PROGRAMACIÓN ORIENTADA A OBJETOS - 22951

Universidad Industrial de Santander

#LaUISqueQueremos

Universidad Industrial de Santander

Escuela de Ingeniería de Sistemas e Informática

Programa de Ingeniería de Sistemas

74 años
1948-2022

Herencia y Polimorfismo

La palabra clave super

- Una subclase **puede** redefinir (sobrescribir) un método de su superclase.
- El método heredado de esta forma será escondido por la redefinición.
- Siempre es posible invocar el método de la superclase a través de la palabra clave **super**

Palabra clave Super - ejemplo

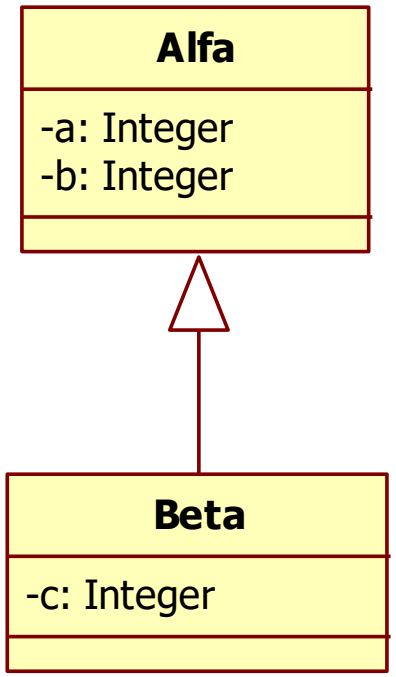
```
public class Alfa {

    public void imprimir() {
        System.out.println("--------------------------");
    }

}
```

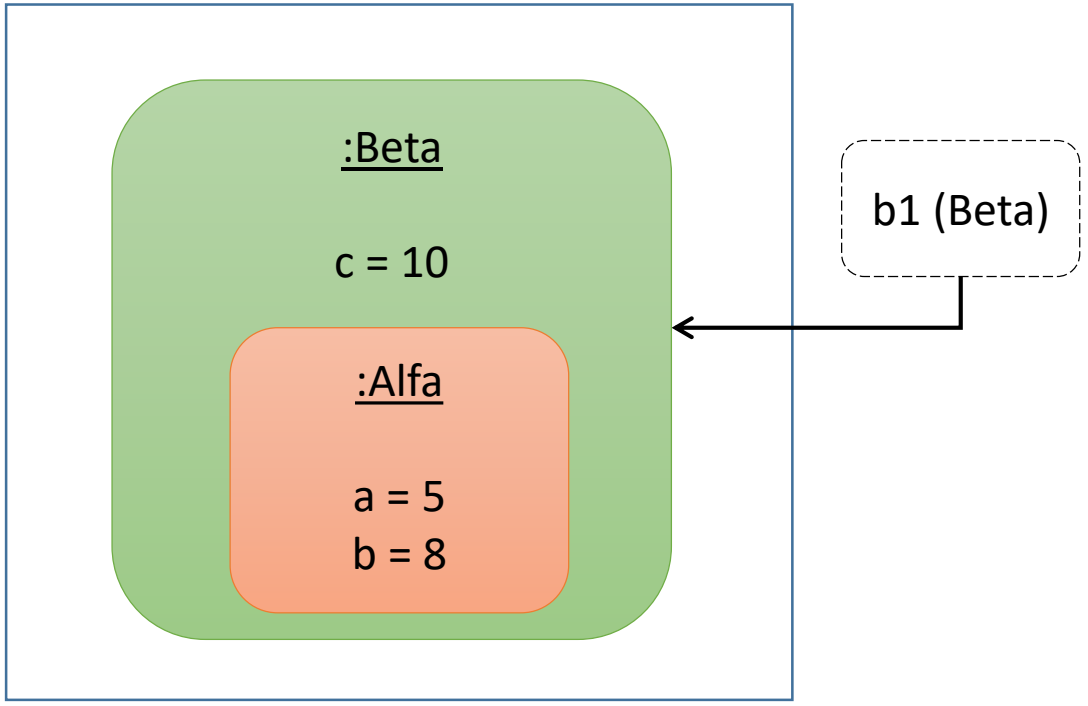
```
public class Beta extends Alfa {

    public void imprimir() {
        super.imprimir();
        System.out.println("Impresion en Beta");
    }

    public void otroMetodo() {
        super.imprimir();
        // Codigo otro metodo
    }

}
```

