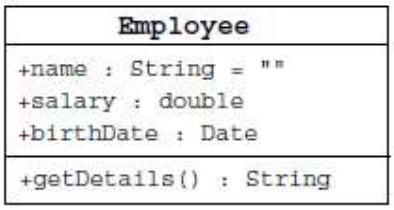
1. Inheritance and Overriding Methods

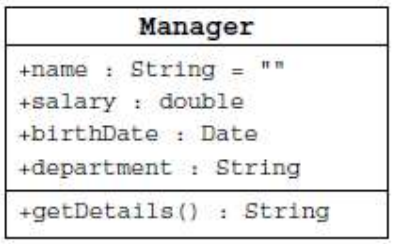
The ***Employee*** class is shown here:




```
public class Employee {
  public String name = "";
  public double salary;
  public Date birthDate;

  public String getDetails() {...}
}
```

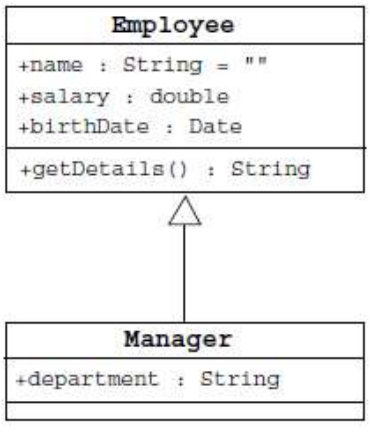
The ***Manager*** class is shown here:




```
public class Manager {
  public String name = "";
  public double salary;
  public Date birthDate;
  public String department;

  public String getDetails() {...}
}
```

Class Diagrams for Employee and Manager Using Inheritance




```
public class Employee {
  public String name = "";
  public double salary;
  public Date birthDate;

  public String getDetails() {...}
}
```

```
public class Manager extends Employee {
  public String department;
}
```

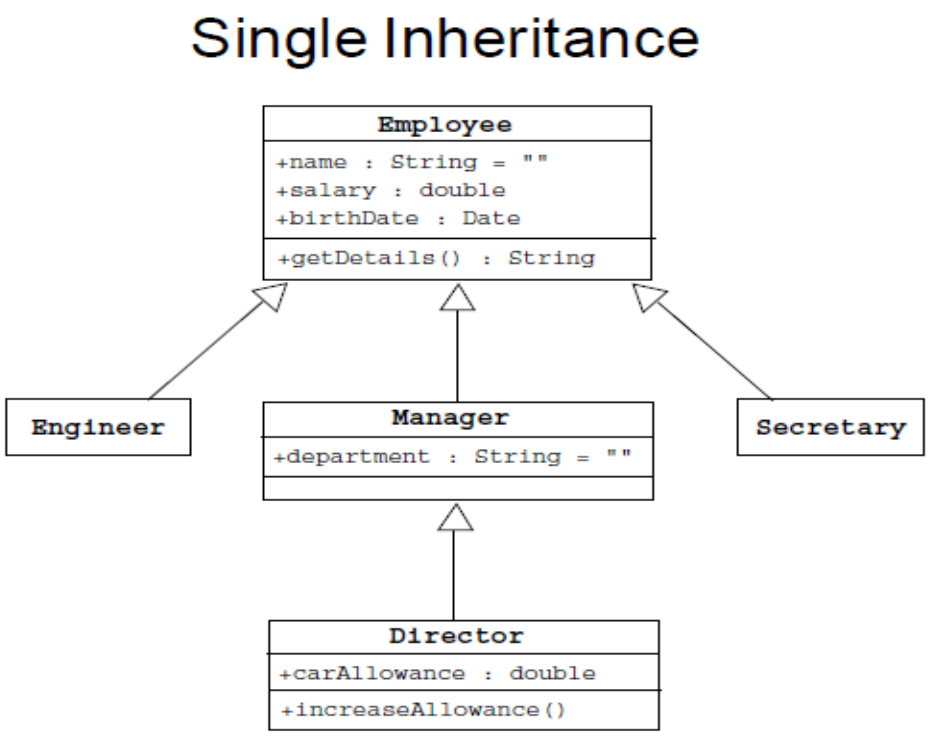
Inheritance is one of the feature of Object-Oriented Programming (OOPs). Inheritance allows a class to use the properties and methods of another class. In other words, the derived class inherits the states and behaviors from the base class. The derived class is also called subclass and the base class is also known as super-class. The derived class can add its own additional variables and methods. These additional variable and methods differentiates the derived class from the base class.

1. Inheritance is a compile-time mechanism. A super-class can have any number of subclasses. But a subclass can have only one superclass. This is because Java does not support multiple inheritance.
2. The superclass and subclass have "is-a" relationship between them.

Single Inheritance

- When a class inherits from only one class, it is called single inheritance.
- **Interfaces** provide the benefits of multiple inheritance without drawbacks.
- Syntax of a Java class is as follows:

***<modifier>* class *<name>* [extends *<superclass>*] {**
<declaration>*
}



A subclass can modify behavior inherited from a parent class.
• A subclass can create a method with different functionality than the parent's method but with the same:

• Name
• Return type
• Argument list

Then the new method is said to *override* the old one.

