Computer programming (1)

Chapter 5

Inheritance

Agenda

- Inheritance
  - Introduction
  - Types of inheritance
  - Superclasses and Subclasses
  - Method overloading and Method overriding
  - Invoking Superclass constructor
  - Relationship between Superclasses and Subclasses
  - Final Classes and Methods
  - The protected Access Specifier

Inheritance

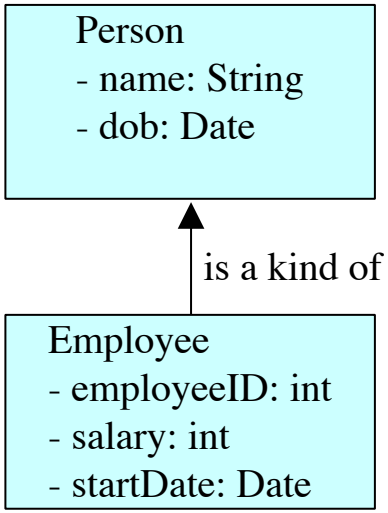
- Inheritance is a fundamental object-oriented design technique used to create and organize reusable classes
- *Inheritance* allows a software developer to derive a new class from an existing one
- The existing class is called the *parent class,* or *super class*, or *base class*
- The derived class is called the *child class* or *subclass*
- As the name implies, the **child inherits** characteristics of the parent

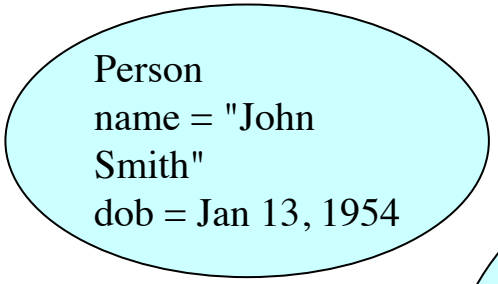
Inheritance

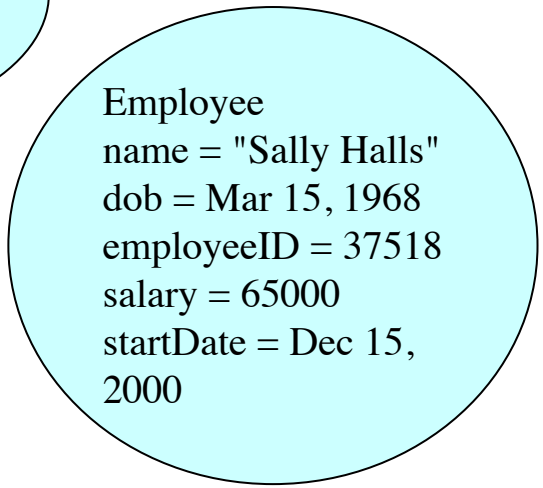
- A programmer can tailor a derived class as needed by adding new variables or methods, or by modifying the inherited ones
- One benefit of inheritance is software reuse

What really happens?

- In this example, we can say that an Employee "is a kind of" Person.
  - An Employee object inherits all of the attributes, methods and associations of Person







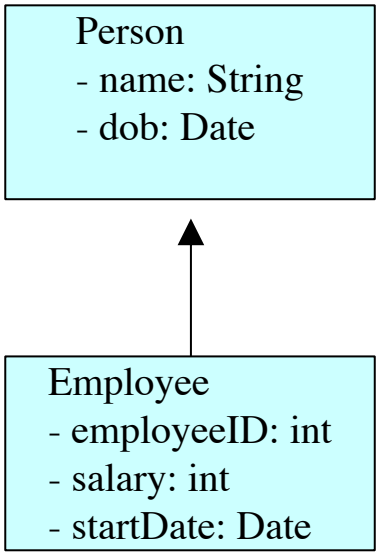
Inheritance in Java

- Inheritance is declared using the "extends" keyword
  - If inheritance is not defined, the class extends a class called Object

```
public class Person
{
   private String name;
   private Date dob;
   [...]
```

```
public class Employee extends Person
{
   private int employeID;
   private int salary;
   private Date startDate;
   [...]
```

```
Employee anEmployee = new Employee();
```



Inheritance

A derived class (subclass, child class) *extends* a base class (superclass , parent class).. It inherits all of its methods (behaviors) and attributes (data) and it may have additional behaviors and attributes of its own.

