

CSE 220 : OOP with Java

Inheritance

Instructor

Name: Dr. Md. Mahbubur Rahman

Inheritance

3

What is Inheritance?

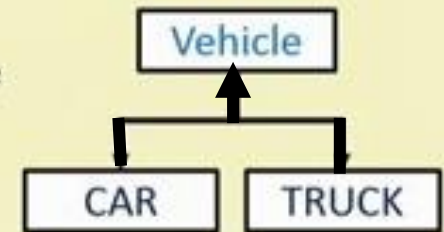
Inheritance is a mechanism in which one class acquires the property of another class. For example, a child inherits the traits of his/her parents.



General to Specific

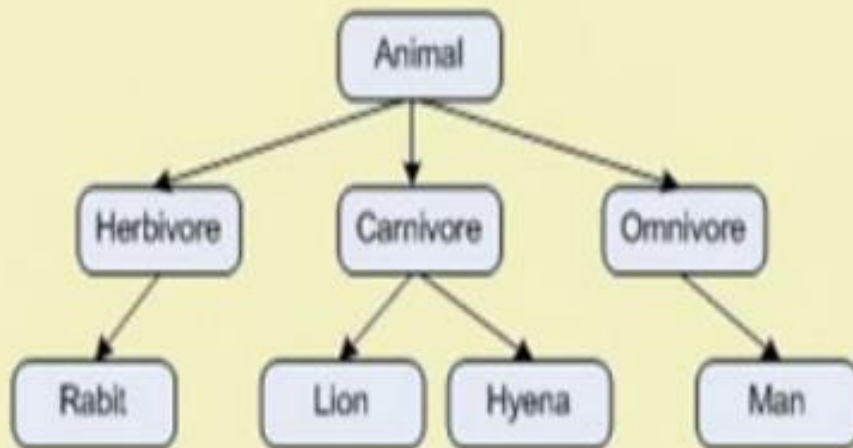
4

- Inheritance is one of the cornerstone of object-oriented programming because it allows the creation of hierarchical classification.
- Using inheritance, one can create a general class that include some common set of items.
- This class then can be used to create more specific classes which has all the items from the base class, in addition to some items of its own.

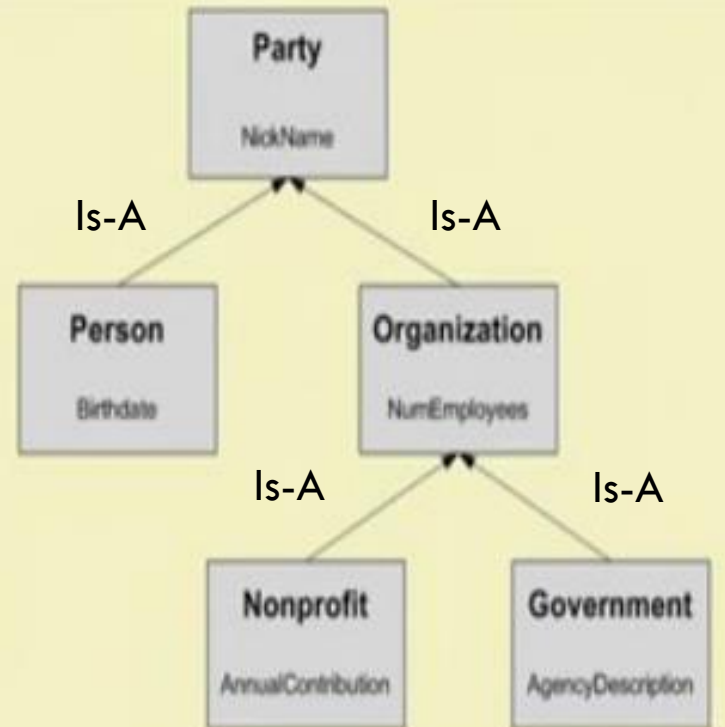


Inheritance in Java

5



Single multi-level inheritance



Class hierarchy

Superclass and subclass relationship

6

- **Superclass:** A class that is inherited is called a superclass.
- **Subclass:** The class that does inheriting is called a subclass.
 - A subclass is a specialized version of a superclass
 - It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements (i.e., variables and methods)
- **Reusability:** It is a mechanism which facilitates you to reuse the data and methods of the existing class when one create a new class.
 - One can use the same data and methods already defined in the previous class.