So, what is the objective of subclass?

Example

```
Public class Employee {

protected String name;
protected double salary;
protected Date birthDate;
```

```
public String getDetails() {
return “Name: “ + name + “\n” +
 “Salary: “ + salary;
}
}
public class Manager extends Employee {

protected String department;

//override method
Public String getDetails() {

return “Name: “ + name + “\n” + “Salary: “ + salary + "\n" +
 “Manager of: “ + department;
}
}
```

The Manager class has a getDetails method by definition because it inherits one from the “Employee class”. However, the original has been replaced, or overridden by the child class ‘version.

Overridden Methods Cannot Be Less Accessible

```
Public class Parent {
Public void doSomething() {      }
}

Public class Child extends Parent {
Private void doSomething() {} // illegal
  }


public class UseBoth {
public void doOtherThing() {
Parent p1 = new Parent();
Parent p2 = new Child();// polymorphism تعددية الاشكال
p1.doSomething();
p2.doSomething();// illegal because of visibility
 }
}
```

The Java programming language semantics dictate that ***p2.method ()*** results in the child version of the method being executed, because the method is declared private, p2 (declared as parent) cannot access it , this the semantics language is violated

2. Invoking Overridden Methods

A subclass method may invoke a ***superclass method*** using the *super* keyword:

- The keyword *super* is used in a class to refer to its superclass.
- The keyword super is used to refer to the members of superclass, both data attributes and methods.
- Behavior invoked does not have to be in the superclass; it can be further up in the hierarchy.

```
public class Employee {
private String name;
private double salary;
private Date birthDate;

public String get Details() {
return "Name: " + name + "\nSalary: " + salary;
}
}
public class Manager extends Employee {
private String department;

public String getDetails() {
// call parent method
return super.getDetails()
+ “\nDepartment: " + department;
}
}
```

Important notes

1. Final class cannot be subclass.
2. Final method cannot be overridden.
3. Final variable such as a constant and any attempt to change the value of a final variable causes a compiler error.

More Example: Case#1 using Super and Sub Class

```
package Inheritance;
public class Transportation {
      /**
       * @param args Mehdi
       */
      protected static int x=12;
      private int y=19;

      public static  void meth1(){
      System.out.println("Calling Method in Super Class");
            }
      private static void meth2(){
      System.out.println("Doesn't Call because of
      Private");
            }}
```

Another Class **extends the Transportation** Class

```
package Inheritance;
public class Cars extends Transportation{
      /**
       * @param args
       */
      public static void main(String[] args) {
      // TODO Auto-generated method stub
   meth1();// don't need an object ref. here >>>> Why????
   System.out.println("The variable x from Super Class is
     "+x);
   meth2(); // This method is error because has private
     modifier
   System.out.println(" The variable Y from Super
     Class"+y);
      }
}
```

Example: Case#2: Call Method from Sub_Class

```
public class A {
      protected int a=9;
      }

class B extends A{
      void test (){
            int a=22;
            System.out.println("The value from SubClass
a="+this.a);
      }
      }

public class C {
      /**
       * @param args
       */
      public static void main(String[] args) {
            // TODO Auto-generated method stub
            B Sub_Class=new B();
            Sub_Class.test();
      }
}
```

Example: Case#3: Call Method from Sub_Class and using (this) KeyWord

```
public class A {
      protected int a=9;
      }

class B extends A{

      void test (){
            //int a=22;
            System.out.println("The value from SubClass
a="+this.a);
      }
      }
```

In this case, the class attribute is not defined in class B. So, we need to inheritance the value of the parent class, which is (9).

```
package Inhertiance;
public class C {
	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		B Sub_Class=new B();
		Sub_Class.test();
	}
}
```

Example: Case#4: Call Method from Sub_Class and using (Super) KeyWord

```
public class A {
	protected int a=9;
	void test_method()
	{
		System.out.println("The Super Class Method");
	}
	}

class B extends A{
	void test1 (){
		super.test_method();
		test2();
		}
	void test2(){
		System.out.println("The method in SupClass");
	}
	}


package Inhertiance;
public class C {
	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		B m=new B();
		m.test1();
	}
}
```

```
// A class to display the attributes of the vehicle
class Vehicle {
   String color;
   int speed;
   int size;

   void attributes() {
      System.out.println("Color : " + color);
      System.out.println("Speed : " + speed);
      System.out.println("Size : " + size);
   }
}

// A subclass which extends for vehicle
class Car extends Vehicle {
   int CC;
   int gears;
   void attributescar() {
      // The subclass refers to the members of the superclass
      System.out.println("Color of Car : " + color);
      System.out.println("Speed of Car : " + speed);
      System.out.println("Size of Car : " + size);
      System.out.println("CC of Car : " + CC);
      System.out.println("No of gears of Car : " + gears);
   }
}
public class Test {
   public static void main(String args[]) {
      Car b1 = new Car();
      b1.color = "Blue";
      b1.speed = 200 ;
      b1.size = 22;
      b1.CC = 1000;
      b1.gears = 5;
      b1.attributescar();
   }
}
Color of Car : Blue
Speed of Car : 200
Size of Car : 22
CC of Car : 1000
No of gears of Car : 5
```