y;
x=10; // legal
y=x; //legal
}
void n( )
{
x=5; // correct
y=10; // illegal
z=6; //correct;
}
}
```

CREATING OBJECT:

- **What is class? Explain how to create objects with an example.**
- **What are objects? How are they created?**

- An object in java is a block of memory that contains space to store all the instance variable.
- Creating an object is also refer to as instantiating an object.
- Object in java are created using new operator.
- The new operator creates an object of the specified class and returns a reference to that object.

SYNTAX:

Classname object_name ; //object declaration

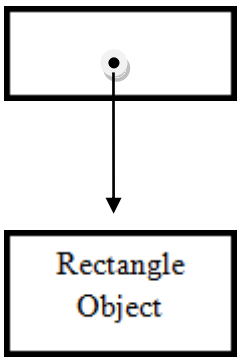
Object_name = new classname ( ); // object creation

EXAMPLE:

```
Rectangle rect;
rect = new Rectangle ( );
```

➔ Above both statement can be combined into one as follow:

```
Rectangle rect = new Rectangle ( );
```



The method Rectangle() is the default constructor of the class. We can create any number of objects of Rectangle. Example:

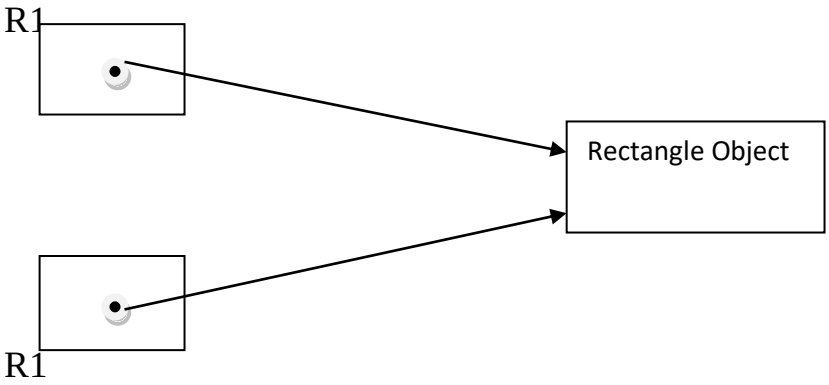
Rectangle rect1 = new Rectangle();

Rectangle rect2 = new Rectangle();// Each object have its own copy of instance variables

It is also possible to create two or more references to the same object.

Rectangle R1 = new Rectangle();

Rectangle R2 = R1;



Both R1 and R2 refer to the same object.

ACCESSING CLASS MEMBER:

- **what is class? How it accomplish data hiding?**

➔ The class member (fields & methods) can be accessed outside the class by using the object of that class.

➔ The use of class allows to hide the important data from the outsider of the class unless it is been defined as public.

➔ The instance variable can be accessed by using the object name dot(.) operator followed by existence variable name.

SYNTAX:

Objectname.variable_name;

➔ To access the method of class use object name dot(.) operator followed by method name to be accessed.

SYNTAX:

Objectname.methodname ( );

EX:

Write a program to calculate the area of Rectangle and prints.

```
class Rectangle
{
int l, b;
void assign( )
{
l=10;
b=10;
}
void compute( )
{
int area=l*b;
System.out.println (“area=”+area);
}
```

```
}
public class Areademo
{
public static void main(String args[ ])
{
Rectangle r= new Rectangle ( );
r.assign( );
r.compute ( );
}
}
```

CONSTRUCTOR:

- **What is constructor? What are its special properties?**
- **Explain different types of constructor with example.**

- Java allows objects to initialize themselves when they are created this automatically initialization is performed through the constructor.
- The constructor is a special method that is used to initialize the variable of class during object initialization.
- The constructors have the same name as the class name on which it resides and is syntactically similar to method.

Properties of constructor:

- They do not specify any return type, not even void.
- They are invoked automatically when the object are created.

- They can have default arguments.
- It can make use of new/delete operator are allocating or releasing memory space.
- Name of the constructor must be same as name of the class.
- A constructor gets invoked automatically when an object gets created.
- Multiple constructors can be used by the same class.

The Constructor are classified into 4 categories:-

1. Default Constructor.
2. Parameterized Constructor.
3. Copy Constructor.
4. Dynamic Constructor.

Q. Write a Java program to illustrate constructor overloading:

