Object Oriented Programming with Java 18IS34

Unit-3

Inheritance

- Here a subclass extends superclass

```
Class TwoDshape{
        double width, height;
        void showDim() {
                System.out.println(“ Width and Height are:”+ width+                “and” + height);
        }
}
class triangle extends TwoDshape{
        String style;
        double area() {     return (width*height)/2;
                        }
        void showstyle(){ System.out.println(“Triangle is” + style);
        }
}// in main initialize width, height, style ... display all values and compute area...
```

Inheritance

- Here a subclass extends superclass

```
class shapes {
    public static void main(String  [ ] args){
    triangle t1 = new triangle();
    t1.width=4.0;      t1.height=4.0;    t1.style=”filled”;
    System.out.println(“Info for t1”);
    t1.showstyle();
    t1.showDim();
    System.out.println(“Area is” + t1.area());
}
```

Inheritance

- Member access and inheritance

```
Class TwoDshape{
    private double width, height;
    double getwidth() { return width;  }
    double getheight() { return height;  }
    void setwidth(double w) {  width = w;  }
    void setheight(double h)  {  height = h;  }
      // void showDim() method to print width and height
}
```

Inheritance

- member access and inheritance

```
class triangle extends TwoDshape{
        String style;
        double area() {     return (getwidth() * getheight())/2;
                        }
// method to display style showStyle()
}
class shape2{
        public static void main(String  [ ] args ){
                Triangle t1 = new Triangle();   Triangle t2 = new Triangle();
                t1.setwidth(4.0);
                t1.setheight(4.0);
                t1.style = “filled”;
        // remaining code.....to make use of all the methods defined.....
}
```

Constructors and Inheritance

```
class TwoDshape{
        .....                   }
// remaining methods.....
}
class triangle extends TwoDshape{
        private string style;
        triangle(String  s, double w, double h ){
                setwidth(w);
                setheight(h);
                style = s;
        // remaining code.....
}
class shape{
        public static void main(String [] args){
                triangle t1 = new triangle(“fluid”,4.0,4.0);
}
```

Using super to call superclass constructors

- A subclass can call a constructor defined by its superclass by use of super.
- super(Parameter list);

Using super to call superclass constructors

```
class twoDshape{
        twoDshape(double w, double h){
                width=w;  height=h;
                //accessor methods are to be included
        }
class triangle extends twoDshape{
        super(w,h);   // call super class constructor
        style = s;
        }
class shape4{
        .
        .
        .
}
```

Using super to call superclass constructors

```
class A{
        int i;
}
class B extends A{
        int i;              //hides i in A
        B(int a, int b) {
                super.i = a;    // i in A
                i=b;            // i in B
                }
        void show(){
                System.out.println("i in superclass" + super.i);
                System.out.println("i in subclass" + i);
        }
}
class usesuper{
        public static void main(String [ ] args) {
                B subob = new B(1,2);
                subob.show();
        }
}
```

Multilevel Hierarchy

- **It is acceptable to use a subclass as a superclass of another**
- **Each subclass inherits all of the traits found in all of its superclasses.**

```
class twoDshape {
        private double width;
        private double height;
        ........
        void showDim()  { System.out.printn("Widht and height are" + width +height);
        }
}
class triangle extends twoDshape {
        private string style;
        ....
        void showstyle() { System.out.println("triangle is " + style);
        }
}
class colortriangle extends triangle{
        private String color;
        void showcolor() {    System.out.println("color is" + color);
        }
}
```

```
class shape{
    public static void main(String [] args){
    colortriangle t1 =  new colortriangle(“Blue”, “outlined”, 8.0,12.0);
    t1.showstyle();
    t1.showDim();
    t1.showcolor();
    System.out.println(t1.area());
}
```

superclass references and subclass objects

- **Java is strongly typed language.**
- **A reference variable for one class type cannot normally refer to an object of another class type.**

```
class X{
        int a;
        X(int i){       a=i; }
        class Y{
        int a;
        Y(int i) { a=i;}
}
}
class Incompatibleref{ public static void main(String [] args){
                                X x = new X(10);
                                X x2;
                                Y y = new Y(5);
                                x2 = x;     //both of same type
                                x2 = y;     // error not of same type;
        }
}
```

Method overriding

- When a method in a subclass has the same return type and signature as a method in its superclass then the method in the subclass is said to override the method in the superclass.

```
class A{
        int i,j;
        A(int a, int b){
        i=a;
        j=b;
        }
        void show()    {
             System.out.println(“i and j” + i + “ ”+ j);
        }
}
```

Method overriding

```
class B extends A {
        int k;
        B(int a, int b, int c) {
                super(a,b);
                k=c;
        }
        void show() {
                System.out.println("k:" + k);
        }
class overrideDemo {
    public static void main(String [] args) {
                B subob = new B(1,2,3);
                subob.show();
        }
}
```

Using Abstract classes

- An abstract method is created by specifying the abstract type modifier.
- An abstract method contains no body and is therefore not implemented by the superclass.

  abstract type name(parameter list);

- It cannot be applied to static methods or to constructors.
- Attempting to create an object of an abstract class by using new will result in a compile time error.