Inheritance in Java

7

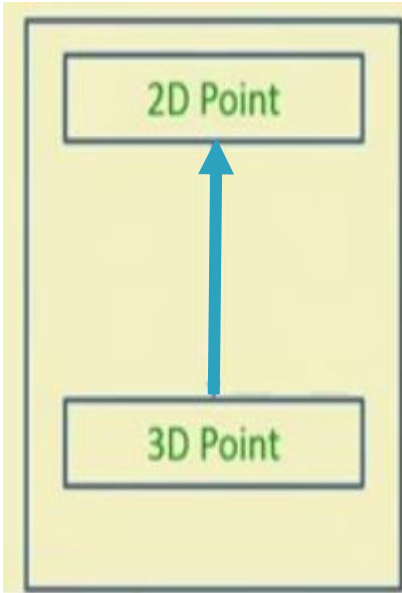
- The **extends** keyword is used to define a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

```
class Class_Name [extends SuperClassName ]  
                  [ implements Interface ] {  
                                          [declaration of member elements ]  
                                          [ declaration of methods ]  
}
```

Those are included with [...] are optional. Member elements will be declared with usual convention as in C++ with the following syntax :

Inheritance in Java

8



```
class Point2D
{
    int x; int y;

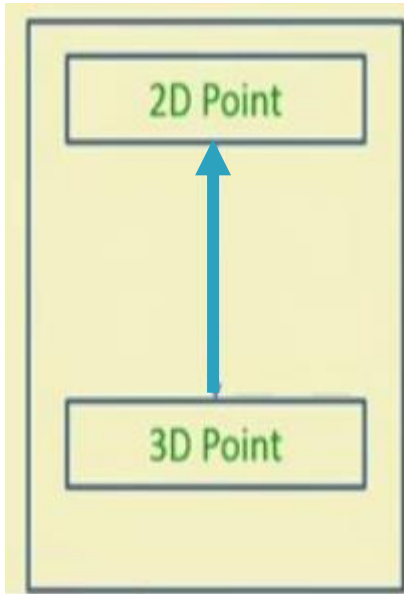
    void setValue(int i, int j) {
        x=i; y=j; }

    void display() {
        System.out.println("x= "+ x + "    y= "+y); }

    void who() {
        System.out.println("From Base Class"); }
}
```


Inheritance in Java

9



```
class Point3D extends Point2D  
{
```

```
    int z;
```

```
    void setValue(int i, int j, int k) {  
        set i, j, z = k;
```

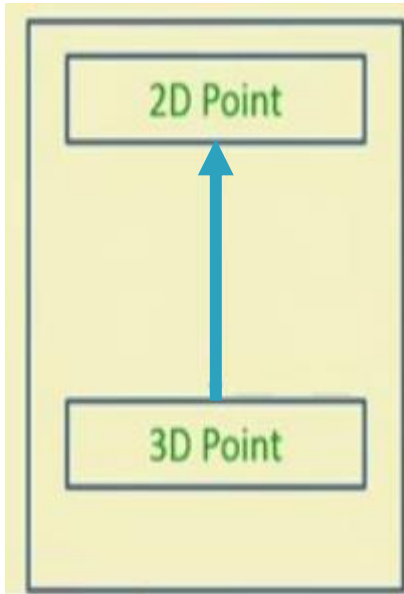
*super(i, j);
z = k;*

```
    void display() {  
        System.out.println("x= " + x + "    y= " + y + "    z= " + z);  
    }
```

```
    void who() {  
        System.out.println("From Derived Class"); }  
}
```