```
class Rectangle
{
int a,b;
Rectangle(int x, int y)
{
a=x;
b=y;
}
Rectangle()
```

```
    {
    a=10;
    b=20;
    }
    void  area()
    {
        System.out.println("The area is = "+(a*b));
    }
}
public class Demo
{
public static void main(String args[])
{
Rectangle r1=new Rectangle();
r1.area();
Rectangle r2=new Rectangle(5,10);
r2.area();
}
}
```

METHOD OVERLOADING:

- **What is meant by method overloading? Explain.**
- **Explain with program the concept of method overloading.**

1. Method overloading allow us to create method that has same name but different parameter list & different definition.
2. It is used when object are required to perform similar task but using different input parameter .This process is known as "Polymorphism".
3. When we call the method in object java matches the method name first and then the no and type of parameter to decide which one of the definition to execute.

Example:

```
class Addition
{
void add(int a, int b)
{
System.out.println ("sum="+(a+b));
}
void add(float a, float b)
{
System.out.println ("sum="+(a+b));
}
}
public class Demo
{
public static void main(String args[])
```

```
    {
    Addition a1=new. Addition( );
    a1.add(10,100);
    a1.add(25.0f, 10.15f);
    }
}
```

STATIC MEMBERS:

- **Explain static method with an example code.**
- **Write a note on static members.**

→The variables and the methods that belongs to the class as a whole rather than the objects created from the class such members can be defined as follows:

Static int count;

Static int max (int x, int y);

→These members that are declared static are called static members and it can be often referred as class variables and class methods.

→These static variables and methods can be called without using the objects.

➔ Method declared as static have several restriction:

1. They can call only other static method.
2. They must only access static data.
3. They can't refer to this in anyways.

Static variable and methods are associated with a class itself; hence they are called class variable or class members.

EX: Write a program to illustrate the use of static member.

```
class Mathoperation
{
static float mul(float  x,float  y)
{
return (x*y);
}
static float  divide(float  x, float  y)
{
return (x/y);
}
}
public class Mathapp
{
public static void main(Strings args[])
{
float  a=Mathoperation.mul(4.0f,5.0f);
float  b=Mathoperation.divide(a,2.0f);
System.out.println("b="+b);
}
```

OUTPUT:

The output would be b=10.0

NESTING OF METHOD:

➔ The method call by an another of same class using the name of method is called nesting of method.

➔ Considered the following example:-

```
class Nesting
{
int m, n;
Nesting (int x, int y) // Construction Method
{
m=x;
n=y;
}
int largest( ) // called method
{
if(m>n)
return(m);
else
return(n);
}
void display( ) // Calling method
{
int large= largest( ); //nesting of method
System.out.println("largest value="+large);
}
}
```

```
public class Demo
{
public static void main(String args[])
{
Nesting a= new Nesting(50,40);
a.display( );
}
}
```

Output:

Largest value=50

NESTED AND INNER CLASS:

- **Write a note on inner classes.**
- **Explain with example nesting of methods.**

1. It is possible to define a class within another class such class are known as nested class.
2. The scope of the nested class is bounded by scope of its enclosing class.
3. There are two type of nested class: static and non-static.
4. A static nested class is one that has the static modified applied because it is static it must access one member of its enclosing class through.
5. An inner class is a non-static nested class it has access to all of the variable and methods of its outer class and may refer directly.

Considered the following example:

- Hence the main method of Inner class Demo creates an instance of class Outer and the display ( ) method is called.

Q.Write a Java program to demonstrate the use of nested class.

```
class Outer
{
int a=10;
class Inner
{
int b=20;
void display()
{
System.out.println("The value of A="+a);
System.out.println("The value of B="+b);
}
}
}
public class Demo
{
public static void main(String args[])
```

