Inheritance and Interfaces

B.Tech. (IT), Sem-5,
Core Java Technology (CJT)

Dharmsinh Desai University
Prof. (Dr.) H B Prajapati

Lecture-6

Inheritance

- Inheritance is a parent-child relationship between classes.
- Inheritance allows code reusability. (advantage)
  - If, we have existing code, we can extend it to change existing behavior.
  - We can also add new behavior. But, not useful in runtime polymorphism.
  - We can override existing behavior. Useful in polymorphism.
- More than code reusability is runtime polymorphism at the language level.

Superclasses and subclasses

- Original code is called
  - Superclass,
  - Base class, or
  - Parent class.
- New class is called
  - subclass,
  - derived class, or
  - child class.
- Example:
  - Base Class: Account
  - Derived class: SavingAccount, CurrentAccount, FDAccount

Polymorphism

- Polymorphism allows us to write generalized code, using general object/data types.
- The real power we get using methods/behaviors
- E.g., Shape class has draw() behavior.
- Derived classes: Square, Circle, Rectangle can also have same behavior draw().
- We can hold all different types of objects in generalized references.

Calling sequence

```
class Box{
    Box(){}
}
class EnBox extends Box{
    EnBox(){}
}
```

- First, superclass' (Box) constructor is called and then subclass' (EnBox) constructor is called

Calling sequence

```
class Box{
    Box(int length, int width, int height){
    }
}
class EnBox extends Box{
    EnBox(int length, int width, int height, int weight){
    }
}
```

- The above has error, as default constructor (Box()) of Base class is not present.

7

Calling sequence

- Two solutions
  - Write default constructor in base class
  - Use super to indicate which constructor of base class we want to call.

8

Calling sequence

- What is the calling sequence when we derive class?
  - Load subclass,
  - Load superclass,
  - initialize static members of superclass,
  - call static block of superclass,
  - initialize static members of subclass,
  - call static block of subclass
  - call constructor of Base class
  - call constructor of Derived class

9

Calling sequence: Example

```
class ClassLoadingBase{
    static int i=initialize();
    static{
        System.out.println("> ClassLoadingBase: static block");
    }
    private static int initialize(){
        System.out.println("> ClassLoadingBase.initialize()");
        return 10;
    }
    ClassLoadingBase(){
        System.out.println("> ClassLoadingBase()");
    }
}
```

10

Calling sequence: Example

```
class ClassLoading extends ClassLoadingBase{
    static int j=initialize();
    static{
        System.out.println("> ClassLoading: static block");
        System.out.println("> ClassLoadingBase.i="+i);
    }
    private static int initialize(){
        System.out.println("> ClassLoading.initialize()");
        return 0;
    }
```

11

Calling sequence: Example

```
    ClassLoading(){
        System.out.println("> ClassLoading()");
    }
    public static void main(String[] args){
        ClassLoading obj=new ClassLoading();
    }
}
```

```
D:\programs\CJT\programs\classes and object>java ClassLoading
> ClassLoadingBase.initialize()
> ClassLoadingBase: static block
> ClassLoading.initialize()
> ClassLoading: static block
> ClassLoadingBase.i=10
> ClassLoadingBase()
> ClassLoading()

D:\programs\CJT\programs\classes and object>
```

12

Use of super keyword

- To call a superclass constructor
- To call a superclass method
- To access superclass data members

13

Use of super keyword

- To call a superclass constructor
  - Suppose, we have Box class and EnhancedBox class, derived from Box (having length, width, and height)
  - Box class has three argument constructor and EnhancedBox has four argument constructor (having length, width, height, and color)
  - We can call constructor of base class from constructor of derived class, as shown below:

```
EnhancedBox(int length, int width, int height, int color){
   super(length, width, height);
   this.color=color;
}
```

14

Use of super keyword

- To call a superclass method
- Suppose, we have a class called Square and DecoratedSquare.
- From draw() method of DecoratedSquare, we would like to first call draw() method or any other method of the base class

```
public void draw(){
   super.draw()
   // logic of derived class
}
```

