UNIT - II

Classes, Objects and Methods:-

Introduction :-

* Class that defines the state and behaviour of the basic program components known as objects
* classes create objects and objects use methods to communicate between them.
* Classes provide a convenient method for packing together a group of logically related data items and functions.
* The data items are called fields and the functions are called methods.

Defining a class:-

* These variables are termed as instance of classes, which are the actual objects.

Basic form

```
class classname [extends superclassname]
{
    [variable declaration;]
    [methods declaration;]
}
```

* Everything inside the square brackets is optional.

Class definition

```
class Empty
{
}
```

* The body is empty, this class does not contain any properties.

* classname and superclassname are any valid java identifiers.

* The keyword extends indicates that the properties of the superclassname class are extended to the classname.

Adding Variables :-

* Data is encapsulated in a class by placing data fields inside the body of the class definition.

* These variables are called instance variables

Eg

```
class rectangle
{
    int length;
    int width;
}
```

or `int length, width;`

* Instance variables are also known as member variables.

Adding Methods:-

* A class with only data fields has no life.
* The objects created by such a class cannot respond to any messages.
* Methods are declared inside the body of the class

Syntax

```
type    methodname (parameter-list)
{
    method-body;
}
```

Method declarations have four basic parts:

1) The name of the method (methodname)
2) The type of the value the method returns (type)
3) A list of parameters (parameter-list)
4) The body of the method.

* The type specifies the type of value the method would return.
* It could be any simple datatype like int, void and so on.
* void method does not return any value.
* The methodname is a valid identifier.
* The parameter list is enclosed in paraentheses.
* That list contains variable name and their type. And also it is seperated by commas.

Eg

(int m, float x, float y) // three parameter

() // Empty list.

* The body describes the operations to be performed on the data.

Eg

```
class Rectangle
{
  int length, width;
  void getgdata (int x, int y)
  {
    length = x;
    width = y;
  }
  int rectarea ()
  {
    int area = length * width;
    return (area);
  }
}
```

// this method has not return any value because of void fn. and it is used instance variables.

// rectarea () returns the result in int type.

creating Objects

* Creating an object is also referred to as instantiating an object.
* Objects in Java are created using the new operator.
* The new operator creates an object of the specified class and returns a reference to that object.

Eg

```
Rectangle rect1;          // declare
rect1 = new Rectangle();  // instantiate
```

Explain the eg

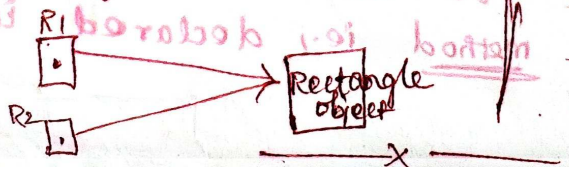
* The first statement declares a variable to hold the object reference and secone statement assigns the object reference to the variable.
* The varible rect1 is now an object of the Rectangle class.

Both statements can be combined in one statement

```
Rectangle rect1 = new Rectangle();
```

* It is possible to create two or more reference to the same object.

```
Rectangle R1 = new Rectangle();
Rectangle R2 = R1;
```



Both R1 & R2 refer the same object

Accessing class Members:-

* All variables must be assigned values before they are used.
* In outside the class, it cannot access the instance variables and methods directly.
* It use the concerned object and the dot operator using:

Syntax

```
objectname.variablename
objectname.methodname(parameter_list);
```

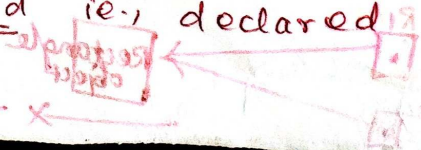
⇒ objectname is the name of the object, Variablename is the name of the instance variable inside the object.
methodname is the method that to wish the call and parameter_list is a comma seperated list of "actual values".

Example:

```
rect1.length = 15;
rect1.width = 10;
rect2.length = 20;
rect2.width = 12;
```

① ⇒ This is one way of assigning values to the variables in the objects.

② ⇒ Assigning values to the instance variables is to use a method ie., declared inside the class.



example

```
Rectangle rect1 = new Rectangle(); // creating object
rect1.getdata(15, 10); // calling the method using the object.
```

Constructors :-

* Two Types

1) The first approach uses the dot operator to access the instance variables and then assigns values to them individually.