Abstract class

```
abstract class TwoDshape {
    private double width;
    private double height;
    TwoDshape(double w, double h) {
        width = w;
        height = h;
        }
    abstract double area();
}
```

```
class triangle extends TwoDshape{
        private String style;
        triangle(String s, double w, double h) {
                super(w,h);
                style=s;
                }
        double area() {
                return getwidth() * getheight() /2;
        }
}
class Absshape{
        public static void main(String [] args){
                TwoDshape [] shapes = new TwoDshape[1];
                shapes[0] = new triangle(“dotted”, 8.0,12.0);
                System.out.println(“Area is” + shapes[0].area());
}
```

Using final

- **In java it is easy to prevent a method from being overridden or a class from being inherited by using keyword final.**

```
 class A {
            final void meth()   {
                  System.out.println("this is final method"):
            }
      }
class B extends A {
      void meth() {   // Error cant override
      }
}
```

- **final prevents inheritance**

```
 final class A{
      }
      class B extend A {   // error cant subclass A
            //
}
```

final data members , member variables act like constants

```
      final int OUTERR = 0;
```

Object Class

- Java has an implicit class that is superclass of all other classes

Method	Description
Object clone():	creates an object that is same as object being created
boolean equals(Object object):	determines object is equal to another
int hashcode():	returns hashcode associated with the invoking object
void notify():	returns execution of a thread waiting on invoking object
void notifyall():	returns execution of all threads
void wait():	waits on another thread of execution

Interfaces

- An interface is a construct that describes functionally without specifying implementation
- Once an interface is defined, any number of classes can implement it.

creating an interface

```
public interface series {
    int getnext();
    void reset();
    void setstart();
}
```

```
class ByTwos implements series{
        int start;
        int val;
        ByTwos() {
                start =val=0;
                }
        public int getNext() {
        val + = 2;
        return val;
        }
        public void reset() {
        val =start;
        }
        public void setstart(int x) {
                start=x;
                val=x;
        }
        }
```

```
seriesDemo {
    public static void main(String [] args){
        ByTwos ob = new ByTwos();
        for(int i=0;i<2;i++)
        System.out.println(“Next val is” + ob.getNext());
        System.out.println(“reset”);
        ob.reset();
        ob.setstart(100);
        for(int i=0;i<2;i++)
            Syem.out.println(“Next val is” + ob.getNext());
    }
}
```

Using interface references

```
class seriesDemo2 {
        public static void main(String [] args){
                ByTwos twoob = new ByTwos();
                Bythrees threeob = new Bythrees();
                series iRef;
                for(int i=0;i <3;i++){
                        iRef = twoob;
        System.out.println(“Next ByTwo value is “ + iRef.getNext());
        iRef = threeob;
        System.out.println(“Next Bythree value is” + iRef.getNext());
}
}
}
```

implementing multiple interfaces

```
inteface IFA {
        void dosomething();
        }
interface IFB {
        void dosomethingelse();
        }
class myclass implements IFA, IFB {
        public void dosomething() {
                System.out.println(“Doing something”);
        public void dosomethingelse() {
                System.out.println(“Doing something else”);
        }
}
```

Constants in interfaces

```
interface Iconst{
    int MIN = 0;
    int MAX = 10;
    String ERROMSG = “Boundary Error”;
  }
```

Interfaces can be extended

```
interface A {
        void meth1();
        void meth2();
        }
interface B extends A {
        void meth3();
        }
class myclass implements B {
        public void meth1() {
        System.out.println(“Hello meth1”):
        }
        public void meth2() {
        System.out.println(“Hello meth2”):
        }
        public void meth3() {
        System.out.println(“Hello meth3”):
        }
}
```

```
class IFextend {
    public static void main(String [] args) {
        myclass ob = new myclass();
        ob.meth1();
ob.meth2();
ob.meth3();
}
```

Nested interfaces

```
interface A{
    public iterface NestedIf {
    boolean isNotNegative(int x);
}
void dosomething();
}
class B implements A.NestedIf {
    public boolean isNotNegative(int x) {
        return x<0 ? false:true;
    }
}
```