



PROGRAMACIÓN ORIENTADA A OBJETOS - 22951



Universidad Industrial de Santander
Escuela de Ingeniería de Sistemas e Informática
Programa de Ingeniería de Sistemas

#LaUISqueQueremos





Universidad
Industrial de
Santander

Herencia





Universidad
Industrial de
Santander

Encapsulación



Encapsulación – Getters y Setters

Los atributos de una clase generalmente deben ser **privados**

```
public class Estudiante {  
  
    private int codigo;  
    private String nombre;  
    private String apellido;  
  
    public Estudiante(int codigo, String nombre, String apellido) {  
        this.codigo = codigo;  
        this.nombre = nombre;  
        this.apellido = apellido;  
    }  
}
```



Encapsulación – Getters y Setters

Los atributos de una clase generalmente deben ser **privados**

```
public class EstudianteTester {  
  
    public static void main(String[] args) {  
        Estudiante estu = new Estudiante(2020345, "Laura", "Lopez");  
        System.out.println(estu.nombre);  
        estu.codigo = 2021567;  
    }  
}
```

No se puede
modificar el
atributo

No se puede leer
el atributo



Encapsulación – Getters y Setters

- Utilizamos métodos para acceder a los atributos de una instancia.
- Métodos de **acceso**: permiten leer el valor.
- Métodos de **modificación**: permiten modificar el valor.
- Convención en Java: atributo llamado *codigo*
 - `getCodigo()` permite leer el valor.
 - `setCodigo()` permite modificar el valor.
- Estos métodos son conocidos como getters y setters.





Universidad
Industrial de
Santander



```
public class Estudiante {  
  
    private int codigo;  
    private String nombre;  
    private String apellido;  
  
    public int getCodigo() {  
        return codigo;  
    }  
  
    public void setCodigo(int cod) {  
        codigo = cod;  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nom) {  
        nombre = nom;  
    }  
  
    public String getApellido() {  
        return apellido;  
    }  
  
    public void setApellido(String apellido) {  
        this.apellido = apellido;  
    }  
}
```



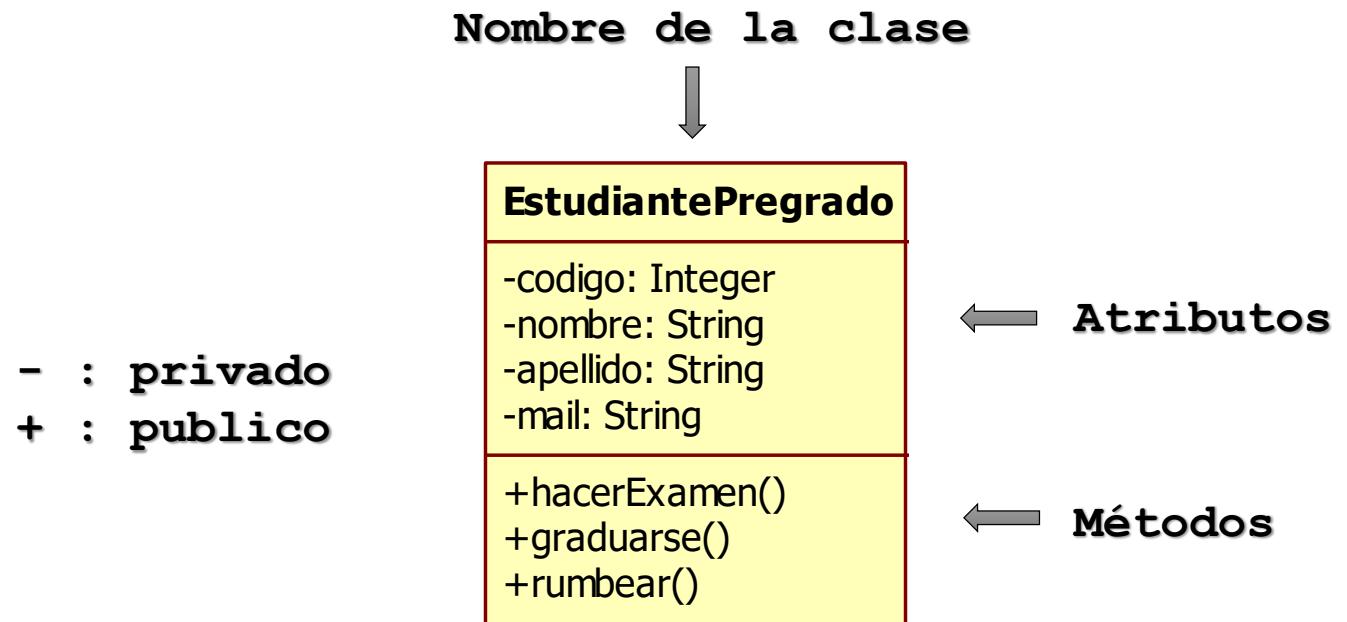
Universidad
Industrial de
Santander

Herencia



Diagrama de clases UML

El lenguaje UML permite especificar y visualizar sistemas orientadas a objetos



Clases con cosas en común

EstudiantePregrado
-codigo: Integer
-nombre: String
-apellido: String
-mail: String
+hacerExamen()
+graduarse()
+rumbear()