Constructores y la palabra super

- Toda clase sin constructores definidos explícitamente posee un constructor implícito por defecto sin parámetros.
- Si al menos un constructor es definido explícitamente la clase no contiene el constructor implícito.
- Toda subclase «contiene» una instancia de su superclase.
- Para crear una instancia de una subclase se debe crear primera la instancia de la superclase.
- Para invocar un constructor de una superclase también se usa la palabra clave ***super***

Constructores y la palabra super





Constructores y la palabra super

```
public class Alfa {
    private int a, b;

    public Alfa(int a, int b) {
        this.a = a;
        this.b = b;
    }
}
```

```
public class Beta extends Alfa {

    private int c;

    public Beta(int c) {
        super(5, c);
        this.c = c;
    }

}
```

Constructores y la palabra this

```
public class Alfa {
    private int a;
    private double b;

    public Alfa() {
        a = 0;
        b = 0.0;
    }

    public Alfa (int a) {
        this();
        this.a = a;

    }

    public Alfa(double b) {
        this();
        this.b = b;
    }
}
```

```
Alfa alfa1 = new Alfa();

Alfa alfa2 = new Alfa(5);

Alfa alfa3 = new Alfa(3.0);

alfa1 = new Alfa(6);
alfa2 = new Alfa(6.0);
```

Constructores y la palabra this

```
public class Alfa {
    private int a;
    private double b;

    public Alfa() {
        a = 0;
        b = 0.0;
    }

    public Alfa (int a) {
        this();
        this.a = a;

    }

    public Alfa(double b) {
        this();
        this.b = b;
    }
}
```

```
public class Beta extends Alfa {

    private int c;

    public Beta() {
        super(10);
        this.c = 10;
    }

    public Beta(int c) {
        super(c * 1.0);
        this.c = c;
    }

}
```

```
Beta beta1 = new Beta();
Beta beta2 = new Beta(3);
```

Polimorfismo

El Elefante Celebes - Max Ernst

Polimorfismo

- Característica fundamental de la POO asociada a la herencia.
- Capacidad de un objeto de tomar múltiples (***Poli***) formas (***morfismo***).
- Facultad de un objeto perteneciente a una clase de ser considerado como una ***instancia*** de su ***propia clase*** o una ***instancia*** de su ***superclase***.

Conversión de tipos

Conversión de tipos

- Un entero se puede convertir en un real automáticamente.
- Un real no se puede convertir en un entero sin perder precisión.
- Los números reales ***contienen*** a los enteros.
- La asignación de una variable de un tipo dado a otro tipo se conoce como conversión de tipos (***Cast*** en ingles).
- **Conversión implícita**: no hay que informar al compilador de la conversión (int to float).
- **Conversión explicita**: hay que informar al compilador de la conversión (float to int).