Inheritance in Java

10



```
public class Test_Inheritance
{
    public static void main(String[] args)
    {
        Point2D p= new Point2D();
        Point3D q= new Point3D();

        p.setValue(10,10);
        q.setValue(20,30,40);

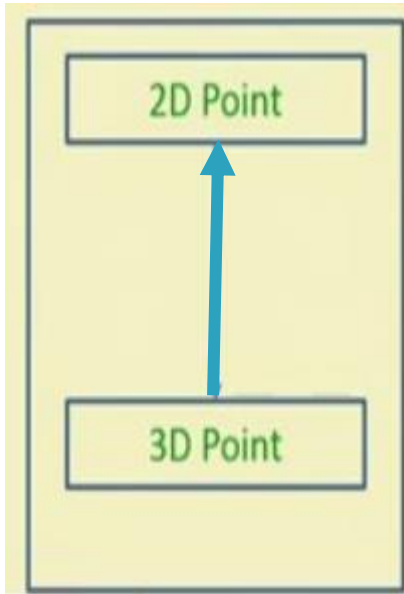
        p.display();
        q.display();

        p.who();
        q.who();

    }
}
```

Inheritance in Java

11



run:

x= 10 y= 10

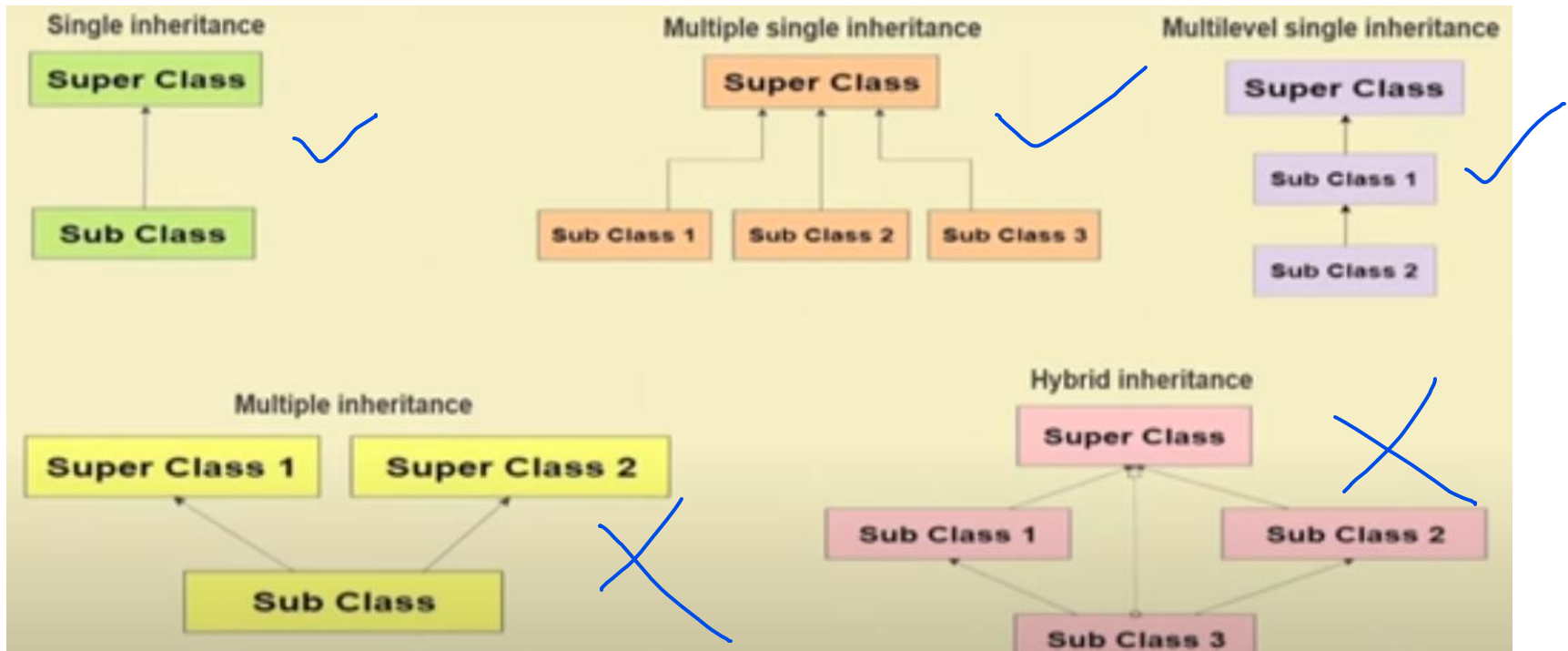
x= 20 y= 30 z= 40

From Base Class

From Derived Class

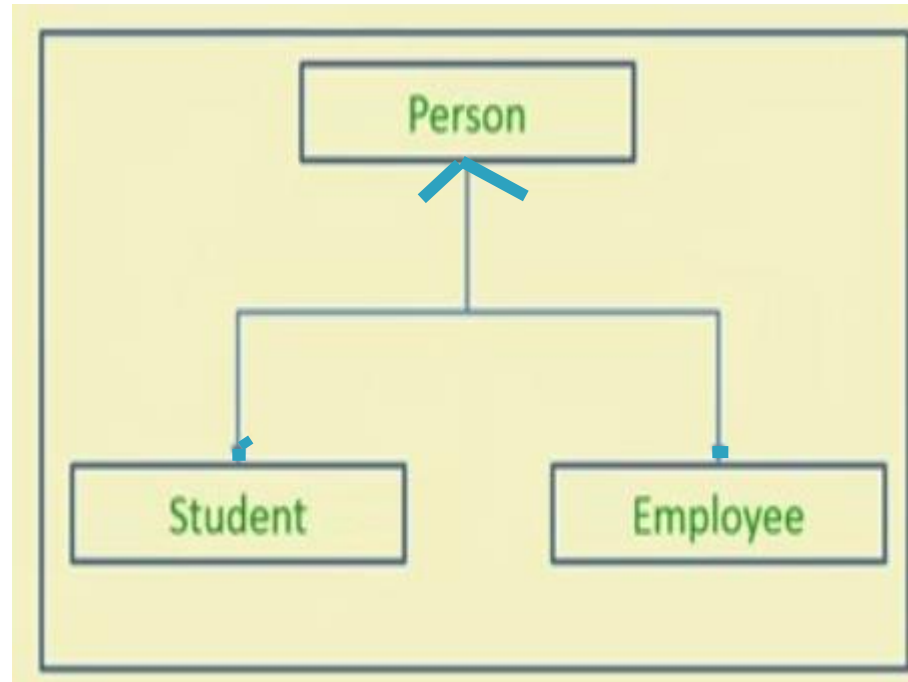
Inheritance Types in Java

12



Simple Inheritance

13



Single Inheritance

14

```
class Person
{
    String name;
    int birthYear;

    public Person(String nm, int byear)
    {
        name = nm;
        birthYear = byear;
    }

    public void show_data()
    {
        System.out.println( "name= "+name+" Birth Year= "+birthYear);
    }
}
```

Single Inheritance

15

```
class Student extends Person
{
    private String dept;

    public Student(String nm, int byear, String dt)
    {
        super(nm, byear);
        dept = dt;
    }

    public void show_data()
    {
        System.out.println( "name= "+name+" Birth Year= "+ birthYear+" Dept=
        "+ dept);
    }
}
```

Single Inheritance

16

```
//Employee.java
class Employee extends Person
{
    private int salary;

    public Employee(String nm, int byear, int sal)
    {
        super(nm, byear);
        salary = sal;
    }

    public void show_data()
    {
        System.out.println( "name= " + name + " Birth Year= " + birthYear + "
Salary= " + salary);
    }
}
```


Single Inheritance

17

```
//PersonTester.java
public class PersonTester
{
    public static void main(String[] args)
    {
        Person a = new Person("Rakib",2000);
        Student b = new Student("Nafis",1998,"CSE");
        Employee c = new Employee ("Faisal", 1990,50000);
        a.show_data();
        b.show_data();
        c.show_data();
    }
}
```

run:

name= Rakib Birth Year= 2000

name= Nafis Birth Year= 1998 Department= CSE

name= Faisal Birth Year= 1990 Salary= 50000

Method Overriding

18

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

- The method must have the same name as in the parent class
- The method must have the same parameter as in the parent class.
- There must be an IS-A relationship (inheritance).

