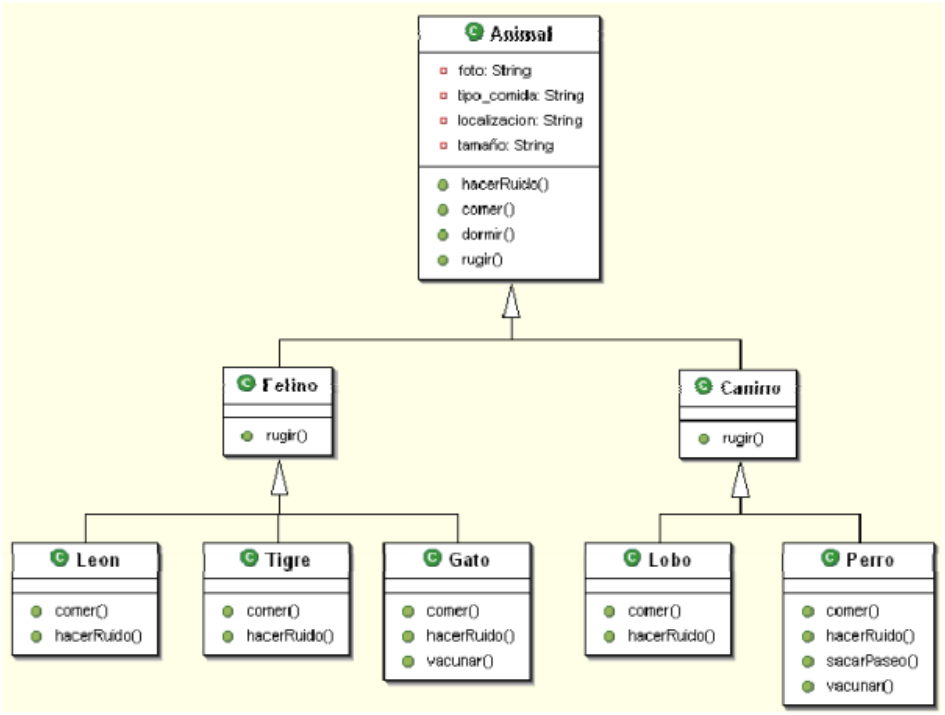
Relación de herencia

- Se basa en la existencia de relaciones de generalización/especialización entre clases.
- Las clases se disponen en una jerarquía, donde una clase hereda los atributos y métodos de las clases superiores en la jerarquía.
- Una clase puede tener sus propios atributos y métodos adicionales a lo heredado.
- Una clase puede modificar los atributos y métodos heredados.

Relación de herencia

- Las clases por encima en la jerarquía a una clase dada, se denominan superclases.
- Las clases por debajo en la jerarquía a una clase dada, se denominan subclases.
- Una clase puede ser superclase y subclase al mismo tiempo.
- Tipos de herencia:
  - Simple.
  - Múltiple (no soportada en Java)

Ejemplo



Herencia

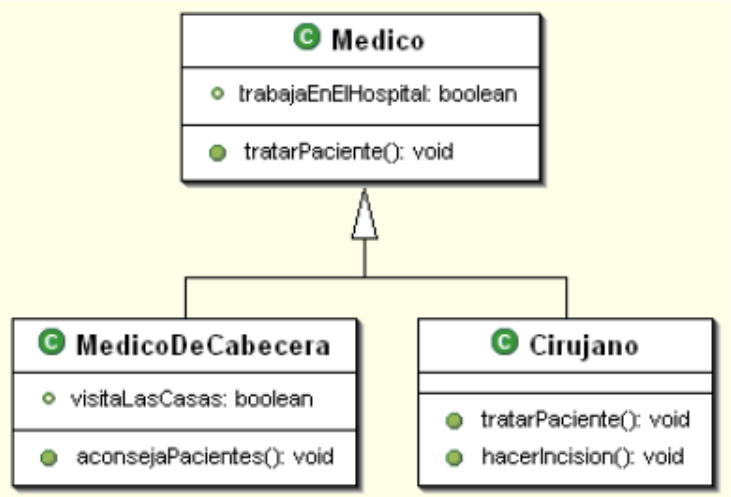
- La implementación de la herencia se realiza mediante la *keyword*: extends.
- Declaración de la herencia:

```
modificador_acceso class nom_clase extends nom_clase
{
}
```