Base class

class A
Base class attributes
Base class methods

Derived class

class B extends A
attributes inherited from base Additional attributes
methods inherited from base Additional methods

extends Keyword

- **extends** is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.

```
class Super{
.....
.....
}

class Sub extends Super{
.....
.....

}
```

Example of Deriving Subclasses

- See Words.java
- See Book.java
- See Dictionary.java

```java
//*******************************************************************
//  Book.java       Author: Lewis/Loftus
//
//  Represents a book. Used as the parent of a derived class to
//  demonstrate inheritance.
//*******************************************************************

public class Book
{
   protected int pages = 1500;

   //-----------------------------------------------------------------
   //  Pages mutator.
   //-----------------------------------------------------------------
   public void setPages (int numPages)
   {
      pages = numPages;
   }

   //-----------------------------------------------------------------
   //  Pages accessor.
   //-----------------------------------------------------------------
   public int getPages ()
   {
      return pages;
   }
}
```

```java
//*******************************************************************
//  Dictionary.java       Author: Lewis/Loftus
//
//  Represents a dictionary, which is a book. Used to demonstrate
//  inheritance.
//*******************************************************************

public class Dictionary extends Book
{
   private int definitions = 52500;

   //-----------------------------------------------------------------
   //  Prints a message using both local and inherited values.
   //-----------------------------------------------------------------
   public double computeRatio ()
   {
      return (double) definitions/pages;
   }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Definitions mutator.
   //-----------------------------------------------------------------
   public void setDefinitions (int numDefinitions)
   {
      definitions = numDefinitions;
   }

   //-----------------------------------------------------------------
   //  Definitions accessor.
   //-----------------------------------------------------------------
   public int getDefinitions ()
   {
      return definitions;
   }
}
```

```java
//*******************************************************************
//  Words.java       Author: Lewis/Loftus
//
//  Demonstrates the use of an inherited method.
//*******************************************************************

public class Words
{
   //-----------------------------------------------------------------
   //  Instantiates a derived class and invokes its inherited and
   //  local methods.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Dictionary webster = new Dictionary();

      System.out.println ("Number of pages: " + webster.getPages());

      System.out.println ("Number of definitions: " +
                          webster.getDefinitions());

      System.out.println ("Definitions per page: " +
                          webster.computeRatio());
   }
}
```

```
//*****************************************************************
//  Words.java
//
//  Demonstrates
//*****************************************************************

public class Words
{
   //-----------------------------------------------------------------
   //  Instantiates a derived class and invokes its inherited and
   //  local methods.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Dictionary webster = new Dictionary();

      System.out.println ("Number of pages: " + webster.getPages());

      System.out.println ("Number of definitions: " +
                          webster.getDefinitions());

      System.out.println ("Definitions per page: " +
                          webster.computeRatio());
   }
}
```

Output

```
Number of pages: 1500
Number of definitions: 52500
Definitions per page: 35.0
```

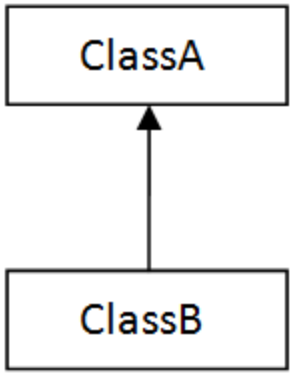
Types of inheritance

The types of inheritance:

- Single Inheritance
- Multiple Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

1-Single Inheritance

When a class extends another class(Only one class) then we call it as **Single inheritance.** The below diagram represents the single **inheritance in java** where **Class B** extends only one class **Class A.** Here **Class B** will be the **Sub class** and **Class A** will be one and only **Super class**.