15

Use of super keyword

- To access superclass data members
- Suppose both base and derived class have same field name.
- We can access field of base class in a derived class using super.

```
super.fieldName;
```

16

Keyword: final

- final class: cannot be extended
- final method: cannot be overridden
- final member: value cannot be changed
- final arguments: argument becomes read-only

17

Keyword: final

- final class: cannot be extended

```
final class Box {
}
```

- Now, the following is not possible

```
class EnBox extends Box{
}
```

18

Keyword: final

- final method: cannot be overridden

```
class Box{
   final void render(){
   }
}
```

- Now, the following is not possible

```
class EnBox extends Box{
   //Overriding is not allowed for final method
   void render(){
   }
}
```

19

Keyword: final

- final member: value cannot be changed

```
class Box{
   final String label="Box";
   Box(){
        // label="box"; //Error, final variable can be
   initialized only once
   }
}
```

20

Keyword: final

- final arguments: argument becomes read-only

```
class Box{
   public void test(final Box b){
        //b=new Box();
        // Error: final parameter cannot be assigned. Read-only
   }
}
```

21

Modifier: abstract

- abstract method: method without body
- abstract class: class' object cannot be created
  - If we write an abstract method in a class, then class has to be declared as abstract.
  - A class can become abstract without having any abstract method.

22

Modifier: abstract

- abstract method: method without body

```
abstract public void draw();
```

- When a class has a single abstract method, the class also becomes abstract

```
abstract class Shape{
   abstract public void draw();
}
```

- Using abstract method, we fix signature of method in the base class. Very useful for runtime polymorphism.
- Abstract means it is not concrete (complete). Something is missing.

23

Modifier: abstract

- Abstract class: class' object cannot be created

```
abstract class Shape{
   abstract public void draw();
}
```

- Now, we cannot write

```
Shape s=new Shape();
```

24

Lecture-7

25

Access modifiers

- Access modifiers
  - private: only within class
  - default: from any class in the same package
  - protected: from any class in the same package, derived class in another package
  - public: from any where

26

Access modifiers: Demo

- Demonstrate using Base, Derived, and AccessModifierDemo classes:

```
class Base{
    private int pBi;
        int dBi;   //It has default access modifier
    protected int ptBi;
    public int pubBi;
}
```

27

Access modifiers: Demo

```
class Derived extends Base{
    private int pDi;
        int dDi; //It has default access modifier
    protected int ptDi;
    public int pubDi;
 public static void main(String [] args){
        Derived d=new Derived();
//          System.out.println("pBi="+d.pBi);
```

28

Access modifiers: Demo

```
        System.out.println("dBi="+d.dBi);
        System.out.println("ptBi="+d.ptBi);
        System.out.println("pubBi="+d.pubBi);
        System.out.println("pDi="+d.pDi);
        System.out.println("dDi="+d.dDi);
        System.out.println("ptDi="+d.ptDi);
        System.out.println("pubDi="+d.pubDi);
    }
}
```

29

Access modifiers: Demo

```
class AccessModifierDemo{
    public static void main(String [] args){
        Derived d=new Derived();
//          System.out.println("pBi="+d.pBi);
        System.out.println("dBi="+d.dBi);
        System.out.println("ptBi="+d.ptBi);
        System.out.println("pubBi="+d.pubBi);
//          System.out.println("pDi="+d.pDi);
        System.out.println("dDi="+d.dDi);
        System.out.println("ptDi="+d.ptDi);
        System.out.println("pubDi="+d.pubDi);
    }
}
```

30

Polymorphism

- Calling methods of derived class using reference of base class
  - PaintBrush example: Shape, Circle, Square, Rectangle, etc.
  - Shape class has draw() method, which is overridden by derived classes

31

Method overriding

- draw() in A
- A <- B <- C
  - B can call A's draw() method using super.draw()
  - C cannot call A's draw() method (C's parent, i.e., B, has changed the behavior of draw(). So B's draw() method is visible to any subclass.