- Ejemplo:

```
public class MiClase extends OtraClase
{
}
```

Ejemplo



Ejemplo

```
public class Medico
{
  public boolean trabajaEnHospital;

  public void tratarPaciente()
  {
    //Realizar un chequeo.
  }
}
```

```
public class MedicoDeCabecera extends Medico
{
  public boolean visitaLasCasas;

  public void aconsejaPacientes()
  {
    //Ofrecer remedios caseros.
  }
}

public class Cirujano extends Medico
{
  public void tratarPaciente()
  {
    //Realizar una operación.
  }

  public void hacerIncision()
  {
    //Realizar la incisión (¡ouch!).
  }
}
```

La clase Object

- En Java todas las clases heredan de otra clase:
  - Si lo especificamos en el código con la *keyword* extends, nuestra clase heredará de la clase especificada.
  - Si no lo especificamos en el código, el compilador hace que nuestra clase herede de la clase Object (raíz de la jerarquía de clases en Java).
- Ejemplo:

```
public class MiClase extends Object
{
   // Es redundante escribirlo puesto que el
   // compilador lo hará por nosotros.
}
```

La clase Object

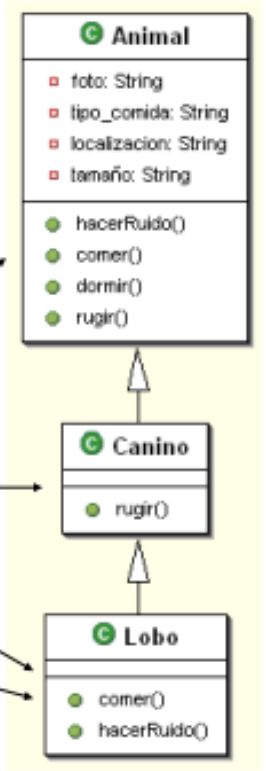
- Esto significa que nuestras clases siempre van a contar con los atributos y métodos de la clase Object.
- Algunos de sus métodos mas importantes son:
  - **public boolean** equals(Object o);
    Compara dos objetos y dice si son iguales.
  - **public** String toString();
    Devuelve la representación visual de un objeto.
  - **public** Class getClass();
    Devuelve la clase de la cual es instancia el objeto.

Sobrescribir un método

- Sobrescribir un método significa que una subclase reimplementa un método heredado.
- Para sobrescribir un método hay que respetar totalmente la declaración del método:
  - El nombre ha de ser el mismo.
  - Los parámetros y tipo de retorno han de ser los mismos.
  - El modificador de acceso no puede ser mas restrictivo.
- Al ejecutar un método, se busca su implementación de abajo hacia arriba en la jerarquía de clases.

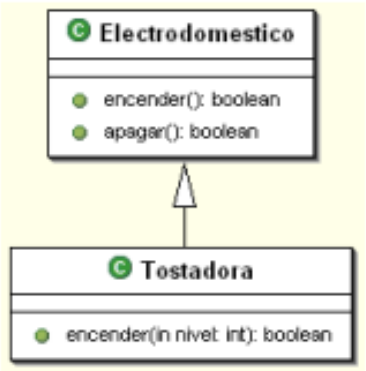
Ejemplo

```
public class Test
{
  public static void main (String[] args)
  {
    Lobo lobo = new Lobo();

    lobo.hacerRuido();

    lobo.rugir();

    lobo.comer();

    lobo.dormir();
}
```



Ejemplo

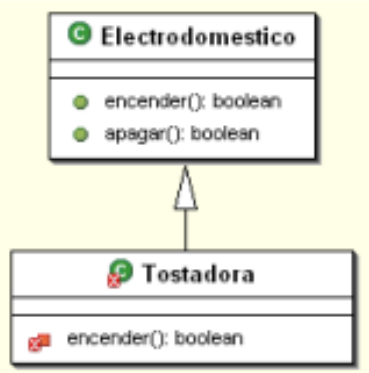
```
public class Electrodomestico
{
  public boolean encender()
  {
    //Hacer algo.
  }
  public boolean apagar()
  {
    //Hacer algo.
  }
}

public class Tostadora extends Electrodomestico
{
  public boolean encender(int nivel)
  {
    //Hacer algo.
  }
}
```



No es sobrescritura. Los parámetros son distintos. Es sobrecarga.

Ejemplo

```
public class Electrodomestico
{
  public boolean encender()
  {
    //Hacer algo.
  }
  public boolean apagar()
  {
    //Hacer algo.
  }
}

public class Tostadora extends Electrodomestico
{
  private boolean encender()
  {
    //Hacer algo.
  }
}
```



No compila. Es sobrescritura restringiendo el acceso.

El uso de la Herencia

- Debemos usar herencia cuando hay una clase de un tipo mas específico que una superclase.
  Es decir, se trata de una especialización.
- Debemos usar herencia cuando tengamos un comportamiento que se puede reutilizar entre varias otras clases del mismo tipo genérico.