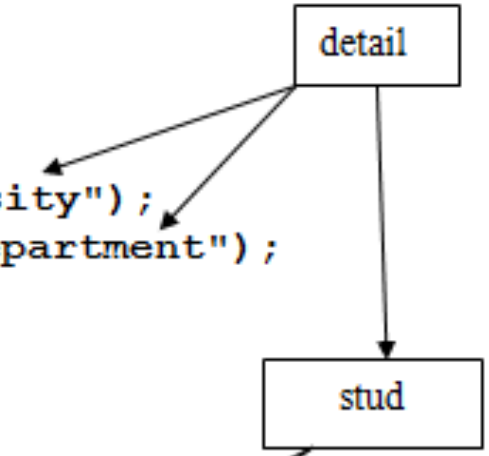
Example 1:Single Inheritance

```
Class A
{
   public void methodA()
   {
     System.out.println("Base class method");
   }
}

Class B extends A
{
   public void methodB()
   {
     System.out.println("Child class method");
   }
   public static void main(String args[])
   {
     B obj = new B();
     obj.methodA(); //calling super class method
     obj.methodB(); //calling local method
  }
}
```

Example 2:Single Inheritance

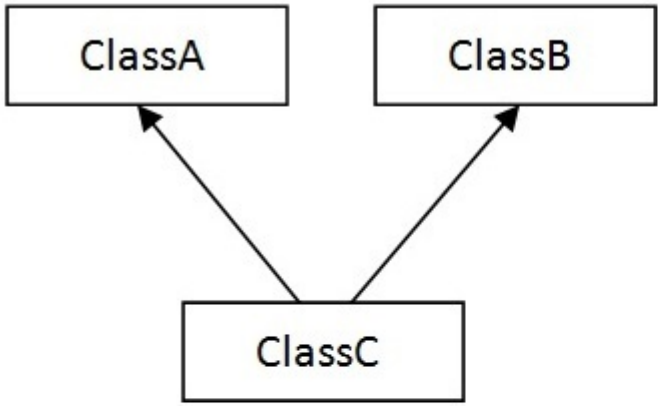
Example for single Inheritance

```
Class detail
{
   public void det()
   {
      System.out.println("Aljouf University");
System.out.println("Computer Science department");

   }
}

Class stud extends detail
{
   public void std()
   {
      System.out.println("Regno:3321234");
System.out.println("Name: Abdullah");
   }

   public static void main(String args[])
   {
      stud s = new stud();
      s.det(); //calling super class method
      s.std(); //calling local method
  }
}
```



2. Multiple Inheritance

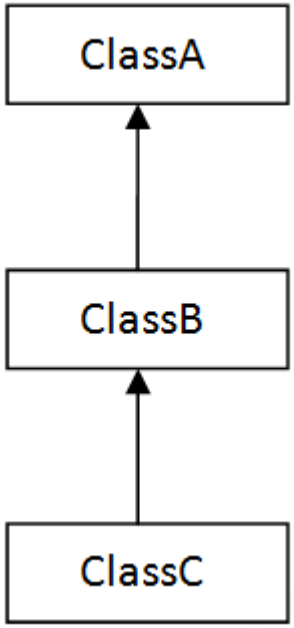
- "**Multiple Inheritance**" refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with "multiple inheritance" is that the derived class will have to manage the dependency on two base classes.



- Note :
  - Most of the new OO languages like **Java and C# do not support Multiple inheritance.** Multiple Inheritance is supported in C++.

3.Multilevel Inheritance

- **Multilevel inheritance** refers to a mechanism in OO technology where one can inherit from a derived class, there by making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of B and B is a child class of A.



- It's pretty clear with the diagram that in Multilevel inheritance there is a concept of grand parent class. If we take the example of above diagram then class C inherits class B and class B inherits class A which means B is a parent class of C and A is a parent class of B. So in this case class C is implicitly inheriting the properties and method of class A along with B that's what is called multilevel inheritance.