Conversión de tipos - Ejemplo

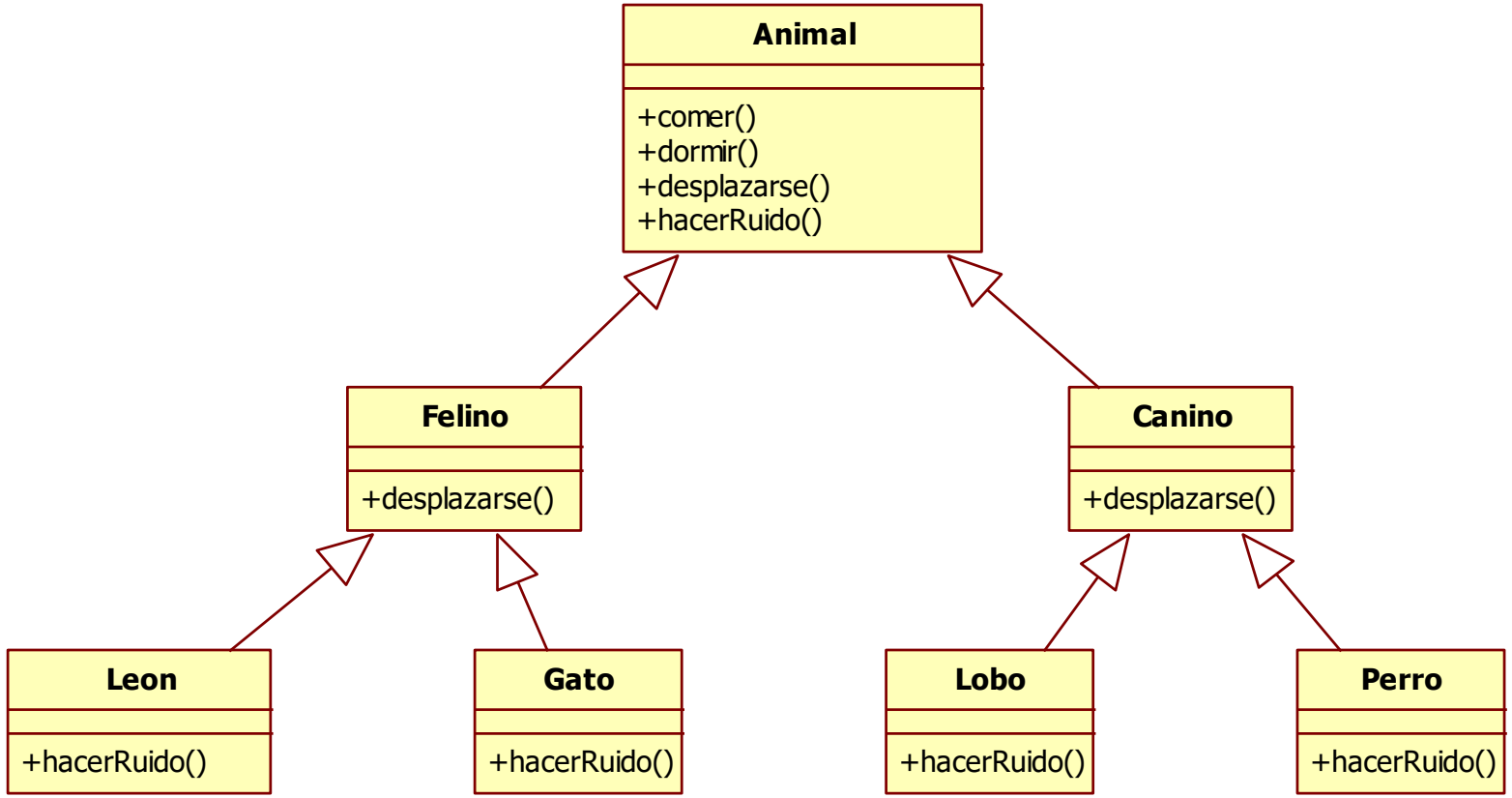
```
private void convertirImplicito() {
    float x = 5.8f;
    int y = 10;

    x = y;
    System.out.println(x);
}
```

```
private void convertirExplicito() {
    float x = 5.8f;
    int y = 10;

    y = (int) x;
    System.out.println(y);
}
```

Conversión de tipos - Objetos

- La noción de **inclusión** de tipos aplica también a las clases definidas por herencia.
- Conversión de tipo **ascendente:**
  - Ver un objeto de una subclase como un objeto de una superclase.
  - La conversión se define de forma **implícita**.
  - La conversión es autorizada por el compilador.
- Conversión de tipo **descendente:**
  - Ver un objeto de una superclase como un objeto de una subclase.
  - Debe ser definida de forma **explicita.**
  - Controlada a la ejecución. El *runtime* verifica si es posible realizar la conversión.