El uso de la Herencia

- No debemos usar herencia solo por el hecho de reutilizar código. Nunca debemos romper las dos primeras reglas.
  - Podemos tener el comportamiento cerrar en Puerta. Pero aunque necesitemos ese mismo comportamiento en Coche no vamos a hacer que Coche herede de Puerta. En todo caso, coche tendrá un atributo del tipo Puerta.
- No debemos usar herencia cuando no se cumpla la regla: Es-un/una

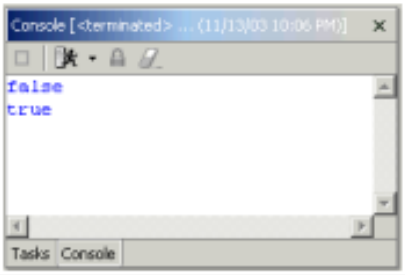
super y this

- super y this son dos *keywords* de Java.
- super es una referencia al objeto actual pero apuntando al padre.
- super se utiliza para acceder desde un objeto a atributos y métodos (incluyendo constructores) del padre.
- Cuando el atributo o método al que accedemos no ha sido sobrescrito en la subclase, el uso de super es redundante.
- Los constructores de las subclases incluyen una llamada a super() si no existe un super o un this.

super y this

- Ejemplo de acceso a un atributo:

```
public class ClasePadre
{
  public boolean atributo = true;
}


public class ClaseHija extends ClasePadre
{
  public boolean atributo = false;
  public void imprimir()
  {
    System.out.println(atributo);
    System.out.println(super.atributo);
  }
}
```

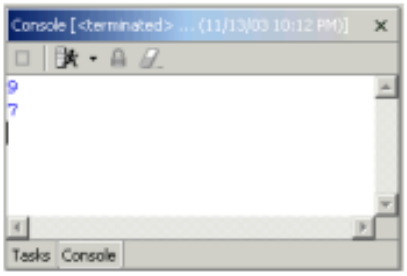


super y this

- Ejemplo de acceso a un constructor:

```
public class ClasePadre
{
  public ClasePadre(int param)
  {
    System.out.println(param);
  }
}
public class ClaseHija extends ClasePadre
{
  public ClaseHija(int param)
  {
    super(param + 2);
    System.out.println(param);
  }
}
```

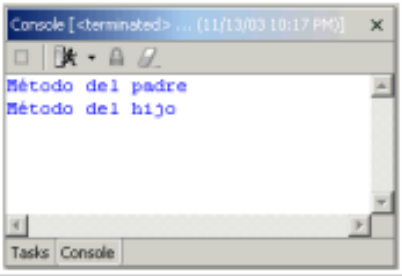
Nota: tiene que ser la primera línea del constructor y solo puede usarse una vez por constructor.



super y this

- Ejemplo de acceso a un método:

```
public class ClasePadre
{
  public void imprimir()
  {
    System.out.println("Método del padre");
  }
}
public class ClaseHija extends ClasePadre
{
  public void imprimir()
  {
    super.imprimir();
    System.out.println("Método del hijo");
  }
}
```



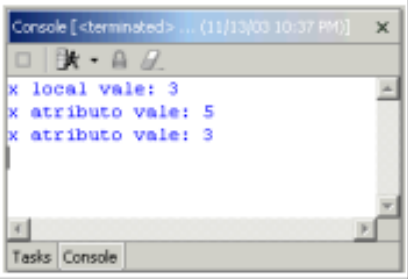
super y this

- this es una referencia al objeto actual.
- this se utiliza para acceder desde un objeto a atributos y métodos (incluyendo constructores) del propio objeto.
- Existen dos ocasiones en las que su uso no es redundante:
  - Acceso a un constructor desde otro constructor.
  - Acceso a un atributo desde un método donde hay definida una variable local con el mismo nombre que el atributo.

super y this

- Ejemplo de acceso a un atributo:

```
public class MiClase
{
  private int x = 5;
  public void setX(int x)
  {
    System.out.println("x local vale: " + x);
    System.out.println("x atributo vale: " + this.x);
    this.x = x;
    System.out.println("x atributo vale: "
                                        + this.x);
  }
}
```



super y this

- Ejemplo de acceso a un constructor:

```
public class MiClase
{
  public MiClase()
  {
    this(2);
    System.out.println("Constructor sin");
  }
  public MiClase(int param)
  {
    System.out.println("Contructor con");
  }
}
```

Nota: tiene que ser la primera línea del constructor y solo puede usarse una vez por constructor.