Tema 11: Herencia

Definición

- Cuando inicialmente se modelan dos cosas y se tienen datos duplicados y además pudiera darse que el número de métodos también es similar así como la implementación, se requiere crear una clase a partir de una ya existente dando lugar al manejo de herencia.

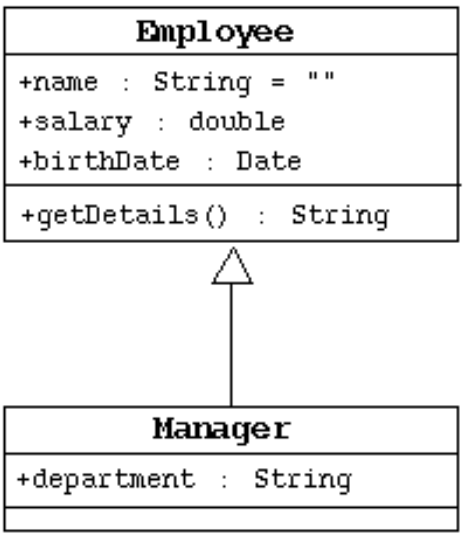
```
public class Employee {
  public String name = "";
  public double salary;
  public Date birthDate;

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

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

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

extends




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

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

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

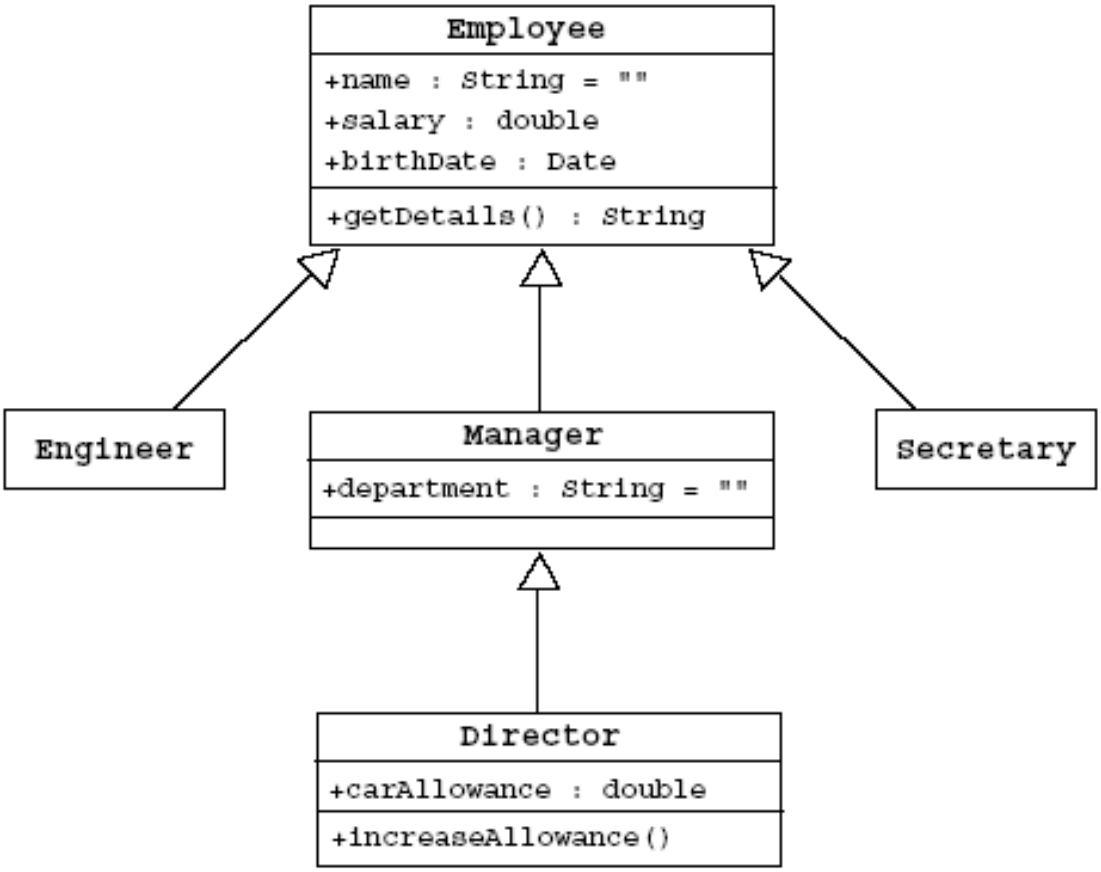
- Sintaxis

  class B extends A { ...}

  Se “lee” B hereda de A e indica que A es superclase de b y B es una **subclase** de A.

Herencia simple

- El lenguaje Java sólo permite heredar una sola clase, esto recibe el nombre de *single inheritance*.



Características

- La subclase puede ocupar todos los atributos de la superclase y tiene la obligación de redefinir los métodos heredados.

Relación “is-a” y/o “has-a”

- Cuando se modelan las clases usadas en un programa se deberán tener en cuenta las relaciones que pudieran darse para lograr el comportamiento deseado entre objetos.