Method Overriding

19

```
public class Override_Test//class Point2D and
                        //Point3D previously defined
```

```
{
```

```
    public static void main(String[] args)
    {
```

```
        Point2D p= new Point2D();
        Point3D q= new Point3D();
        p.setValue(10,10);
        q.setValue(20,30,40);
```

```
        p.display();
```

```
        q.display(); //q has another inherited
                    //display method
```

```
        p.who();
```

```
        q.who();
```

```
        Point2D x=(Point2D) q; // cast q to an instance of class Point2D
        x.display();
```

```
}
```

Method
Overloading

Method
Overriding

This is basically up casting; that means, q is a point of 3D, but we can cast into 2D using this kind of special features

run:

x= 10 y= 10

x= 20 y= 30 z= 40

From Base Class

From Derived Class

x= 20 y= 30 z= 40)

Method Overriding: C++

20

```
class Base
{
    ... ..
public:
    void getData(); ←-----
    {
        ... ..
    }
};

class Derived: public Base
{
    ... ..
public:
    void getData(); ←-----
    {
        ... ..
    }
};

int main()
{
    Derived obj;
    obj.getData(); -----
}
```

Function call

This function will not be called

Method Overriding and Overloading: Java

21

Methods of superclass
defined in subclass
with same name and
signature

Overriding

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}
class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }

    public void bark(){
        System.out.println("bowl");
    }
}
```

Same Method Name,
Same parameter

Multiple methods with
same name within a
class

Overloading

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }

    //overloading method
    public void bark(int num){
        for(int i=0; i<num; i++)
            System.out.println("woof ");
    }
}
```

Same Method Name,
Different Parameter

Accessing super class members and methods from subclass

22

- A sub class object can reference a super class variable or method if it is not overridden.
- A super class object cannot reference a variable or method which is explicit to the sub class object.

Super keyword

23

The **super** keyword in **Java** is a reference variable which is used to refer immediate parent class members.

Whenever you create an instance of a sub class, an instance of its parent class is created implicitly, which is referred by **super** keyword.

Super keyword

24

Usage of Super Keyword

1

Super can be used to refer immediate parent class instance variable.

2

Super can be used to invoke immediate parent class method.

3

super() can be used to invoke immediate parent class constructor.

Super keyword

25

1) super is used to refer immediate parent class instance variable.

```
class Animal{
    String color="white";
}
class Dog extends Animal{
    String color="black";
    void printColor(){
        System.out.println(color);//prints color of Dog class
        System.out.println(super.color);//prints color of Animal class
    }
}
class TestSuper1{
    public static void main(String args[]){
        Dog d=new Dog();
        d.printColor();
    }
}
```

black
white

Super keyword

26

2) super can be used to invoke parent class method

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
void bark(){System.out.println("barking...");}
void work(){
    super.eat();
    bark();
}
}
class TestSuper2{
public static void main(String args[]){
    Dog d=new Dog();
    d.work();
}}
```

eating...
barking...

Super keyword

27

3) super is used to invoke parent class constructor.

```
class Animal{
    Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
    Dog(){
        super();
        System.out.println("dog is created");
    }
}
class TestSuper3{
    public static void main(String args[]){
        Dog d=new Dog();
    }
}
```

animal is created
dog is created

Super keyword:real use

28

```
class Person{
    int id;
    String name;
    Person(int id,String name){
        this.id=id;
        this.name=name;
    }
}
```

```
class Emp extends Person{
    float salary;
    Emp(int id,String name,float salary){
        super(id,name);//reusing parent constructor
        this.salary=salary;
    }
    void display(){System.out.println(id+" "+name+"
"+salary);}
}
```

