Object Oriented Programming with Java 18IS34

Unit-2

Introducing classes, objects and methods

- A class is a template that defines the form of an object.
- Objects are instances of a class

Defining a class

```
class classname {
        type var1;
        type var2;
        . .. type varn;
        type method1(params) {
                // body of method
        }
        ....
        type methodn(params) {
                // body of method
        }
}
```

Introducing classes, objects and methods

- Defining a class

```
class vehicle {
        int passengers, fuelcap,mpg;
        }
class vehicleDemo{
        public static void main( String [] args){
        vehicle minivan = new vehicle();
        int range;
        minivan.passengers = 7;
        minivan.fuelcap = 16;
        minivan.mpg = 21;
        range = minivan.fuelcap * minivan.mpg;
System.out.println(“Minivan can carry” + minivan.passengers + “ with a range of “ + range);
                }
}
```

Introducing classes, objects and methods

- Defining a class
- object creation

vehicle car1 = new vehicle();

vehicle car2 = car1

- methods

```
class vehicle {
        int passengers, fuelcap,mpg;
        void range(){
                System.out.println(“range is :” + fuelcap *mpg);
        }
```

```
class vehicleDemo{

        public static void main( String [] args){
         vehicle minivan = new vehicle();

        vehicle sportscar = new vehicle();

              int range1,range2;

        //int range;

        minivan.passengers = 7;

        minivan.fuelcap = 16;

        minivan.mpg = 21;

        sportscar.passengers = 7;

        sportscar.fuelcap = 16;

        sportscar.mpg = 21;

//      range = minivan.fuelcap * minivan.mpg;

//      System.out.println("Minivan can carry" + minivan.passengers + "with a range of" + range);   //“Minivan can carry” + minivan.passengers + //“ with a range of “ + range)

      minivan.range();

      sportscar.range();                }

}
```

Returning from a method

```
int myMath()
{
    for int (i=0;i<10;i++){
        if(i=5) return;
        System.out.println(i);
}
```

Returning a value

```
class vehicle{
        int range(){
                return mpg * fuelcap;
        }
}
class setMath{
        public static void main(String [] args){
        range1=minivan.range();
        range2 = sportscar.range();
        //print the values
}
}
```

using parameters

```
public class Parademo {


    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        checknum e = new checknum();
            if(e.isEven(10))
                    System.out.println("10 is Even");


    }
}
```

using parameters

```
class checknum{
        boolean isEven(int x){
            if((x%2) == 0)
                return true;
            else
                return false;
        }
 }
public class Parademo {
    public static void main(String[] args) {
        // TODO code application logic here
        checknum e = new checknum();
            if(e.isEven(10))
                    System.out.println("10 is Even")
    }
    }
```

Constructors

- **Constructor initializes an object when it is created.**

```
class myclass{
        int x;
        myclass( ) { x=10; }
 }
public class consDemo {
    public static void main(String[] args) {
        // TODO code application logic here
        myclass t1 = new myclass();
        myclass t2 = new myclass();
         System.out.println(t1.x + "  " + t2.x);
    }
    }
```

Parmaterised Constructors

- **Constructor initializes an object when it is created.**

```
class myclass{
        int x;
        myclass(int i) {  x = i;  }
 }
public class consDemo {
    public static void main(String[] args) {
        // TODO code application logic here
        myclass t1 = new myclass(10);
        myclass t2 = new myclass(8);
        System.out.println(t1.x + "  " + t2.x);
    }
    }
```

Adding Constructor to Vehicle class

- Add the code

Adding Constructor to Vehicle class

```
class vehicle {
        int passengers, fuelcap,mpg;
        vehicle(int p, int f, int m)
        {
                passengers=p;   fuelcap=f;  mpg = m;
        }
        }
class vehicleDemo{
        public static void main( String [] args) {
        vehicle minivan = new vehicle(7,6,21);
        vehicle sportscar = new vehicle(2,14,12);
        int range;
                range = minivan.fuelcap * minivan.mpg;
System.out.println("Minivan can carry" + minivan.passengers + " with a range of " + range);
                }
} // Netbeans consVehicle project
```

Adding Constructor to Vehicle class

- Objects are dynamically allocated from a pool of free memory by using new operator.
- Garabage collector reclaims objects automatically.
- It is possible to define a method called a finalizer.