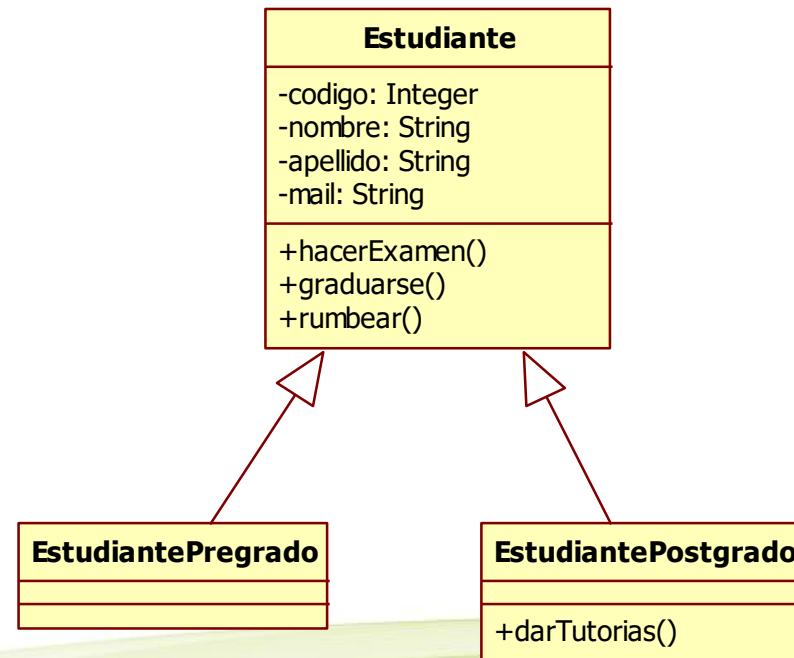
EstudiantePostgrado
-codigo: Integer
-nombre: String
-apellido: String
-mail: String
+hacerExamen()
+graduarse()
+rumbear()
+darTutorias()



- Duplicación de atributos y métodos en las clases
- Como eliminar la redundancia?

Abstrayendo cosas comunes

- Jerarquía de herencia:
 - Subclases (*EstudiantePregrado*, *EstudiantePostgrado*) **heredan** de la clase *Estudiante*.
 - Una flecha con la punta vacía indica **generalización** en UML.



Subclases y superclases

- Una subclase (e.g. *EstudiantePregrado*) hereda de una superclase (e.g. *Estudiante*).
- Una subclase es una **especialización** de una superclase.
- Una superclase es una **generalización** de una subclase.



Herencia

- Las subclases heredan todos los miembros (atributos y métodos) de la superclase.
- La subclase solo puede acceder a los miembros **públicos** de la superclase.
- En Java se usa la palabra clave **extends**
- Una subclase puede agregar nuevos miembros.
- Por defecto, los métodos heredados de una superclase tienen la misma implementación en la subclase:
 - Excepto si la clase **sobrescribe** los métodos heredados.



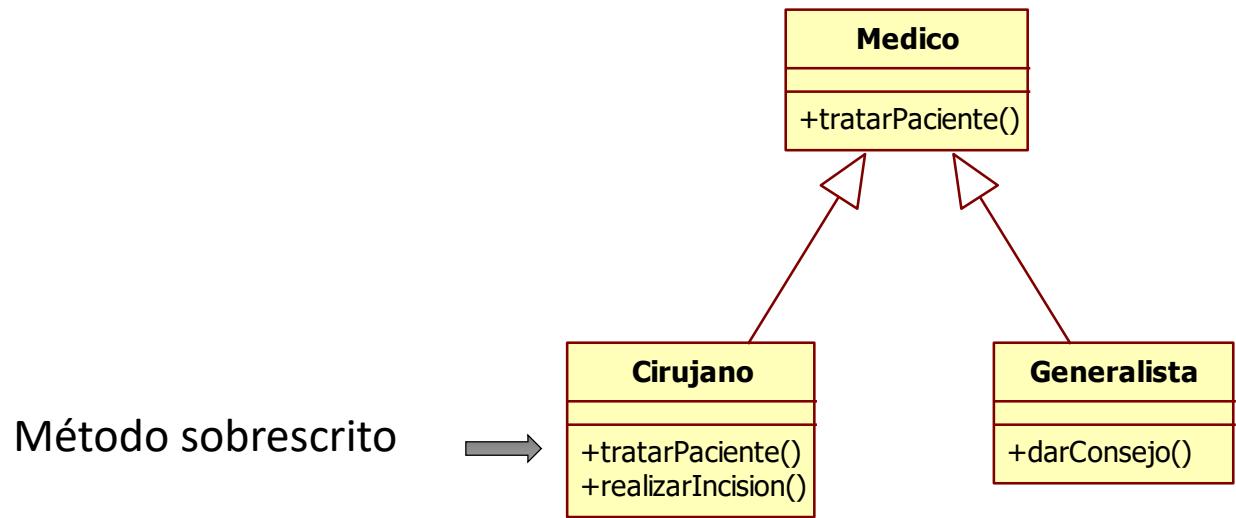
X is-a Y?