```
class TestSuper{
    public static void main(String[] args){
        Emp e1=new Emp(1,"Babar",50000f);
        e1.display();
    }
}
```

Runtime Polymorphism in Java

29

Dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile-time. Also, it is called Runtime polymorphism.

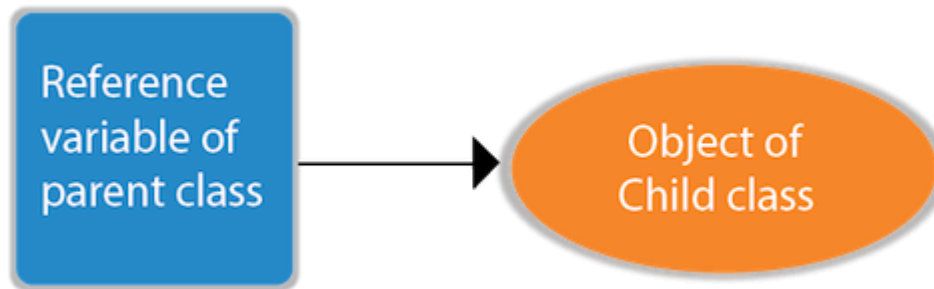
In this process, an overridden method is called through the reference variable of a super class. The determination of the method to be called is based on the object being referred to by the reference variable.

Runtime Polymorphism in Java

30

Upcasting

If the reference variable of Parent class refers to the object of Child class, it is known as upcasting. For example:



```
class A{  
class B extends A{
```

```
A a=new B();//upcasting
```

Runtime Polymorphism in Java

31

Upcasting

```
class Bike{
    void run(){System.out.println("running");}
}
class Splendor extends Bike{
    void run(){System.out.println("running safely
with 60km");}

    public static void main(String args[]){
        Bike b = new Splendor();//upcasting
        b.run();
    }
}
```

running safely with 60km.

Runtime Polymorphism in Java

32

```
class A {  
    void callMe ( ) {  
        System.out. println ( "I am from A " ) ;  
    }  
}  
class B extends A {  
    void callMe ( ) {  
        System.out.println ( "I am from B " );  
    }  
}  
public class Who {  
    public static void main(String args [ ] ) {  
        A a = new B ( ) ;  
        a.callMe ( ) ;  
    }  
}
```

I am from B
I am from B

Access Specifier

33

Public : Member elements and methods can be marked as public and then they can be accessed from any other method in Java Programs.

The public modifier can be applied to classes as well as methods and variables. It then allows to make a class accessible to other classes in other Packages.

The public access specification is automatic, in the sense that, if no access specifier is mentioned then **by default it is having public accessibility.**

Access Specifier

34

Private : Member elements and methods marked private can be used only marked private can be used only **from inside their class**. A private element / method is not visible in any other class, including subclasses . Also, a subclass cannot override a non-private method and make the new method private.

Protected : Member elements and methods marked protected can be used only **from inside their class** or **in subclasses of that class**. **A subclass can still override a protected method or variable.**

Access Specifier

35

| Modifiers | Within Same Class | Within same package | Outside the package- (Subclass) | Outside the package- (Global) |
|------------------|-------------------|---------------------|-------------------------------------|-------------------------------|
| Public | Yes | Yes | Yes | Yes |
| Protected | Yes | Yes | Yes (only to <u>derived class</u>) | No |
| Default | Yes | Yes | No | No |
| Private | Yes | No | No | No |

Access Specifier

36

`/* Example-1 : Private access modifier */`

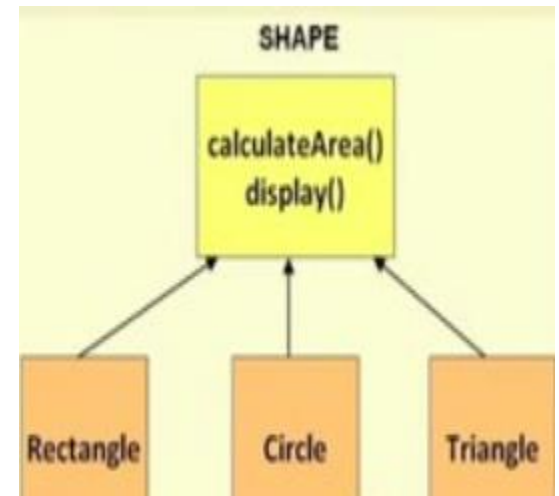
```
public class A{
    private int data = 40;
    public void msg(){
        System.out.println("Class A: Hello Java!");
    }
}
```