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

this keyword

- When a method is called it is automatically passed an implicit argument that is a reference to the invoking object. This reference is called this.
- netbeans..... power project

Controlling access to class members

- Class provides two major benefits.
  - 1st it links data with the code that manipulates it.
  - 2nd it provides means by which access to members can be controlled.
- Three access modifiers: public, private and protected.
- Private members can be accessed only by members of its class.
- Public members can be accessed by every code.(even in other classes)
- Protected modifier is useful in inheritance.

Controlling access to class members

```
class myclass {
        private int alpha;
        public int beta;
        int gamma;
        void setalpha( int a){
                alpha = a;
        }
        void getalpha(){
                return alpha;
        }
}  // Netbeans ControlAccess project
```

Controlling access to class members

```
class AccessDemo{
    public static void main(String {} args){
    myclass ob = new myclass();
    ob.setalpha(-99);
    System.out.println("ob.alpha is" + ob.getalpha());
    //ob.alpha = 10;
    ob.beta =88;
    ob.gamma = 99;
    }
}
```

Controlling access to class members

- **fail-soft int array in which boundry errors are prevented avoiding runtime exeption.**
- **Encapsulating array as a private member of a class allowing access to the array only through member methods.**
- **Netbeans FSDemo project**

```
class FailsoftArray{
        private int [ ] a;
        private int errval;
        public int length;
        public FailsoftArray(int size, int errv)
        {
           a = new int[size];
           errval = errv;
           length = size;
         }
}
```

Controlling access to class members

```
public class FSDemo {
      public static void main(String[] args) {
       // TODO code application logic here
       FailsoftArray fs = new FailsoftArray(5,-1);
       int x;









}
```

```
class FailsoftArray{
        private int [ ] a;
        private int errval;
        public int length;
        public FailsoftArray(int size, int errv)
        {
           a = new int[size];          errval = errv;          length = size;
         }
public int get(int index)
        {
           if(ok(index)) return a[index];          return errval;

        }
        public boolean put(int index, int val)
        {   if(ok(index)) {
             a[index]=val;          return true;
           }
           return false;
                 }
```

```
    private boolean ok(int index)
    {
        if(index>=0 && index<length)
            return true;
        return false;

    }
}
```

Pass objects to methods

- **passing of objects to methods.**

```
class Block{
        int a,b,c;
        int volume;
        Block(int i,int j, int k){
           a=i;
           b=j;
           c=k;
           volume= a*b*c;
      }
      boolean sameBlock(Block ob){
           if((ob.a == a) &&(ob.b == b) && (ob.c == c))
                      return true;
           else
              return false;

      }//passobj project in Netbeans
}
```

Pass objects to methods

```
public class Passobj {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Block ob1 = new Block(10,2,5);
        Block ob2 = new Block(10,2,5);
        Block ob3 = new Block(6,5,5);
        System.out.println("ob1 has same dimensions as ob2:" + ob1.sameBlock(ob2));
        System.out.println("ob1 has same dimensions as ob3:" + ob1.sameBlock(ob3));
        System.out.println("ob1 has same volume as ob2:" + ob1.samevolume(ob2));
        System.out.println("ob2 has same volume as ob3:" + ob2.samevolume(ob3));

    }

}
```

Passing of arguments

There are two ways in which passing of the arguments can be done

- pass/call by value : This approach copies the value of an argument into the formal parameters of the subroutine.
- Pass/Call by reference: A refernce to an argument is passed to the parameter.

```
class test{
   int a,b;
   void noChange(int i, int j){
      i=i+j;
      j=-j;
      }
   test()
   {

   }
} // PassArgs project in Netbeans
```

Passing of arguments

There are two ways in which passing of the arguments can be done

- pass/call by value : This approach copies the value of an argument into the formal parameters of the subroutine.
- Pass/Call by reference: A refernce to an argument os passed to the parameter.

```
public class PassArgs {


    public static void main(String[] args) {
    // TODO code application logic here
    test ob1 = new test();
    int a=15,b=20;
    System.out.println("a and b before call:\n"+ a + " " +b);
    ob1.noChange(a,b);
    System.out.println("a and b after call: \n"+a+""+b);
    test ob2 = new test(15,20);
}
}
```