32

Object class

- Two important methods
  - public boolean equals(Object)
    - ob1==ob2 It compares whether two references point to same objects
  - public String toString()

33

Casting object

- Up casting (implicit)
- Down casting (explicit)
- Use of instance of operator

34

Casting object

- Up casting (implicit): in argument
- In equals() method, defined in Object class

```
class Shape{
    public boolean equals(Object o){
        //…
    }
}
```

- How to use?

```
Shape s1=new Shape(); Shape s2=new Shape();
s1.equals(s2);
```

35

Casting object

- Up casting (implicit): Use in return type
- Base class

```
class Shape{
    Shape getShape(){
    }
}
```

- Derived class

```
class Rectangle{
    Shape getShape(){
        //it can return an object of Rectangle
    }
}
```

36

Casting object

- Down casting (explicit): Use in argument

```
public boolean equals(Object o){
    //If we know incoming object is Box, we can downcast it
     Box b=(Box)o;
```

37

Casting object

- Use of instance of operator

```
public void draw(Shape s){
   if(s instanceof Rectangle){
        // perform Rectangle specific operation
   }
   else if(s instanceof Triangle){
        // perform Triangle specific operation
   }

}
```

38

Example: Compare two objects

- Compare two Box objects using Object class' equals() method. Use down casting.

```
class Box{
    private int l,w,h;
    public Box(int l,int w,int h){
        this.l=l;
        this.w=w;
        this.h=h;
    }
```

39

Example: Compare two objects

```
   public boolean equals(Object o){
        Box b=(Box)o;
        if((l==b.l)&&
            (w==b.w)&&
            (h==b.h)
        )
            return true;
        return false;
    }
   public String toString(){
        return "Box: "+l+"X"+w+"X"+h;
   }
}
```

40

Example: Compare two objects

```
class CompareBox{
    public static void main(String[] args){
        Box b1=new Box(10,11,12);
        Box b2=new Box(10,11,13);
        Box b3=new Box(10,11,12);
        System.out.println("Comparing b1="+b1.toString()+" and
b2="+b2);
        if(b1.equals(b2)){
            System.out.println("Both are equal");
        }
        else{
            System.out.println("Both are not equal");
        }
```

41

Example: Compare two objects

```
        System.out.println("Comparing b1="+b1+" and
b3="+b3);
        if(b1.equals(b3)){
            System.out.println("Both are equal");
        }
        else{
            System.out.println("Both are not equal");
        }
    }
}
```

42

Example: Compare two objects

```
Comparing b1=Box: 10X11X12 and b2=Box: 10X11X13
Both are not equal
Comparing b1=Box: 10X11X12 and b3=Box: 10X11X12
Both are equal
```

43

Is-a relation vs has-a relationship

- Inheritance (is-a relationship): Car is-a Vehicle
- Composition (has-a relationship)
  - Containership: Car has four wheels
    - If main is deleted (Car), its parts also get deleted (Wheel)
  - Aggregation: A room has four walls
    - If main is deleted (Room), its parts (Wall) may not get deleted (Wall)
- Inheritance + add methods in derived class (is-like-a relationship)

44

Interface inheritance

- Creating an interface
  - All methods are by default public
  - All members are by default final static
- How to implement interface
  - Interface implemented by more than one class
- How to use interface (using reference of interface)
  - Same reference can point to two different implementations.
- Multiple interface inheritance is possible

interface C extends A, B{}

- Multiple interface implementation is possible

class Airplane implements Vehicle, Flyable{
}

45

Example: interface

- We can create an interface for Stack

```
interface Istack{
    public void push(int element);
    public int pop();
}
```

- Implement interface. Here, Stack is inheriting interface (IStack)

```
class Stack implements IStack{
}
```

- How to use Interface and implementation?

```
IStack stack=new Stack();
```

46

Lecture-8

47

Generic Matrix and Its implementation

- It's a good example to understand various concepts
  - Inheritance
  - Abstract class
  - Runtime polymorphism
  - Abstract method
  - Use of Object datatype