Example 1: Multilevel Inheritance

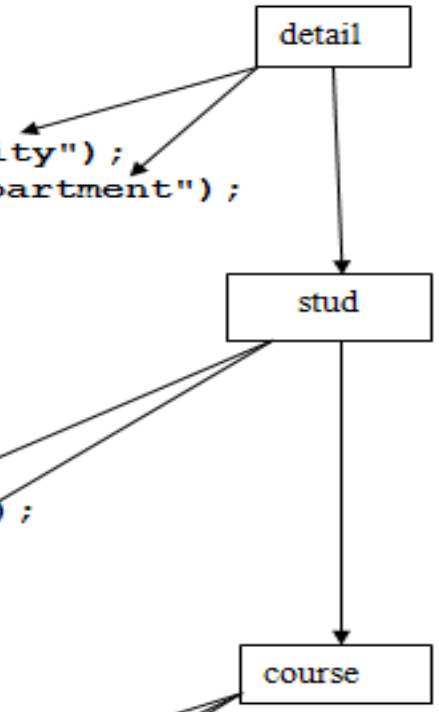

```
Class X    //Super class
{
   public void methodX()
{
     System.out.println("Class X method");
   }
}
```

```
Class Y extends X
{
     public void methodY()
     {
       System.out.println("class Y method");
      }
}
```

```
Class Z extends Y{
   public void methodZ()   {
     System.out.println("class Z method");
   }
   public static void main(String args[])   {
     Z obj = new Z();
     obj.methodX(); //calling grand parent class method
     obj.methodY(); //calling parent class method
     obj.methodZ(); //calling local method  }
}
```

Example Multilevel Inheritance

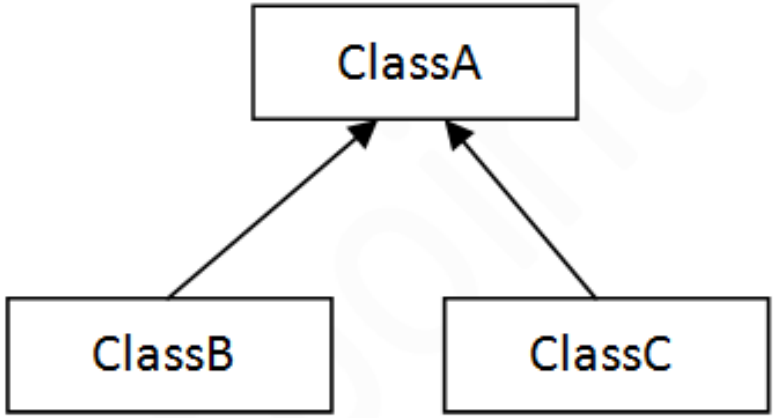
```
Class detail
{
   public void det()
   {
     System.out.println("Aljouf University");
System.out.println("Computer Science department");

   }
}

Class stud extends detail
{
   public void std()
   {
     System.out.println("Regno:3321234");
System.out.println("Name: Abdullah");
   }
}

Class course extends stud
{
public void crs()
   {
     System.out.println("CSC 101");
      System.out.println("CSC102");
      System.out.println("CSC 104");
   }
   public static void main(String args[])
   {
     course c = new course();
     c.det(); //calling grand parent(details) class
method
     c.std(); //calling parent(stud) class method
     c.crs(); //calling local(course) method


  }
}
```