Method Overloading

- **Two or more methods within same class can share the same name, but their parameter declarations are different in these cases methods are said to be overloaded and is called method overloading.**
- **// MethOverload Proj in Netbeans**

```
class overload {
    void ovlDemo(){
        System.out.println("No Parameters");          }
    void ovlDemo(int a){
        System.out.println("One parameter: " + a);    }
    int ovlDemo(int a,int b){
        System.out.println("Two parameters: " + a + " " + b);          return(a+b);     }
    double ovlDemo(double a,double b){
        System.out.println("Two double parameters: " + a + " " + b);       return(a+b);    }
    }
```

Overloading Constructors

```
class myclass {
     int x;
    void myclass(){
       System.out.println("Inside Myclass( )");   x=0;          }
   void myclass(int i){
       System.out.println("Inside myclass(int): ");  x=i   }
   void myclass(double d){
       System.out.println("Inside myclass(double): “);          x = (int)d;     }
   double ovlDemo(int a,int b){
       System.out.println("Inside myclass(int, int):” );    x = i * j;    }
   }
```

understanding static

- **When a member is declared static it can be accessed before any objects of its class are created and without reference to any object.**
- **Both variables and methods can be declared as static.**

```
class staticDemo {
        int x;
        static int y;
        int sum() {
                return x+y;
        }
}
  class SDemo{
public static void main(){
        staticDemo ob1=new staticDemo();
        staticDemo ob2=new staticDemo();
        ob1.x = 10;
        ob2.x = 20;
        staticDemo.y = 19;
        System.out.println(ob1.sum());
        System.out.println(ob2.sum());
}
```

static methods

- **Methods declared as static are global methods.**
- **They can be called without any object.**

```
class staticMeth {
    static int val = 1024;
    static int valDiv2() { return val/2; }
}

  class SDemo{
public static void main(){
System.out.println(“val is”, staticMeth.val);
System.out.println(“staticMeth.valDiv2():”+ staticMeth.valDiv2());
}
}
```

static Blocks

- **Java allows you to declare a static block.**
- **It is executed when class is loaded.**

```
class staticBlock {
    static double root2;
    static { root2 = Math.sqrt(2.0); }
}

  class SDemo{
public static void main(){
System.out.println("Square rot of 2 is"+ staticBlock.root2);
}
}
```

Recursion

- **Method calling itself is called recursion.**

```
void drawstars(int n) {
        if(n == 1)
        {
                System.out.println(“*”);

        }

        else
        {
        System.out.println(“*”);
        drawstars(n-1);
        }
}
```

nested and innerclasses

- **A class can be declared within another class.**

```
class outer {
        int [ ] nums;
        outer(int [ ] n) {
                nums = n;
        }
        void analyze() {
        inner inob = new inner();
        System.out.println("Minimum: " + inob.min());
        }
        class inner{
                int min(){
                int m = nums[0];
                for(i=1; i<nums.length; i++)
                        if(nums[i] < m)
                                m = nums[i];

                return m;
                }
}
```

nested class

```
class nestedDemo{
    public static void main(String [ ] args) {
        int [ ] x = {3,2,1,5,6,9,7,8};
        outer ob = new outer(x);
        ob.analyze();
    }
}
```

vargs

- **variable length arguments. Sometimes a method can be created that takes a variable no of arguments.**
- **It is specified by three dots ...**

```
class varArgs {
        static void vaTest(int ... v){
                System.out.println("Number of args" + v.length);
                System.out.println("Contents:");
                for(int i=0;i<v.length;i++)
                        System.out.println("args" + i + ":" + v[i]);
                System.out.println();
}
public static void main(String [] args){
                vaTest(10);
                vaTest(1,2,3);
        }
} //   Number of args: 1    contents: arg 0: 10
//   Number of args: 3    contents: arg 0: 1   arg 1: 2       arg 2:  3
```

overloading varargs methods:

```
static void vaTest(int ...v){          }
static void vaTest(boolean ...v) {          }

In main
vaTest(1,2,3);
vaTest(true, false, false);
```