```
public class Demonstration {
    public static void main(String args[]){
        A obj = new A();           //OK : Class A is public
        System.out.println(obj.data); //Compile Time Error : data is private
        obj.msg();                 //OK : msg is public
    }
}
```

Abstract class

37

- **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.
- Abstraction lets you focus on what the object does instead of how it does it.
- A class which is declared with the **abstract** keyword is known as an **abstract class** in Java. It can have abstract and non-abstract methods (i.e., method with the body only without its definition).

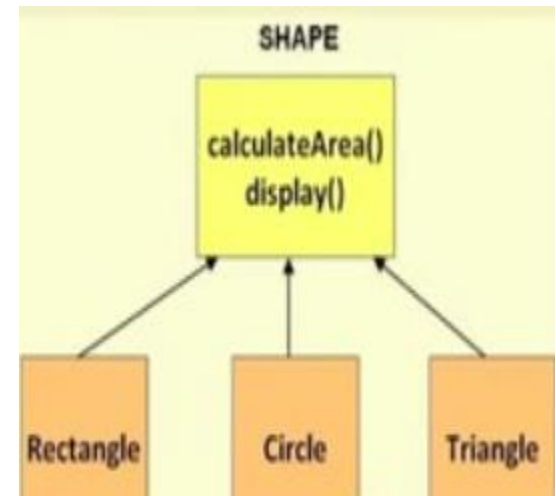


Abstract class

38

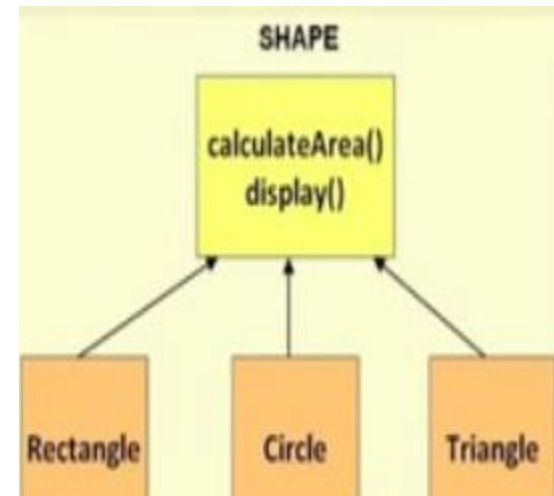
```
// An example abstract class in Java
abstract class Shape {
    int color;

    // An abstract function (like a pure virtual
    function in C++)
    abstract void draw();
}
```



Abstract class

```
39 abstract class Base {  
    abstract void fun();  
}  
class Derived extends Base {  
    void fun() { System.out.println("Derived fun()  
called"); }  
}  
class Main {  
    public static void main(String args[]) {  
  
        // Uncommenting the following line will  
        // cause compiler error as the  
        // line tries to create an instance of  
        // abstract class.  
        // Base b = new Base();  
  
        // We can have references of Base type.  
        Base b = new Derived();  
        b.fun();  
    }  
}
```



Abstract class

40

Points to remember

- An **abstract class** must be declared with an **abstract** keyword.
- It can have abstract and non-abstract **methods**.
- It **cannot be instantiated**.
- It can have constructors and static methods also.
- It can have **final methods** which will force the sub class not to change the body of the method.

final keyword

41

The **final** keyword in Java is used to restrict the access of an item from its super class to a sub class. The Java **final** keyword can be used in many context.

- Variable : a variable cannot be accessed in sub class
- Method : a method cannot called from a sub class object
- Class : a class cannot be sub classed.

final keyword

42

// Final Class Inheritance An Example

```
final class Bike{
```

```
class Honda1 extends Bike{
```

```
    void run(){
```

```
        System.out.println("Running safely with 100kmph");
```

```
    }
```

```
}
```

```
final class Demonstration_611 {
```

```
    public static void main(String args[]){
```

```
        Honda1 honda = new Honda1();
```

```
        honda.run();
```

```
    }
```

```
}
```

References

43

Courtesy:

Object-Oriented Programming with C++ and Java- Debasis Samanta and course slides

References:

The Complete Reference Java 2: Herbert Schildt

Web materials