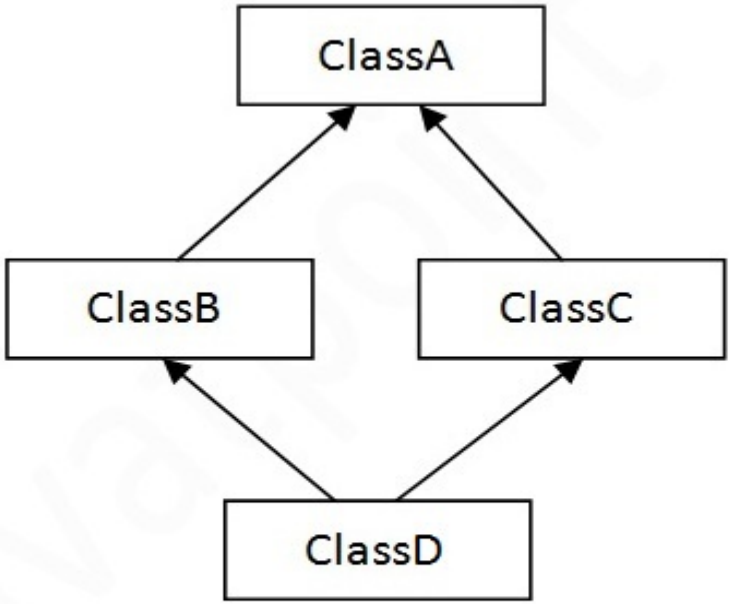
Example3:Multilevel Inheritance

```
Inheritance program

class Car{
        public Car()
        {
                System.out.println("Class Car");
        }
        public void vehicleType()
        {
                System.out.println("Vehicle Type: Car");
        }
}
class Toyota extends Car{
        public Toyota ()
        {
                System.out.println("Class Toyota ");
        }
        public void brand()
        {
                System.out.println("Brand: Toyota");
        }
        public void speed()
        {
                System.out.println("Max: 300Kmph");
        }
}
public class LandCruiserV8 extends Toyota {

         public LandCruiserV8()
         {
                 System.out.println("Toyota Model: Land Cruiser V8");
         }
         public void speed()
                {
                        System.out.println("Max: 200Kmph");
                }
         public static void main(String args[])
         {
                 LandCruiserV8 obj=new LandCruiserV8();
                 obj.vehicleType();
                 obj.brand();
                 obj.speed();
         }
}
```

4. Hierarchical Inheritance in java

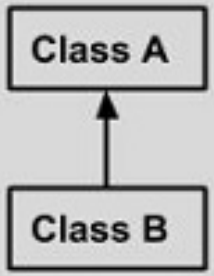


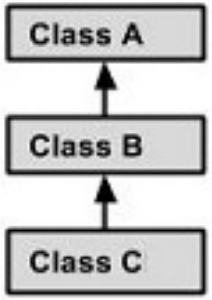
5. hybrid inheritance

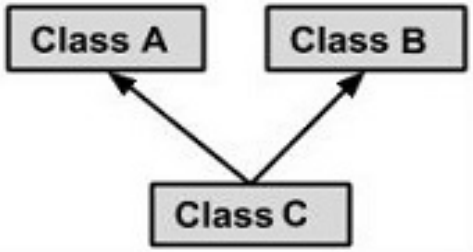


5) Hybrid

Conclusions

Single Inheritance
 
 	
	public class A {
	.....
	}
	public class B **extends** A {
	.....
	}

Multi Level Inheritance  	public class A { .....} public class B **extends** A {.....} public class C **extends** B {..... }

Multiple Inheritance
 
 	
	public class A { .....}
	public class B {.....}
	public class C **extends** A,B {
	.....
	} // Java does not support mutiple Inheritance

Subclasses & Constructors

- Note:
  - A subclass inherits all the members (fields, methods, and nested classes) from its superclass. Constructors are not members, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass

The super keyword

- The **super** keyword is similar to **this** keyword .
- Following are the scenarios where the super keyword is used.
  - It is used to **differentiate the members** of superclass from the members of subclass, if they have same names.
  - It is used to **invoke the superclass** constructor from subclass.

Differentiating the members

- If a class is inheriting the properties of another class and if the members of the superclass have the names same as the sub class, to differentiate these variables we use super keyword as shown below.
  - super.variable
  - super.method();

Method Overloading

- Method Overloading is a feature that allows a class to have two or more methods having same name, if their argument lists are different

- We discussed constructor overloading that allows a class to have more than one constructors having different argument lists.

- **Argument lists could differ in :**
  1. Number of parameters.
  2. Data type of parameters.
  3. Sequence of Data type of parameters.

Example 1:Method Overloading: changing no. of arguments

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
        public static void main(String[] args){
                System.out.println(Adder.add(11,11));
                        System.out.println(Adder.add(11,11,11));
}}
```

Out put:
22
33

Example 2:Method Overloading: changing data type of arguments

```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b){return a+b;}
}
class TestOverloading2{
        public static void main(String[] args){
                System.out.println(Adder.add(11,11));
                        System.out.println(Adder.add(12.3,12.6));
}
```

Out put:
22
24.9

Quick Check – case 1

If we have:

```
int     mymethod(int  a,   int  b)

float   mymethod(int  var1,   int  var2)
```

What will happen?

Quick Check – case 1