Recordar ejemplo jerarquía animal



Conversión Ascendente – referencias e instancias

Enlace del objeto y la variable

```
Lobo lobito = new Lobo( );
```

Declaración de una variable de referencia

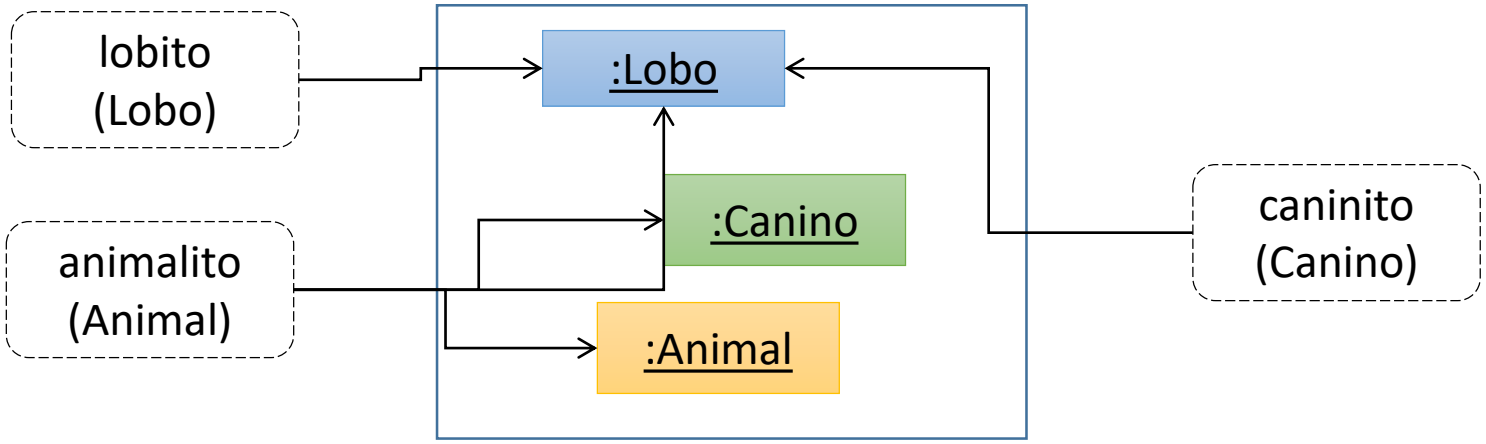
Creación de una instancia de tipo Lobo

Conversión Ascendente – referencias e instancias

```
Animal animalito = new Lobo( );
```

El tipo de la variable de referencia puede ser de una *superclase* de la instancia asignada

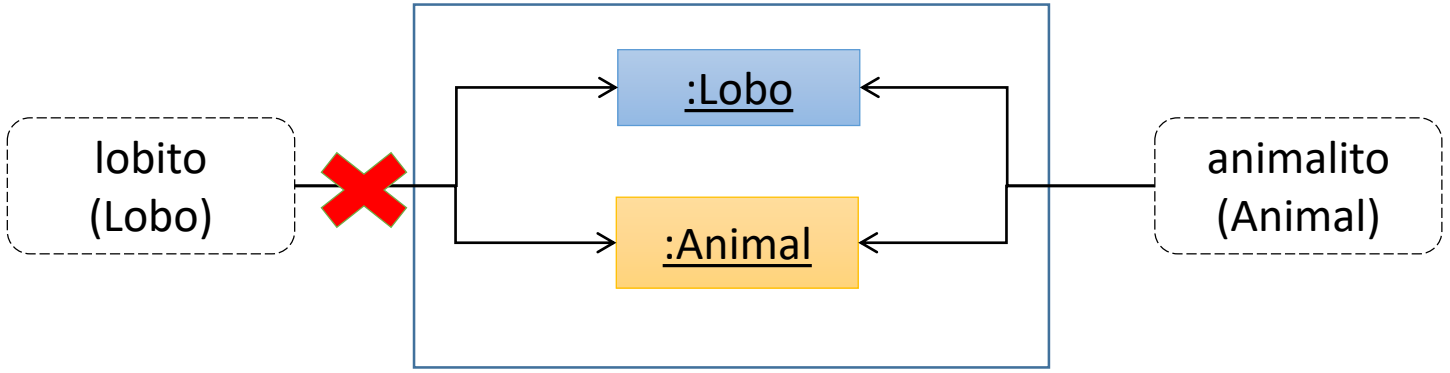
Conversión Ascendente




```
public void test() {
    Lobo lobito = new Lobo();
    Animal animalito = null;
    Canino caninito = null;
    animalito = lobito;
    animalito = new Canino();
    caninito = lobito;
    animalito = new Animal();
}
```