```
{
Outer out1=new Outer();
Outer.Inner in1=out1.new Inner();
in1.display();
}
}
```

Inheritance:

Extending the Classes:

- **Explain what 'protected' visibility label mean, when used in super classes.**
- **Describe different forms of inheritance with figure.**

→Inheritance is a mechanism of deriving a new class from an old one.

→The old class is known as base class or super class or parent class and the new one is called as the sub class or derived class or child class.

→In java Inheritance may take different forms:

i. Single inheritance : (only one super class)
ii. Multiple inheritance : (several super class)
iii. Hierarchical inheritance : (1 super class many sub class)
iv. Multilevel inheritance : (Derived from a desired class)
v. Hybrid inheritance: (Combination of multiple and hierarchical inheritance)

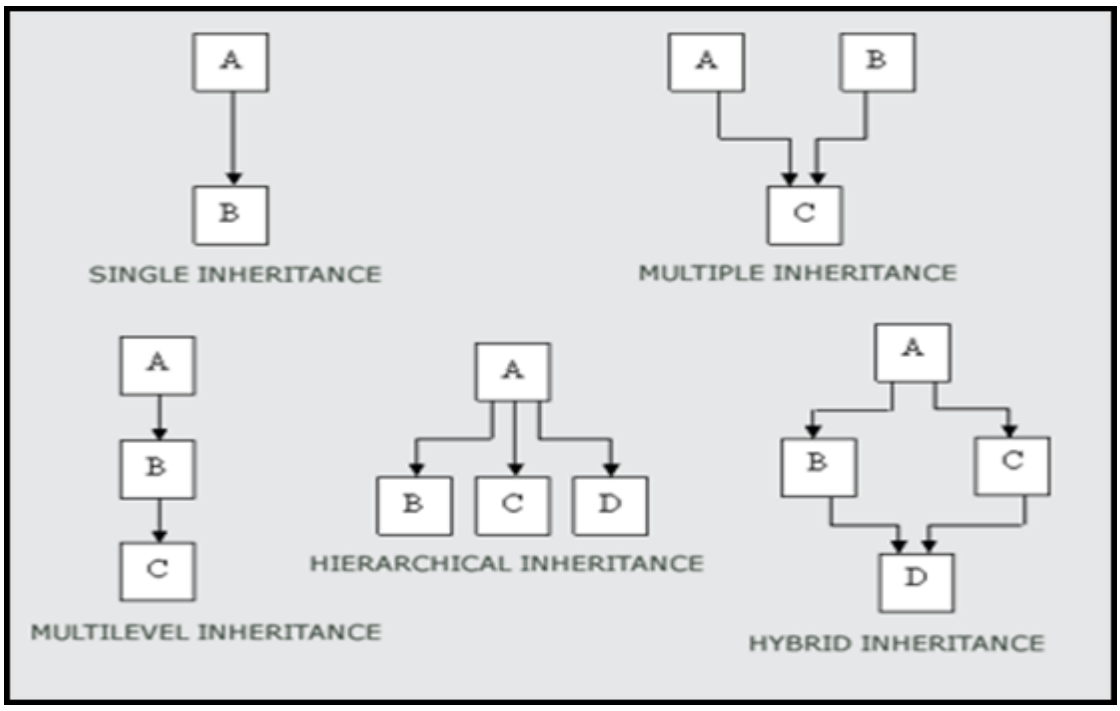


Fig: Forms of inheritance

Defining of a Sub Class:

- **Explain how to define sub class.**
- To inherit a class, you simply incorporate the definition of one class into another by using the 'extends' keyword.
- *Syntax: class sub_class_name extends super_class_name*

  {

  Variable declaration;

  Method declaration;

  }

- The keyword 'extends' signifies that the properties of the superclass_name are extended to the subclass_name.

- Although a subclass includes all of the members of its super class, it can't access those member of the super class that have been declared as private.

  Ex. class B extends A

{

//body of class

}

Example of Single Inheritance:

```
class Room
{
int length, breadth;
 Room (int x,int y)
 {
length=x;
breadth=y;
}
 int area()
{
return (length * breadth);
     }
 }
class Bedroom extends Room
{
int height;
```

```
 Bedroom(int x,int y,int z)
 {
super (x,y);
height = z;
 }
int volume ()
{
return (length*breadth*height);
 }
}
   public class Inherit
    {
   public static void main(String args[])
    {
  Bedroom  room1 =new Bedroom(10,20,30);
   int area=room1.area ();
   int volume= room1.volume ();
  System.out.println ("Area="+area);
  System.out.println ("volume="+volume);
      }
}
```

OUTPUT:

```
  Area = 200
 Volume =6000
```

Subclass constructor:

1. The subclass constructor is used to construct the instance variables of both the subclass and the super class.
2. A subclass constructor use the keyword 'super' to invoke the constructor method of the super class.
3. Super has two general forms. The first calls the super class constructor. The second is used to access a member of the super class that has been hidden by a member of a subclass using 'super' to call super class constructor.

   →A subclass can call a constructor defines by it super class by use of the following form of super class.

   super (arg_list);

   →Here, arg_list specifies any arguments needed by the constructor in the super class.

The following are the conditions for using 'super':

i. Super may only be used within a subclass constructor method.
ii. The call to super class constructor must appear in the first statement within a subclass constructor.
iii. The parameter in super class must match the order and type of instance variable declared in the super class.

Using 'super' to access a member of super class:

The second form of 'super' acts somewhat like this except that it always refers to the super class of the subclass it is used.

Syntax:

super.member;

Here, member can be either a method or an instance variable.

Multilevel Inheritence: -

The class A serves as a base class for the derived class B which in class serves as a base class for the derived class C.The Chain ABC is known as Inheritance path.

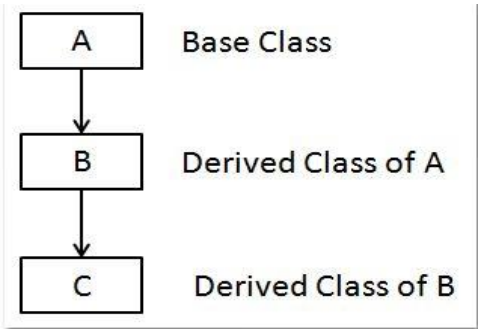


Fig. Multilevel Inheritance

A derives class with multilevel base classes is declared as follows.

class A

{

……

……

}

class B extends A *//first level*

{

……

……

}

class C extends B *// Second level*

{

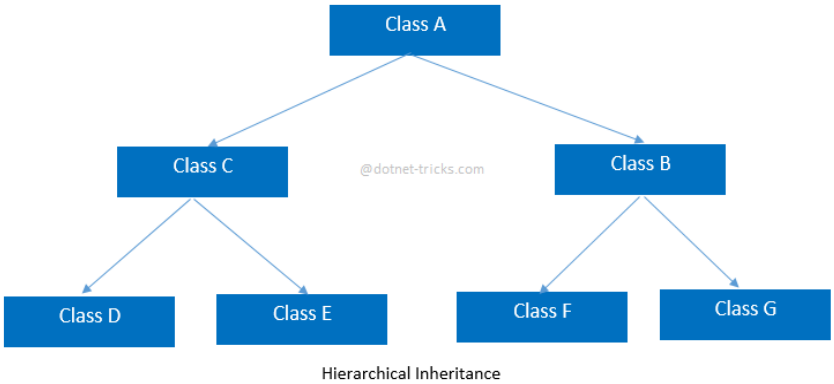
…….

……

}

Hierarchical Inheritance: -

Fig below shows a hierarchical classification of different classes.



Hierarchical Inheritance

Overriding Methods:

- **Compare overloading and overriding methods.**
- **Explain with example overriding of methods.**
- **What do you mean by method overriding.**

→When a method in a subclass has the same name and type signature as a method in its super class, then the method in the subclass is said to be over ride method.

When an overridden method is called from within a subclass, it will always refer to the method which is defined by the subclass and the method defined by the super class will be hidden.

→Consider the following example:

```
class Using_super
{
int x;
Using_super(int x)
{
this.x=x;
}
void display()
{
System.out.println("super x="+x);
}
}
```

```
class Sub Extends Using_super
 {
 int y;
 sub(int  x,int  y)
{
super (x);
this.y=y;
 }
void display() //method over ridden
 {
System.out.println("super x="+x);
System.out.println("sub y="+y);
 }
}
public class OverrideTest
{
public static void main (String args[])
{
Sub s1 = new Sub (10,20);
s1.display ();
}
}
```

OUTPUT:

```
super x=10
sub    y=20
```

Note that the method display () defined in the subclass is invoked

Final Variables and Methods:

- **Explain the uses of final modifier.**
- **Explain the use of keyword final.**

→A variable can be declared as 'final', doing so prevents its contents from being modified.

→'final' is a keyword which is used for declaring variables, methods and classes.

→final keyword can be used to

i. To declare constants

Syntax: final data_type variable = value;

Ex: final int FILE count = 10;

ii. To prevent method overloading

Syntax: final return_type method_name()

{

// body of method

}

iii. To prevent inheritance of class or extending of classes.

Syntax: final class Classname

{

//body of class

}

→It is a common coding convention to choose all upper case letters for final variables. This type of a final variable is essentially constants.

→Variable declared as **final** do not take any memory space on individual objects of the class.

Final Classes:

Sometimes we may like to prevent a class being further subclasses for security reasons. A class that cannot be sub classed is called final class. This is achieved in java using the keyword final as follows:

final class Aclass {……..}

final class Bclass Extends Someclass(……..)

Any attempt to inherit these classes will cause an error and the compiler will not allow it. Declaring a class final prevents any unwanted extensions to the class. It also allows the compiler to perform some optimizations when a method of a final class is invoked.

The finalize () methods:

- **Write a note on: Finalized method.**
- **What is significance of finalize () method?**

1. 'finalize ()' method is a method similar to the destructor method of the c++ to program garbage collection of non-object resources.

2. To add a finalize to a class, you simply define the finalize () and can be called in any class.
3. Java calls the finalize () method whenever it is about reclaim the space for the object.
4. The finalize () method has this general form

```
void finalize ()
{
//finalization code
}
```

The finalize () method should Explicitly define the task to be performed.

Abstract Methods and Classes:

- **What are abstract classes? Explain.**
- **Write a short note on: Abstract Methods.**
- **When to declare a class abstract? Explain with example.**

★ Abstract is a keyword used for declaring classes and methods.

★ Abstract method is a method which is just as declared but not defined we can declare methods with semicolon.

Ex: abstract void show ();

★ Abstract class has to be inherited and the class which is inheriting the abstract class must give the definition for the abstract method declared in the abstract class

★ Following are the condition for using abstract class:

1. We can't use abstract class to instantiate objects directly.
2. The abstract method of an abstract class must be defined in its sub class.
3. We can't declare abstract constructor or abstract static methods.

Abstract classes Example:

```
abstract class A
{
int a,b,c;
void assign()
{
a=b=10;
}
abstract void add();// declaring abstract method
}
class B extends A
{
void add()
{
c=a+b;
System.out.println("c="+c);
}
}
class Demo
{
public static void main(String args[])
```

```
{
B b1 = new B ();
b1.assign ();
b1. add ();
}
}
```

Methods With Variable Arguments (Varargs):

→Programs can create methods that accept unspecified number of arguments.

→An arguments type followed by eclipsed (**…**) in a method parameter list indicates that the method receives a variable member of arguments of that type.

The use of the ellipse (**…**) can occur only once in a parameter lists.

Syntax:

*Access_specifier static void method_name (**objects…args**)*

```
{
//body of method
}
```

→The ellipses with its type must be placed at the end of the parameter list.

→*consider the following example:*

```
class Exampleprg
{
```

```
Static void Exa(String...person)
{
for(String name:person)
{
System.out.println("Hello "+name);
}
}
public static void main(String args[])
{
Exa("John","David","Suhel");
}
}
```

The above program produces the following output:

Hello John

Hello David

Hello Suhel

At the compile time String…var_arg is converted to String[] var_arg.

Visibility Control:

- **Explain different levels of access protection available in java.**
- **Discuss the various levels of access protection available for packages.**
- **List and explain visibility modifier used in java.**
- **Explain visibility control in java.**

We can restrict access to certain variables and methods from outside the class by applying the visibility modifier or access modifier which controls the visibility of variables and methods.

The different access modifiers available in java are:

1. Public.
2. Protected.
3. Default (friendly).
4. Private.
5. Private protected.

Public Access:

The variable or a method declared as public as the widest possible visibility and accessible everywhere.

Protected Access: The protected visibility level lies in between the public access and friendly access the protected modifier makes the field visible not only to all classes and sub classes in same package but also subclasses in other packages.

Default Access:

Default access makes field visible only in the same package but not in other package.

Private Access:

It provides the highest degree of protection, they are accessible only within this own class.

They can't be inherited, the method declared as private behave like method declared as final.