48

Generic Matrix (Defer implementation details)

```
abstract class GenericMatrix{
    Object[][] matrix;
    GenericMatrix(Object[][] matrix){
        this.matrix=matrix;
    }
    public Object[][] addMatrix(Object[][] matrix){
        Object[][] result=new
    Object[matrix.length][matrix[0].length];
        //check size
```

49

Generic Matrix

```
        if((this.matrix.length!=matrix.length) ||
                (this.matrix[0].length!=matrix[0].length)){
                System.out.println("Error: matrices do not have
    the same size");
                System.exit(0);
        }
        //perform addition
        for(int i=0;i<result.length;i++)
                for(int j=0;j<result[i].length;j++)
                        result[i][j]=add(this.matrix[i][j], matrix[i][j]);
        return result;
    }
```

50

Generic Matrix

```
    public Object[][] multiplyMatrix(Object[][] matrix){
        Object[][] result=new
    Object[this.matrix.length][matrix[0].length];
        //check sizes
        if(this.matrix[0].length!=matrix.length){
                System.out.println("Error: matrices do not have
    valid size");
                System.exit(0);
        }
```

51

Generic Matrix

```
        //perform multiplication
        for(int i=0;i<result.length;i++)
                for(int j=0;j<result[i].length;j++){
                        result[i][j]=zero();
                        for(int k=0;k<this.matrix[0].length;k++){
                                result[i][j]=add(result[i][j],multiply(
                                        this.matrix[i][k],
                                                matrix[k][j]));
                        }
                }
        return result;
    }
```

52

Generic Matrix

```
    abstract public Object add(Object o1, Object o2);
    abstract public Object multiply(Object o1, Object o2);
    abstract public Object zero();
    public static void displayMatrix(Object[][] m){
        for(int i=0;i<m.length;i++){
                for(int j=0;j<m[0].length;j++)
                        System.out.print(m[i][j].toString()+" ");
                System.out.println();
        }
    }
}
```

53

Integer Matrix (Concrete Implementation)

```
class IntegerMatrix extends GenericMatrix{
    IntegerMatrix(Integer[][] matrix){
        super(matrix);
    }
    public Object add(Object o1,Object o2){
        Integer i1=(Integer)o1;
        Integer i2=(Integer)o2;
        return new Integer(i1.intValue()+i2.intValue());
    }
```

54

Integer Matrix (Concrete Implementation)

```
public Object multiply(Object o1,Object o2){
    Integer i1=(Integer)o1;
    Integer i2=(Integer)o2;
    return new Integer(i1.intValue()*i2.intValue());
}
public Object zero(){
    return new Integer(0);
}
}
```

55

Use Integer Matrix (Concrete Implementation)

```
class TestIntegerMatrix{
  public static void main(String[] args){
    Integer[][] m1=new Integer[3][3];
    Integer[][] m2=new Integer[3][3];
    //initialize two matrices
    for(int i=0;i<m1.length;i++)
        for(int j=0;j<m1[0].length;j++){
            m1[i][j]=new Integer(i);
            m2[i][j]=new Integer(i+j);
        }
```

56

Use Integer Matrix (Concrete Implementation)

```
    //create an instance of IntegerMatrix
    IntegerMatrix im=new IntegerMatrix(m1);
    //perform integer matrix addition and multiplication
    Object[][] addResult=im.addMatrix(m2);
    Object[][] mulResult=im.multiplyMatrix(m2);

    //display m1, m2, addResult, and mulResult
    System.out.println("m1=");
    IntegerMatrix.displayMatrix(m1);
```

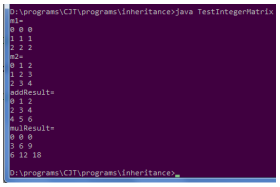
57

Use Integer Matrix (Concrete Implementation)

```
    System.out.println("m2=");
    IntegerMatrix.displayMatrix(m2);
    System.out.println("addResult=");
    IntegerMatrix.displayMatrix(addResult);
    System.out.println("mulResult=");
    IntegerMatrix.displayMatrix(mulResult);
  }
}
```

58

Use Integer Matrix (Concrete Implementation)



59