- Para comprobar la correcta relación ente clases es usar la “prueba” “es-un” o “tiene-un” en la descripción de una clase para así saber si se usa herencia o no respectivamente.

“is-a” y/o “has-a”

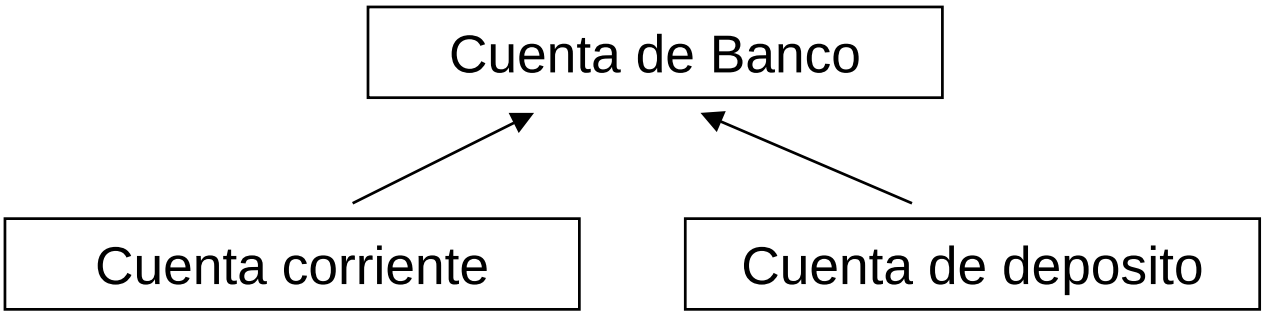
Relación entre clases	Prueba	Código usado en Java
Herencia	es-un	extends
Composición (agregación)	tiene-un o consiste-de	new

- Ejemplo:
  - » Manzana es una fruta o fruta **puede ser** una manzana.
  - » Auto tiene una llanta o Llanta **es parte de** un auto.

Ejemplo “is-a” y/o “has-a”

- De acuerdo a la siguiente especificación modelar la relación:

  Una cuenta de banco describe el nombre de una persona, su dirección, número de cuanta y balance actual. Hay dos tipos de cuenta: corriente y de depósito. Los deudores tienen que avisar con una semana de anticipación si van a retirar de una cuenta de deposito, pero la cuenta acumula intereses.



- Clasificación de:
  1. Casa, puerta, techo, vivienda;
  2. Persona, hombre mujer;
  3. Auto, caja de velocidades, motor;
  4. Vehículo, auto, autobús.
  5. Triángulo, rectángulo, pentágono, línea, polígono, punto.

Polimorfismo

- Una variable es polimórfica porque puede ser referida a objetos de diferentes formas.
- Esto quiere decir que un objeto puede ser creado con la referencia de su padre.
- Cuando hay herencia hay sobreescritura y por lo tanto hay polimorfismo.
- Ejemplos:
  - ✓ Padre e= new Hija();
  - ✓ Hija m = new Hija();
  - X Hija w= new Padre();

Ejemplo