```
int     mymethod(int a,   int b)

float   mymethod(int var1,   int var2)
```

Result: Compile time error. Argument lists are exactly same. Even though return type of methods are different, it is not a valid case. Since return type of method doesn't matter while overloading a method.

Quick Check – case 2

If we have:

```
float   mymethod(int a,   float b)

float   mymethod(float var1,   int var2)
```

What will happen?

Quick Check – case 2

```
float mymethod(int a, float b)

float mymethod(float var1, int var2)
```

Result: Perfectly fine. Valid case for overloading. Sequence of the data types are different, first method is having **(int, float)** and second is having **(float, int)**

Method Overriding

- If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java.**
- The benefit of overriding is: ability to define a behavior that's specific to the subclass type, which means a subclass can implement a parent class method based on its requirement.
- Rules for Java Method Overriding
  - method must have same name as in the parent class
  - method must have same parameter as in the parent class
  - must be inheritance relationship

Example1:


```
class Animal {
    public void move() {
        System.out.println("Animals can move");
    }
}
class Dog extends Animal {
    public void move() {
        System.out.println("Dogs can walk and run");
    }
}
public class TestDog {
    public static void main(String args[]) {
        Animal a = new Animal();    // Animal reference and object
        Animal b = new Dog();       // Animal reference but Dog object
        a.move();                   // runs the method in Animal class
        b.move();                   // runs the method in Dog class
    }
}
```

Out put:

Animals can move

Dogs can walk and run

Method Overriding

- In the above example, you can see that even though **b** is a type of Animal it runs the move method in the Dog class. The reason for this is: In compile time, the check is made on the reference type. However, in the runtime, JVM figures out the object type and would run the method that belongs to that particular object.

- Therefore, in the above example, the program will compile properly since Animal class has the method move. Then, at the runtime, it runs the method specific for that object.

Example2: with error occurred


```
class Animal {
   public void move() {
      System.out.println("Animals can move");
   }
}
class Dog extends Animal {
   public void move() {
      System.out.println("Dogs can walk and run");
   }
   public void bark() {
      System.out.println("Dogs can bark");
   }
}
public class TestDog {
   public static void main(String args[]) {
      Animal a = new Animal();      // Animal reference and object
      Animal b = new Dog();         // Animal reference but Dog object

      a.move();                     // runs the method in Animal class
      b.move();                     // runs the method in Dog class
      b.bark();
   }
}
```

Out put:

This program will throw a compile time error since b's reference type Animal doesn't have a method by the name of bark.

Method Overriding

- This program will throw a compile time error since b's reference type Animal doesn't have a method by the name of bark.

Method overloading vs. Method overriding in java

- **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...");}
}
```

Example of Method Overriding with super keyword

- Super Class

```
public class Super_Class {

int num = 20;

public void display(){

  System.out.println("This is the display method of
                the Super class");

}

}
```

Example of Method Overriding with super keyword

- Subclass

```
public class Sub_Class extends Super_Class {

  int num = 10;
  public void display(){
       System.out.println("This is the display method of
                          the Sub class");
  }
  public void my_method(){
       display();
       super.display();
       System.out.println("Value of the variable named num "
                                + "in Sub Class is "+num);
       System.out.println("Value of the variable named num "
                                + "in Super Class is +super.num);
  }

public static void main(String[] args){
Sub_Class obj = new Sub_Class();
obj.my_method();}
}
```

Output:

```
package SuperKey;

public class Sub_Class extends Super_Class {

    int num = 10;
    public void display(){
        System.out.println("This is the display method of the Sub class");
    }
    public void my_method(){
        display();
        super.display();
        System.out.println("Value of the variable named num "
```

```
<terminated> Sub_Class [Java Application] C:\Program Files (x86)\Java\jdk1.8.0\jre\bin\javaw.exe (22 nov. 2017 23:42:48)
This is the display method of the Sub class
This is the display method of the super class
Value of the variable named num in Sub Class is 10
Value of the variable named num in Super Class is 20
```

Invoking Superclass constructor

- If a class is inheriting the properties of another class, the subclass automatically acquires the default constructor of the super class.
- But if you want to call a parametrized constructor of the super class, you need to use the super keyword as shown in the next slide.