Permite ver un objeto de una subclase como un objeto de una superclase

Conversión descendente




```
public void testDes() {
Lobo lobito = null;
Animal animalito = new Lobo();
lobito = (Lobo) animalito;
animalito = new Animal();
lobito = (Lobo) animalito;
}
```


Permite ver un objeto de una superclase como un objeto de una subclase

Enlace dinámico

Enlace dinámico

- Sea una clase Beta que hereda de una clase Alfa.
- Sea un método m() definido en Alfa y redefinido en Beta
- Sea a1 una instancia de Alfa y b1 una instancia de Beta
- Si se realiza la conversión ascendente a1 = b1;
- La invocación del método m sobre a1 a través de a1.m() ejecuta el código definido en la clase B
- Permite guardar en una estructura de datos toda una jerarquía y que los objetos se comporten según la su clase real

Ejemplo Enlace Dinamico

```
public void testDinamico() {
  Animal[] animales = new Animal[4];

  animales[0] = new Perro();
  animales[1] = new Lobo();
  animales[2] = new Gato();
  animales[3] = new Leon();

  for (int i = 0; i < animales.length; i++) {
     animales[i].hacerRuido();
  }

  for (Animal animal : animales) {
     animal.hacerRuido();
  }
}
```

Jerarquía a varios niveles

```
public class Animal {

    public void comer() {
        System.out.println("Yumi Yumi");
    }

    public void dormir() {
        System.out.println("Zzzzzzzzzz");
    }

    public void desplazarse() {
        System.out.println("Caminando por las calles ....");
    }

    public void hacerRuido() {
        System.out.println("Haciendo ruido ...");
    }

}
```

Clases Felino y canino

```
public class Felino extends Animal {

   public void desplazarse() {
      System.out.println("Desplazamiento solitario");
   }

}

 public class Canino extends Animal {

    public void desplazarse() {
       System.out.println("Desplazamiento en manada");
    }
 }
```

Clases Leon y Gato

```
public class Leon extends Felino {

   public void hacerRuido() {
      System.out.println("Grrrrrrr");
   }

}
```

```
public class Gato extends Felino {

   public void hacerRuido() {
      System.out.println("Miau Miau");
   }

}
```

Clases Perro y Lobo

```
public class Perro extends Canino {

    public void hacerRuido() {
        System.out.println("Guau Guau");
    }

}

public class Lobo extends Canino {

    public void hacerRuido() {
        System.out.println("Aauuuuuuu");
    }

}
```

Ejemplo Polimorfismo - Conversión

```
private void test1() {
    Canino can1 = new Canino();
    Lobo lobo1 = new Lobo();

    can1.hacerRuido();
    can1 = lobo1;
    can1.hacerRuido();
}
```

```
private void test2() {
    Canino can1 = new Canino();
    Lobo lobo1 = new Lobo();

    lobo1.hacerRuido();
    lobo1 = can1;
    lobo1.hacerRuido();
}
```

```
private void test3() {
    Canino can1 = new Canino();
    Lobo lobo1 = new Lobo();

    lobo1.hacerRuido();
    lobo1 = (Lobo) can1;
    lobo1.hacerRuido();
}
```

```
private void test4() {
    Canino can1 = new Lobo();
    Lobo lobo1 = new Lobo();

    lobo1.hacerRuido();
    lobo1 = (Lobo) can1;
    lobo1.hacerRuido();
}
```

Universidad Industrial de Santander

#LaUISqueQueremos

¡Gracias!

#OrgulloEISI