- El test is-a (es-una):
 - La claseX es una subclase de la claseY?
 - Test: podemos decir que la claseX es-una (un tipo de) la claseY?
 - Debe: la claseX debe tener todos los miembros (o mas) de la claseY?

- Cocina es una subclase de cuarto?
- Cuarto es una subclase de casa?
- Violinista es una subclase de Músico?
- Lavaplatos es una subclase de Cocina?
- Músico es una subclase de Persona?
- Lady Gaga es una subclase de Músico?
- Estudiante es una subclase de Músico?
- Delantero es una subclase de Futbolista?



Ejemplo Medico



Ejemplo Medico

```
public class Medico {  
  
    private String nombre;  
  
    public void tratarPaciente() {  
        System.out.println("Realiza una revisión");  
    }  
}
```



Ejemplo Medico

```
public class Generalista extends Medico {  
  
    public void darConsejo() {  
        System.out.println("Aconseja como tratar La enfermedad");  
    }  
}
```

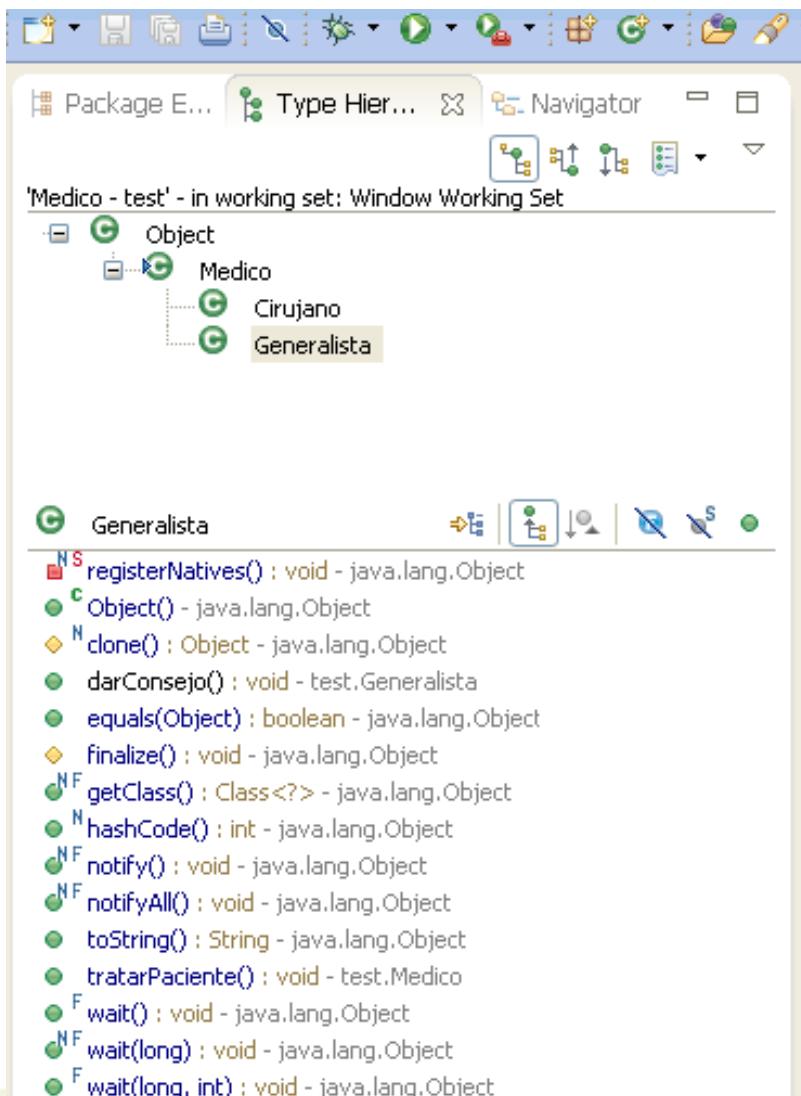


Ejemplo Medico

```
public class Cirujano extends Medico {  
  
    // Método sobrescrito  
    public void tratarPaciente() {  
        System.out.println("Realizar una cirugía");  
    }  
  
    // Método adicional  
    public void realizarIncision() {  
        System.out.println("usando el escalpelo");  
    }  
}
```