* It can be a tedious (cumbersome) approach to initialize all the variables of all the objects.

2) The second approach takes the help of a method like getdata to initialize each object.

Eg rect1.getdata(15, 10);

* Java supports a special type of method called a constructor, that enables an object to initialize itself when it is created.

* 1) Constructors have the same name as the class.

2) They do not specify a return type, not even void.

Eg

```
class rectangle
{
 int length, width;
 rectangle(int x, int y)   // define constructor
 {
   length = x;
   width = y;
 }
int rectArea()
 {
  return (length * width);
 }

class Rectarea2
{
  public static void main(String args[])
  {
   rectangle rect1 = new rectangle(15,10);
                          ↳ // calling construct
   int area1 = rect1.rectarea();
   System.out.println("Area1 = " + area1);
  }
}
```

O/P

Area1 = 150.

Methods Overloading

* It is possible to create methods that have the same name, but different parameter lists and different definitions.
* This is called method overloading.
* It is used when objects are required to perform similiar tasks but using different i/p parameters.
* Java matches up the method name first and then the number and type of parameters. This known as polymorphism.
* Each parameter list must be unique.

Ex

```
class Room
{
    float length,width;
    Room (float x, float y)   // constructor1
    {
    length = x;
    width = y;
    }
    Room (float x)            // constructor 2
    {
    length = width = x;
    }
int area()
{
return (length * width);
}
```

Static Members:-

* A class basically contains two sections.
  1. declares variables
  2. declares methods.
* These variables and methods are called instance variables and instance methods.
* Every time the class is instantiated, a new copy of each of them is created.
* To define a member that is common to all the objects and accessed without using particular object.
* Example

```
static int count;
static int max(int x, int y);
```

* The members that are declared static as shown are called static members.
* The static variables and static methods are referred to as class variables and class m
* Java class libraries contain a large numbe class methods.

Eg

```
class Mathoperation
{
  static float mul(float x, float y)
  {
    return x*y;
  }
```

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

O/P
b = 10.0

Static methods are called using class names.
No objects have been created for use.
Static methods have several restrictions.

1. They can only call other static methods
2. They can only access static data.
3. They cannot refer to this or super in any way.

Nesting of Methods :-

* A method can be called by using only its name by another method of the same class.
* This is known as nesting of methods.

Example

```
class Nesting
{
    int m, n;
    Nesting (int x, int y)          // Constructor method
    {
        m = x;
        n = y;
    }
    int largest()
    {
        if (m >= n)
            return (m);
        else
            return (n);
    }
    void display()
    {
        int large = largest();      // calling a method
        System.out.println ("Largest value = " +large);
    }
}
class NestingTest
{
    public static void main (String args[])
    {
        Nesting nest = new Nesting (50, 40);
        nest.display();
    }
}
```

* In this Example have one constructor and two methods. The method display() calls the method largest() to determine the largest of the two numbers and display the result.

— x —

Inheritance : Extending a class :-

* Creating new classes, reusing the properties of existing ones.
* Deriving a new class from old class is called inheritance.
* old class is also called as base class or superclass or parent class.
* new class is called as subclass or derived class or child class.

* The inheritance allows subclasses to inherit all the variables and methods of their parent classes.

Type

1. Single inheritance
2. Multiple inheritance
3. Hierarchical "
4. Multi level

: Java does not directly implement multiple inheritance.

* This concept is implemented using a interface.

Defining a Subclass

Syntax

```
class subclassname extends superclassname
  {
    variable declaration;
    methods declaration;
  }
```

* The keyword extends signifies that the properties of the superclassname are extended to the subclassname.
* The subclass contain its own variables and methods of the superclass.

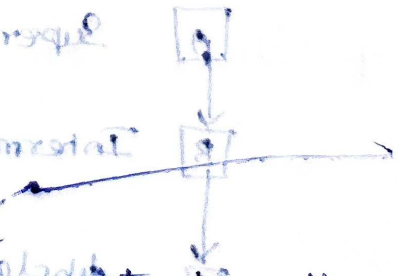
eg

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

```
class BedRoom extends Room   // inheriting Room
{
  int height;
  BedRoom(int x, int y, int z)
  {
    super(x, y);   //pass values to super class
    height = z;
  }
  int volume()
  {
    return( length * breadth * height);
  }
}
class InherTest
{
  public static void main(String args[])
  {
    BedRoom room1 = new BedRoom(14, 12, 10);
    int area1 = room1.area();   // superclass method
    int volume1 = room1.volume();   // base class
    System.out.println("Area1 = " + area1);
    System.out.println("Volume1=" + volume1);
  }
}
```

op

Area1 = 168
volume1 = 1680



Subclass constructor

* It is used to construct the instance variables of both the subclass and the superclass.

