Inheritance in Java

MODULE NAME: OBJECT ORIENTED PROGRAMMING WITH JAVA

MODULE CODE: CST 05206

INSTRUCTORS: JAMILA L.

Introduction

- Inheritance is a mechanism that enables one class to inherit all the attributes and behavior of another class.
- Through inheritance, a class immediately has all the functionality of an existing class.
- Because of this, you must define only how the new class is different from an existing class.
- Inheritance is one of the primary features of OOP — which is a form of software reuse.
- A class that is inherited is called superclass
- The class that inherits is called subclass
- A subclass is a specialized version of a superclass

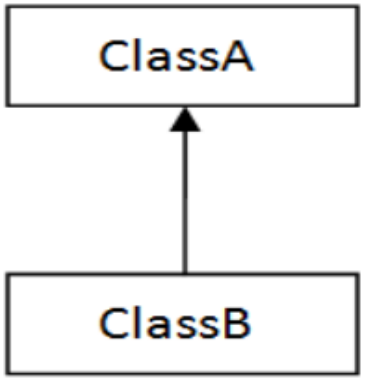
Introduction cont…

extends – ("inherits from") keyword is used to inherit superclass

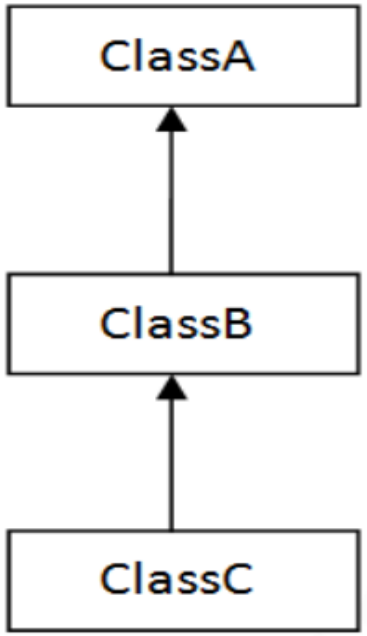
- Java does not support multiple super-classes into single subclass (this differs from C++)
- A subclass can be a superclass of another subclass

Types of inheritance

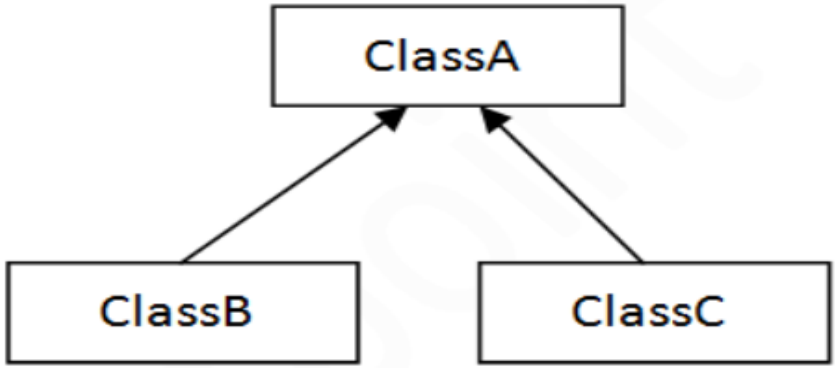
On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.



1) Single



2) Multilevel



3) Hierarchical

Inheritance example

//create superclass

```
public class A {
    int x,y;
    void showXY(){
        System.out.println("x and y: "+x+ " "+y);
    }
}
```

Inheritance example cont...

```
//create subclass by extending class A
class B extends A{
    int z;
    void showZ(){
        System.out.println("k: "+z);
    }
    void sum(){
        int sum=x+y+z;
        System.out.println("sum: "+sum);
    }
}
```

Inheritance example cont…

```
class ABTest{
    public static void main(String[] args){
        A superObj=new A();
         B subObj=new B();
         //The superclass may be used by itself
         superObj.x=10;
         superObj.y=20;
         System.out.println("Contents of the superObj");
         superObj.showXY();
         System.out.println();
```

Inheritance example cont…

```
//The subclass has access to all public members of its superclass
        subObj.x=7;
           subObj.y=8;
           subObj.z=9;
           System.out.println("Contents of the subObj");
           subObj.showXY();
           subObj.showZ();
           System.out.println();
           System.out.println("The sum of x, y, and z in subObj");
           subObj.sum();
    }
}
```

Inheritance and member access

- A subclass cannot access those members of the superclass that have been declared as private.
- A class's private members are accessible only from within the class itself.
- A subclass can change the state of private superclass instance variables only through non-private methods provided in the superclass and inherited by the subclass.