Vista de jerarquía de tipos en Eclipse

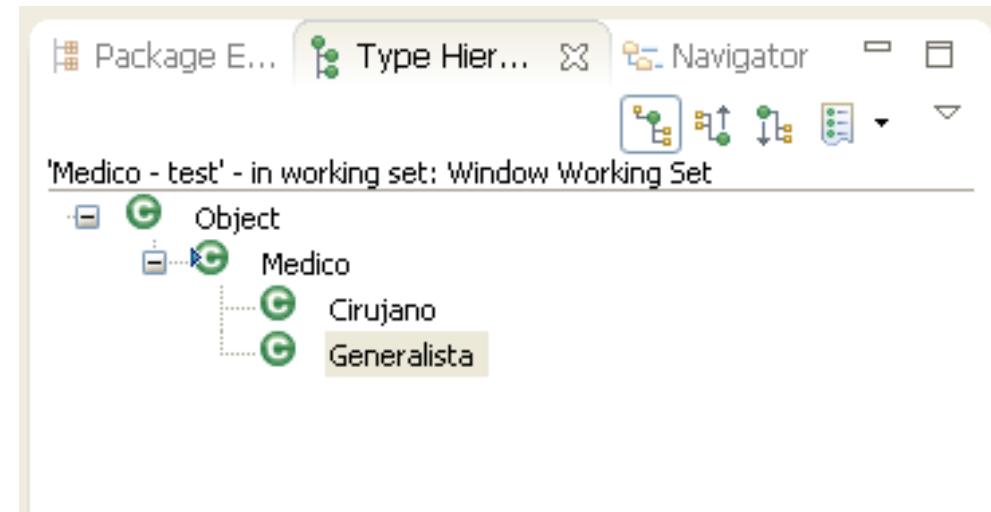


← Panel de jerarquía

← Panel de miembros



Panel de jerarquía - Eclipse



Muestra la jerarquía de la clase seleccionada.

Panel de miembros - Eclipse

The screenshot shows the Eclipse IDE's Member Hierarchy view for the class 'Generalista'. The tree view displays the following members:

- registerNatives() : void - java.lang.Object** (NS)
- Object() - java.lang.Object** (C)
- clone() : Object - java.lang.Object** (N)
- darConsejo() : void - test.Generalista**
- equals(Object) : boolean - java.lang.Object**
- finalize() : void - java.lang.Object** (NF)
- getClass() : Class<?> - java.lang.Object** (NF)
- hashCode() : int - java.lang.Object** (NF)
- notify() : void - java.lang.Object** (NF)
- notifyAll() : void - java.lang.Object** (NF)
- toString() : String - java.lang.Object**
- tratarPaciente() : void - test.Medico**
- wait() : void - java.lang.Object** (F)
- wait(long) : void - java.lang.Object** (NF)
- wait(long, int) : void - java.lang.Object** (F)



Muestra los miembros heredados y no heredados de la clase seleccionada

Proceso de diseño



Universidad
Industrial de
Santander

- Identificar los objetos que tienen **atributos y comportamiento comunes.**
- Diseñar una clase que represente el estado y comportamiento común.
- Decidir si una subclase necesita implementaciones de métodos que son específicas.
- Llevar a cabo otra abstracción para identificar grupos de subclases que tienen comportamiento común.





Universidad
Industrial de
Santander

Encapsulación y Herencia



Encapsulación y herencia

```
public class Estudiante {  
  
    private int codigo;  
    private String nombre;  
    private String apellido;  
  
    public Estudiante(int codigo, String nombre, String apellido)  
    {...}  
  
    public int getCodigo() {...}  
  
    public void setCodigo(int cod) {...}  
  
    public String getNombre() {...}  
  
    public void setNombre(String nom) {...}  
  
    public String getApellido() {...}  
  
    public void setApellido(String apellido) {...}
```



Encapsulación y herencia

```
public class EstudiantePostgrado extends Estudiante {  
  
    private int codAux;  
  
    ...  
  
    public String toString() {  
        return "EP - Nombre: " + nombre + " Apellido " + apellido +  
               " Código " + codigo + " Cód auxiliatura" + codAux;  
    }  
}
```

Wrong



nombre, apellido y código son atributos privados en la superclase y no pueden ser accedidos

Encapsulación y herencia

```
public class EstudiantePostgrado extends Estudiante {  
  
    private int codAux;  
  
    public EstudiantePostgrado(int codigo, String nombre, String apellido) {  
        super(codigo, nombre, apellido);  
    }  
  
    public String toString() {  
        return "EP - Nombre: " + getNombre() + " Apellido " +  
getApellido() +  
            " Código " + getCodigo() + " cod auxliatura" + codAux;  
    }  
}
```



Encapsulación y herencia

- Los atributos privados no pueden ser accedidos directamente por las subclases.
- Se deben acceder a través de los métodos getters y setters definidos en la superclase.
- Los métodos getters y setters son públicos, por esta razón se pueden acceder en la subclase.



Palabra Clave Protected

Permite definir miembros que solamente las subclases (y la propia clase) pueden acceder:

```
public class Alfa {  
  
    private int i;  
    protected int j;  
    public int k;  
  
}  
  
public class Beta extends Alfa {  
  
    private void imprimir() {  
        System.out.println(j+k);  
    }  
  
}
```