Example of Invoking Superclass constructor

```
class Superclass{

    int age;

    Superclass(int age){
        this.age=age;
    }

    public void getAge(){
        System.out.println("The value of the variable named age in super class is: " +age);
    }

}

public class Subclass extends Superclass {

    Subclass(int age){
        super(age);
    }

    public static void main(String argd[]){
        Subclass s= new Subclass(24);
        s.getAge();
    }

}
```

Result:
The value of the variable named age in super class is: 24

Private Members in a Superclass

- A subclass does not inherit the private members of its parent class.
- However, if the superclass has **public** or **protected** methods for accessing its private fields, these can also be used by the subclass. [The subclass 'has' the fields of its superclass, but does not have access to them directly.]
- A nested class has access to all the private members of its enclosing class—both fields and methods. Therefore, a public or protected nested class inherited by a subclass has indirect access to all of the private members of the superclass.

The protected Modifier

- Visibility modifiers affect the way that class members can be used in a child class
- Variables and methods declared with private visibility cannot be referenced by name in a child class
- They can be referenced in the child class if they are declared with public visibility -- but public variables violate the principle of encapsulation
- There is a third visibility modifier that helps in inheritance situations: protected

The protected Modifier

- The protected modifier allows a child class to reference a variable or method directly in the child class
- It provides more encapsulation than public visibility, but is not as tightly encapsulated as private visibility
- A protected variable is visible to any class in the same package as the parent class

The protected Access Modifier

- This modifier can be applied to both instance variables and methods.
- Variables and methods declared protected are accessible from the classes defined in the same package and also from subclasses which are defined in other packages.

Relation between a Super Class and Sub Class

- A subclass has an 'is a' relationship with its superclass.
- This means that a sub class is a special kind of its super class.
- When we talk in terms of objects, a sub class object can also be treated as a super class object.
  - And hence, we can assign the reference of a sub class object to a super class variable type. However, the reverse is not true. A reference of a super class object may not be assigned to a sub class variable.

Example :

```
Account a = new SavingAccount(clientName, AccountNo, balance,
  interset);

SavingAccount s = new Account(clientName, AccountNo, balance);
  //error
```

Final Classes and Methods

- Inheritance is useful features in Java. But it may be desired that a class should not be extensible by other classes to prevent exploitation (for security reasons). I
- In Java, we use the final keyword_to prevent some classes from being extended.
- A class declared as final **cannot be extended** while
- A method declared as final **cannot be overridden** in its subclasses.

Final Classes and Methods

- A final class can be a subclass but not a super class.

```
final public class A {
    //code
}
```

OR

```
public final class A {
    //code
}
```

Final Classes and Methods

- Final methods are also declared in a similar way

```
public final void someMethod() {
    //code
}
```

What You Can Do in a Subclass

- A subclass inherits all of the *public* and *protected* members of its parent, no matter what package the subclass is in. If the subclass is in the same package as its parent, it also inherits the *package-private* members of the parent. You can use the inherited members as is, replace them, hide them, or supplement them with new members:
  - The inherited fields can be used directly, just like any other fields.
  - You can declare a field in the subclass with the same name as the one in the superclass, thus *hiding* it (not recommended).
  - You can declare new fields in the subclass that are not in the superclass.
  - The inherited methods can be used directly as they are.
  - You can write a new *instance* method in the subclass that has the same signature as the one in the superclass, thus *overriding* it. [@Override]
  - You can write a new *static* method in the subclass that has the same signature as the one in the superclass, thus *hiding* it.
  - You can declare new methods in the subclass that are not in the superclass.
  - You can write a subclass constructor that invokes the constructor of the superclass by using the keyword super.

References

- http://www.cloudbus.org/~raj/254/Lectures/Lecture10.pdf
- http://www.tutorialspoint.com/java/java_inheritance.htm
- https://docs.oracle.com/javase/tutorial/java/landl/subclasses.html
- http://www.javawithus.com/tutorial/relation-between-a-super-class-and-a-class

+

Thanks!