Inheritance and member access cont...

```
//Example:
public class X {
    int number1, number2;
    private int number3;
void setNumber3(int number){
    number3=number;
}
int getNumber3(){
    return number3;
}
}
```

Inheritance and member access cont...

```
class Y extends X{
    void sum(){
        int sum=number1+number2+super.getNumber3();
    /*error - number3 is not directly accessible,
instead use get method as done here!*/
        System.out.println(sum);
    }
}
```

Inheritance and member access cont...

```
class XYTest{
    public static void main(String[] args){
        Y y=new Y();
        y.number1=4;
        y.number2=8;
        y.setNumber3(9);
        y.sum();
    }
}
```

Box example

A Box class has three instance variables width, height and depth and one member method volume.

- It also has different formats of constructor methods to initialize the instance variables, such as
  - Through parameters w, h and d
  - No parameter
  - Single parameter length of each side
- A WeightBox class that has all the properties of Box with an additional instance variable weight.
- To initialize this variable, it needs constructor
- Another class ColouredBox has all the properties of Box with an additional property color.
- Write down the structures of these three classes (ColouredBox left as an assignment)

The use of keyword super

- If a superclass keeps its data members private, then there would be no way for subclass to directly access or initialize its parent's instance variables
- The solution is the use of keyword super
- Whenever a subclass needs to refer to its immediate superclass, it can do so by the use of the keyword super
- Super has two general forms,
  - For accessing constructors
  - For accessing members of superclass

Using super for calling superclass constructor

- super (argument-list);
- super( ) must always be the first statement to be executed inside a subclass' constructor
- Example on next slide demonstrates how to call superclass's constructor in the subclass.

Box example program

```
//superclass Box
public class Box {
    float length,width,depth;
    public Box(float l,float w,float d){
        length=l;
        width=w;
        depth=d;
    }
  float volume(){
          return length*width*depth;
     }
  }//end superclass Box
```

Box example program cont…

```
class WeightBox extends Box{
    float weight;
    WeightBox (float l, float w, float d, float wt){
        //initializing constructor of subclass using super
        super(l,w,d);
        weight=wt;
    }
   float volumeWeightBox(){
            return super.volume();
        }
   }//end subclass WeightBox
```

Box example program cont…

```
class BoxTest{
    public static void main(String[] args){
        Box eb=new Box(5,4,3);
        WeightBox wb=new WeightBox (5,4,3,12);
        System.out.println("Empty box details");
        System.out.println("The volume is "+eb.volume());
        System.out.println("\nWeighing box details");
        System.out.println("The volume is "+wb.volumeWeightBox());
        System.out.println("The weight is "+wb.weight);
 }//end method main
}//end class BoxTest
```

A second use of keyword super

- Here super acts as somewhat like this.
- Mostly applicable when member names of a subclass hide members by the same name in the superclass.
- This practice is also known as variable shadowing.
- Shadowing refers to the practice of using two variables with the same name within scopes that overlap.
- The higher-level scope is hidden because the variable with lower-level scope overrides it.

Method overriding

- When a method in a subclass has the same name and type signature as a method in its superclass, then the method in the subclass is said to override the method in the superclass.
- When an overridden method is called from within a subclass, it will always refer to the version of that method defined by the subclass.

Method overriding cont…

```
public class Number {
    int number1, number2;
    void showNumbers(){
        //display number1 and number2
        System.out.println("number1 and number2: "+number1+" "+number2);
    }
}
```

Method overriding cont…

```
class MoreNumber extends Number{

    int number3;

    /*display number3 - this overrides
       showNumbers() in Number*/

    void showNumbers(){

    System.out.println("number3: "+number3);

    }

}
```

Method overriding cont…

```
class NumberTest{
    public static void main(String[] args){
        MoreNumber mn=new MoreNumber();
        mn.number1=1;
        mn.number2=2;
        mn.number3=4;
        mn.showNumbers();
    }
}
```

Method overriding cont…

- Accessing superclass version of overridden method using super keyword.
- E.g.,

```
void showNumbers(){
    super.showNumbers();/*this solves the
problem of overriding*/
     System.out.println("number3: "+number3);
 }
```

Using final to prevent overriding

- Methods declared as final cannot be overridden

```
class A {
  final void meth() {
    System.out.println("This is a final method.");
  }
}
class B extends A {
  void meth() { // ERROR! Can't override.
    System.out.println("Illegal!");
  }
}
```

Using final to prevent overriding cont…

- Normally Java resolves calls to methods dynamically, at run time (late binding)
- Since, final methods cannot be overridden, a call to one can be resolved at compile time (early binding).

Overriding Vs. Overloading

- Method overriding occurs only when the names and the type signatures of the two methods are identical.
- Method overloading occurs when the names of two or more methods are the same within the class but different signatures.

Java Method Overloading example

```
class OverloadingExample{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
```

Java Method Overriding example

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
}
```

Using final keyword to prevent inheritance

- final keyword before a class declaration prevents it from being inherited.
- Declaring a class as final implicitly declares all of its methods as final
- Therefore, final class does not promote software reuse.

```
final class A {
  // ...
}
// The following class is illegal.
class B extends A { // ERROR! Can't subclass A
  // ...
}
```

Thank you!

for listen