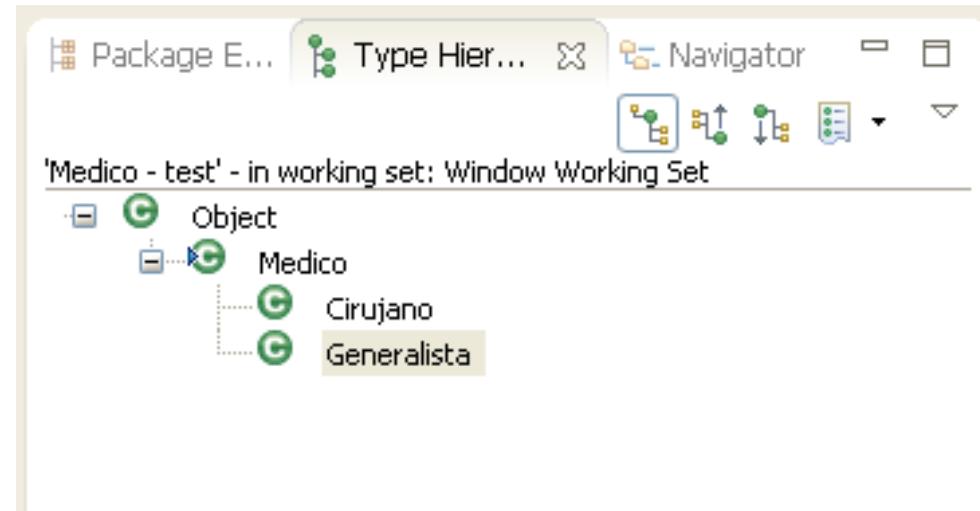


Universidad
Industrial de
Santander

LA SUPERCLASE OBJECT



Ejemplo de Medico



En donde encaja la clase Medico en la jerarquía?

La clase Object

La clase **Object** es la superclase de la clase Medico

```
public class Medico extends Object {  
  
    public void tratarPaciente() {  
        // Realizar una revision  
    }  
  
}
```



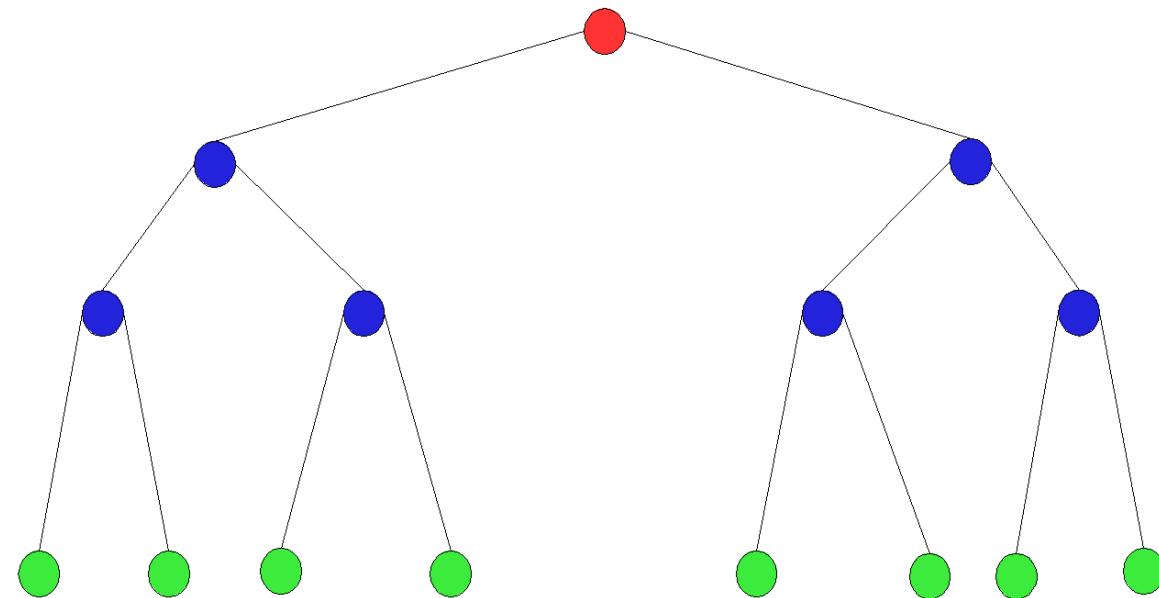
- La clase **Object** es la superclase de todas las clases en Java.
- Si una clase no extiende explícitamente otra clase, entonces implícitamente extiende a **Object**.
- No necesitamos entonces añadir **extends Object**.