```
public class Padre{
      public static String getClase {
          return “Clase Padre”;
      }
       public static String getObjeto {
          return “Objeto de clase Padre”;
      }
 }

 public class Hija extends  Padre{
      public static String getClase {
          return “Clase Hija”;
      }
       public static String getObjeto {
          return “Objeto de clase Hija”;
      }
 }
```

... Ejemplo

```
public class PadreEHija{
    public static void main(String [ ] arg) {
        Padre p=new Padre();
        System.out.println(p.getClase());
        System.out.println(p.getObjeto());

        Hija h=new Hija();
        System.out.println(h.getClase());
        System.out.println(h.getObjeto());

         Padre ph=new Hija();
        System.out.println(ph.getClase());
        System.out.println(ph.getObjeto());

    }
}
```

Sobreescritura (overriding)

- Definición

  Es la diferente implementación que hace un método hijo del método padre.

- Características
  - ✓ Para que existe sobreescritura debe de haber herencia.
  - ✓ Debe respetarse el mismo nombre del método, mismos parámetros (número y tipo de datos) , mismo tipo de retorno (ver. < 1.5) o en >= 1.5 se puede retornar una clase de igual o menor jerarquía.

Sobreescritura de métodos

- . . . Características
  - ✓ El método hijo deberá tener una implementación diferente a la del método padre.
  - ✓ El modificador del método de la subclase debe ser igual o más flexible.
  - ✓ Los constructores no se heredan pero pueden invocarse mediante super().
  - ✓ super.metodo() permite que la subclase invoque un método de la clase padre (se invocan toda la implementación así como del contenido de los atributos privados del padre que se encuentran en ese método.

Ejemplo 1

```
class A {
 String name() { return "A"; }
 }
 class B extends A {
    String name() { return "B"; }
}
class C extends A {
   String name() { return "C"; }
 }
 public class Overriding {
     public static void main(String[] args) {
    A[ ] tests = new A[ ] {  //Colección heterogénea
                          new A(),
                          new B(),
                          new C() };
    for (int i = 0; i < tests.length; i++)
     System.out.print(tests[i].name());
   }
 }
```

Ejemplo 2

```
public class Employee {
  protected String name;
  protected double salary;
  protected Data   birthDate;

  public String getDetails() {
    return "Name: " + name + "\n" +
           "Salary: " + salary;
  }
}

public class Manager extends Employee {
   protected String department;

  public String getDetails() {
    return "Name: " + name + "\n" +
           "Salary: " + salary + "\n" +
           "Manager of: " + department;
  }
}
```

// Otra versión del método getDetails()

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

Ejemplo 3

```
public class Ejemplo {
     public static void main(String args[]) {
          B b= new B();
          b.f(3);
     }
}
public class A {
     public void f (int x) {
          System.out.println("En clase A: " + x);
     }
}
public class B extends A {
     public void f (int x) {
          System.out.println("En clase B: " + x);
          super.f(x);
     }
}
```

Efecto de los modificadores

Modifier	Same Class	Same Package	Subclass	Universe
`private`	Yes			
default	Yes	Yes		
`protected`	Yes	Yes	Yes	
`public`	Yes	Yes	Yes	Yes

Argumentos polimórficos

- Cuando se crea una referencia de una clase padre la invocación de los métodos son del método de la referencia a este método se le llama **virtual method**.

- Ejemplo:

```
Employee e = new Manager();
e.getDetails();
```

- Un método puede recibir parámetros diferentes a los primitivos, es decir aceptan objetos genéricos. Por ejemplo al recibir a un dato de tipo clase padre.

Casting de objetos

- Este tipo de casting generalmente es usado cuando se recibe como parámetro una clase padre y se requiere determinar que tipo de objeto es (una subclase en particular) usando el operador *instanceof* y a partir de conocer el objeto se hace un *cast* para poder usar un método o ejecutar alguna sentencia referente a éste.

- Ejemplo:
- Casting explícito:

```
Padre p =new Hija();
Hija h=(Hija) p;
```

- Casting automático:

```
Hija h =new Hija();
Padre p= h;
```

Ejemplo instanceof

```
class A {    int i, j;  }

class B {   int i, j; }

class C extends A {    int k;  }

class D extends A {    int k;  }

class InstanceOf {
  public static void main(String args[]) {
    A a = new A();
    B b = new B();
    C c = new C();
    D d = new D();
   if  (a instanceof A)
      System.out.println("a is instance of A");
   if  (b instanceof B)
      System.out.println("b is instance of B");
```

Ejemplo instanceof

```
if  (c instanceof C)
     System.out.println("c is instance of C");
if (c instanceof A)
     System.out.println("c can be cast to A");
if (a instanceof C)
     System.out.println("a can be cast to C");
System.out.println();

// compara tipos derivados de los tipos
   A ob;
   ob = d; // A referencia a d
   System.out.println("ob ahora refiere a d");
   if (ob instanceof D)
        System.out.println("ob is instancia de D");
   System.out.println();

   ob = c; // A referencia a c
   System.out.println("ob ahora refiere a  c");
```

Ejemplo instanceof

```
  if (ob instanceof D)
     System.out.println("ob puede hacer cast a D");
  else
     System.out.println("ob NO puede hacer cast a D");
   if (ob instanceof A)
     System.out.println("ob puede hacer cast a A");
   System.out.println();

  if (a instanceof Object)
     System.out.println("a podría hacer cast a Object");
    if (b instanceof Object)
     System.out.println("b podría hacer cast a Object");
    if (c instanceof Object)
     System.out.println("c podría hacer cast a Object");
    if (d instanceof Object)
     System.out.println("d podría hacer cast a Object");
  }
} // Fin del ejemplo
```