* The subclass constructor uses the keyword super to invoke the constructor method of the superclass.

* The keyword super is used to some conditions

1. super may be used within a subclass constructor method
2. The call to superclass constructor must appear as the first statement within the subclass constructor.
3. The parameters in the super call must match the order and type of the instance variable declared in the superclass.

Multilevel Inheritance

* A common requirement in object-oriented programming is the use of a derived class as a super class.
* Java supports this concept and uses it class library.

Superclass

Intermediate superclass

subclass.

Hierarchical Inheritance

* One base class contains many derived class.

Overriding Methods:-

* Super class is inherited by its subclass and is used by the objects created by the subclass.
* Method inheritance enables to define and use methods repeatedly in subclass.
* That means, override the method defined in the superclass.
* ie., subclass method having the same name, same arguments of the super class.

Definition

⇒ When that method is called, the method defined in the subclass is invoked and executed instead of the one in the superclass.

⇒ This is known as overriding.

Eg

```
class super
{
    int x;
    super (int x)
    {
        this.x = x;
    }
```

```
    void display()                    // method defined
    {
    System.out.println("Super x=" +x);
    }
}
class sub extends Super
{
  int y;
  sub(int x, int y)
  {
    super(x);
    this.y = y;
  }
  void display()                      // again defined
  {
System.out.println("super x=" +x);
System.out.println("sub y=" +y);
  }
}
class OverrideTest
{
public static void main(String args[])
  {
    sub s1 = new sub(100,200);
    s1.display();
  }
}
```

Final Variables and Methods

* All methods and variables can be overridden by default in subclasses.
* To prevent the subclasses from overriding the members of the superclass, declare them as final using the keyword final as a modifier.

Eg

```
final int size = 100;
final void showstatus() {...}
```

* Final variables behave like class variables and they do not take any space on individual objects of the class.

Final classes:-

* A class that cannot be subclassed is called a final class.

Syntax

```
final class Aclass {...}
final class Bclass extends someclass
{...}
```

Finalizer Methods:-

* A constructor method is used to initialize an object when it is declared.
* This process is known as initialization.
* Java supports a concept called finalization. it is just opposite of initialization.
* Java run-time is an automatic garbage collecting system.
* It automatically frees up the memory resources used by the objects.
* Finalizer method is similar as destructors.
* This method is simply finalize() and can be added to any class.
* Java class that method it reclaim the space for that object.

Abstract Methods and classes:-

* A method must be redefined in a subclass thus making overriding compulsory.
* This is done using the modifier keyword abstract in the method definition.

Eg:

```
abstract class shape
{
....
abstract void draw();
....
}
```

* When a class contains one or more abstract methods, it should be declared abstract. a

conditions

1) It cannot use abstract classes to instantiate objects directly.

Eg Shape s = new Shape()

This is illegal bcoz Shape is an abstract class.

2) The abstract methods of an abstract class must be defined in its subclass.

3) It cannot declare abstract constructors

Visibility Control

* Java applying visibility modifiers to the instance variables and methods.

* The visibility modifiers are also known as access modifiers.

* Three Types of visibility modifiers:

1) public
2) private
3) protected

Public Access

* Any variable or method is visible to the entire class in which it is defined.

Eg public int number;

public void sum() {...}

friendly Access

* When no access modifier is specified, the member defaults to a limited version of public accessibility known as "friendly" level of access.

Difference between Public access and friendly access

* Public modifier makes fields visible in all classes.
* friendly modifier makes fields visible only in the same package.

protected Access

* The visibility level of a "protected" field lies in between the public access and friendly access.
* It is not only to all classes and subclasses in the same package

private Access

* private fields provide the highest degree of protection.
* It is accessible only within their class.
* It cannot be inherited by subclasses.

private protected Access

* It can be declared in two keywords.

private and protected

Eg

private protected int codeNumber;

⇒ It gives a visibility level in between the "protected" access and "private" access.

⇒ This modifier makes the fields visible in all subclasses.

4