Algunos métodos de la clase Object

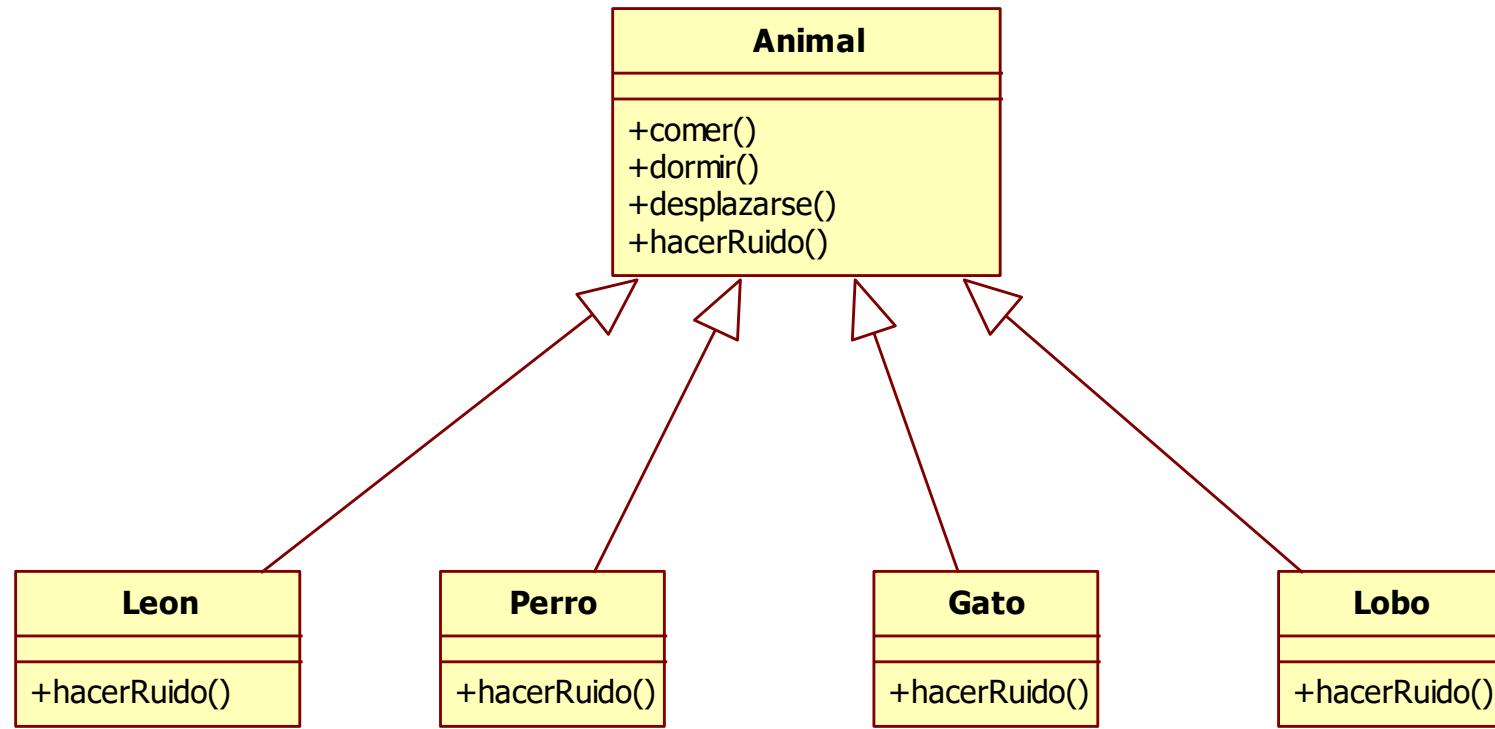
- equals(Object o) – determina si dos instancias son iguales.
- hashCode() – un ID numérico, instancias iguales deben tener el mismo código hash.
- toString() – regresa una representación textual de una instancia. Este método es invocado automáticamente por System.out.println()
- Dado que cada clase hereda el método toString() ya hemos sobrescrito este método en el ejemplo de la jerarquía de Estudiante.



JERARQUÍAS PLANAS Y A VARIOS NIVELES



Jerarquía plana de la clase Animal



Jerarquía plana de la clase Animal

```
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 ...");  
    }  
}
```



Jerarquía plana de la clase Animal

- Leon es una subclase de Animal.
- Leon sobrescribe el método hacerRuido porque necesita una implementación particular.

```
public class Leon extends Animal {  
  
    public void hacerRuido() {  
        System.out.println("Grrrrrrr");  
    }  
  
}
```



Jerarquía plana de la clase Animal

```
public class Gato extends Animal {  
    public void hacerRuido() {  
        System.out.println("Miau Miau");  
    }  
}
```

```
public class Lobo extends Animal {  
    public void hacerRuido() {  
        System.out.println("Aauuuuuuu");  
    }  
}
```

```
public class Perro extends Animal {  
    public void hacerRuido() {  
        System.out.println("Guau Guau");  
    }  
}
```

- Todas son subclases de Animal
- Cada clase sobrescribe el método hacerRuido



Usando las clases ...



Universidad
Industrial de
Santander

```
public class TestAnimal {  
  
    public static void main(String[] args) {  
        System.out.println("Lobo");  
        Lobo elLobo = new Lobo();  
        elLobo.hacerRuido();  
        elLobo.comer();  
        elLobo.desplazarse();  
  
        System.out.println();  
  
        System.out.println("Leon");  
        Leon elLeon = new Leon();  
        elLeon.hacerRuido();  
        elLeon.comer();  
        elLeon.desplazarse();  
    }  
}
```



Usando las clases - Ejecución

Lobo

Aauuuuuuu

Yumi Yumi

Caminando por las calles . . .



Leon

Grrrrrrrr

Yumi Yumi

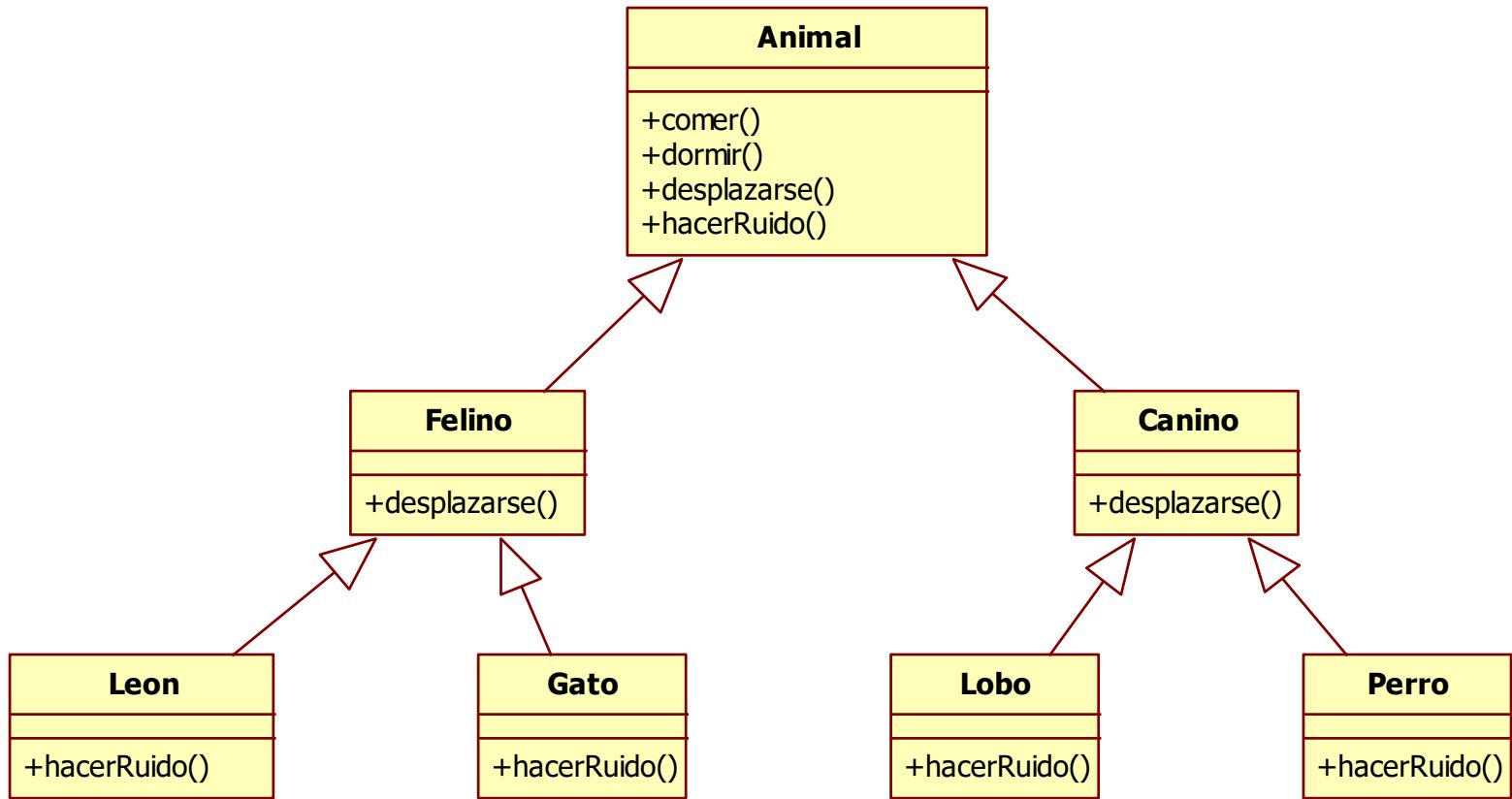
Caminando por las calles . . .

Jerarquía a varios niveles

- Leones y gatos pueden agruparse en la categoría de Felinos, los felinos poseen comportamientos comunes (desplazamiento).
- Perros y lobos pueden agruparse en la categoría de Caninos, ellos también poseen comportamiento común en el desplazamiento.



Jerarquía a varios niveles



Jerarquía a varios niveles

```
public class Animal {  
  
    public void comer() {  
        System.out.println("Yumi Yumi");  
    }  
  
    public void dormir() {  
        System.out.println("Zzzzzzzzz");  
    }  
  
    public void desplazarse() {  
        System.out.println("Caminando por las calles ....");  
    }  
  
    public void hacerRuido() {  
        System.out.println("Haciendo ruido ...");  
    }  
}
```



Nuevas 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");  
    }  
}
```



Las clases Leon y Gato modificadas

```
public class Leon extends Felino {  
  
    public void hacerRuido() {  
        System.out.println("Grrrrrrrr");  
    }  
  
}  
  
public class Gato extends Felino {  
  
    public void hacerRuido() {  
        System.out.println("Miau Miau");  
    }  
  
}
```

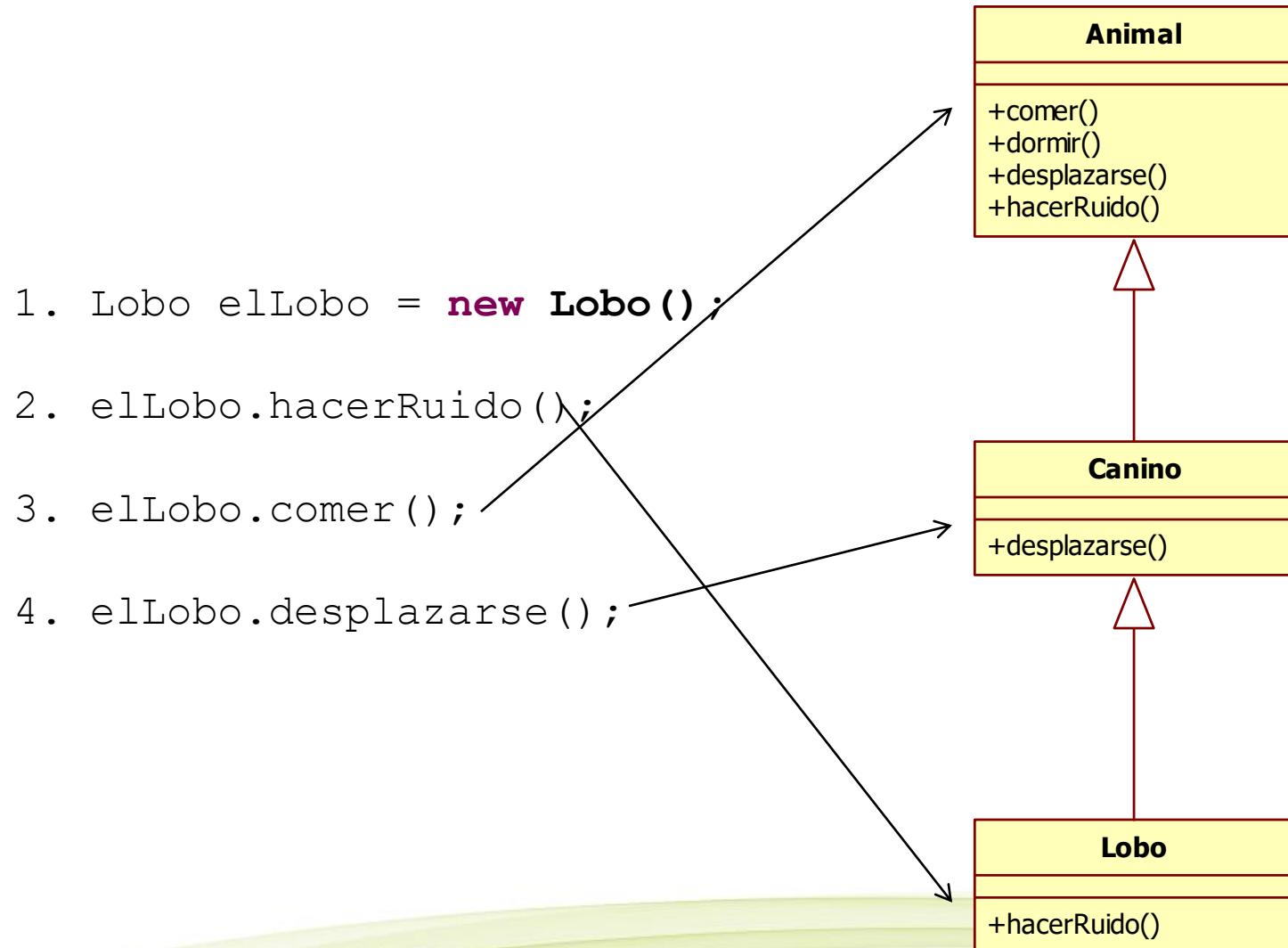


Las nuevas clases Perro y Lobo

```
public class Perro extends Canino {  
  
    public void hacerRuido() {  
        System.out.println("Guau Guau");  
    }  
  
}  
  
public class Lobo extends Canino {  
  
    public void hacerRuido() {  
        System.out.println("Auuuuuuuu");  
    }  
  
}
```



Como se invocan los métodos



Usando las clases ...



Universidad
Industrial de
Santander

```
public class TestAnimal {  
  
    public static void main(String[] args) {  
        System.out.println("Lobo");  
        Lobo elLobo = new Lobo();  
        elLobo.hacerRuido();  
        elLobo.comer();  
        elLobo.desplazarse();  
  
        System.out.println();  
  
        System.out.println("Leon");  
        Leon elLeon = new Leon();  
        elLeon.hacerRuido();  
        elLeon.comer();  
        elLeon.desplazarse();  
    }  
}
```



Usando las clases - Ejecución

Lobo

Aauuuuuuu

Yumi Yumi

Desplazamiento en manada



Leon

Grrrrrrrr

Yumi Yumi

Desplazamiento solitario

Para Profundizar

- Leer Capítulo 5, Sección 5 de *Introduction to Programming Using Java*
- Leer Capítulo 6 de *Thinking in Java*
- Ojear http://en.wikipedia.org/wiki/Class_diagram





Universidad
Industrial de
Santander



#OrgulloEISI

¡Gracias!

#LaUISqueQueremos