Programación

Unidad 4: Herencia

Unidad 4

- Herencia
- Sobrecarga de métodos
- Sobreescritura de métodos
- super y this
- Modificadores de acceso

Herencia

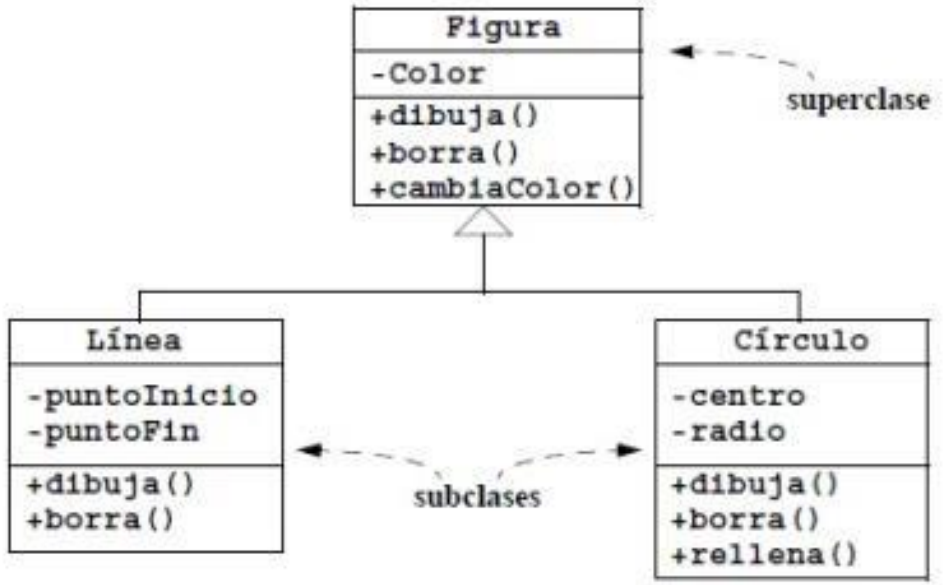
Unidad 4: Herencia

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.



Unidad 4: Herencia

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.

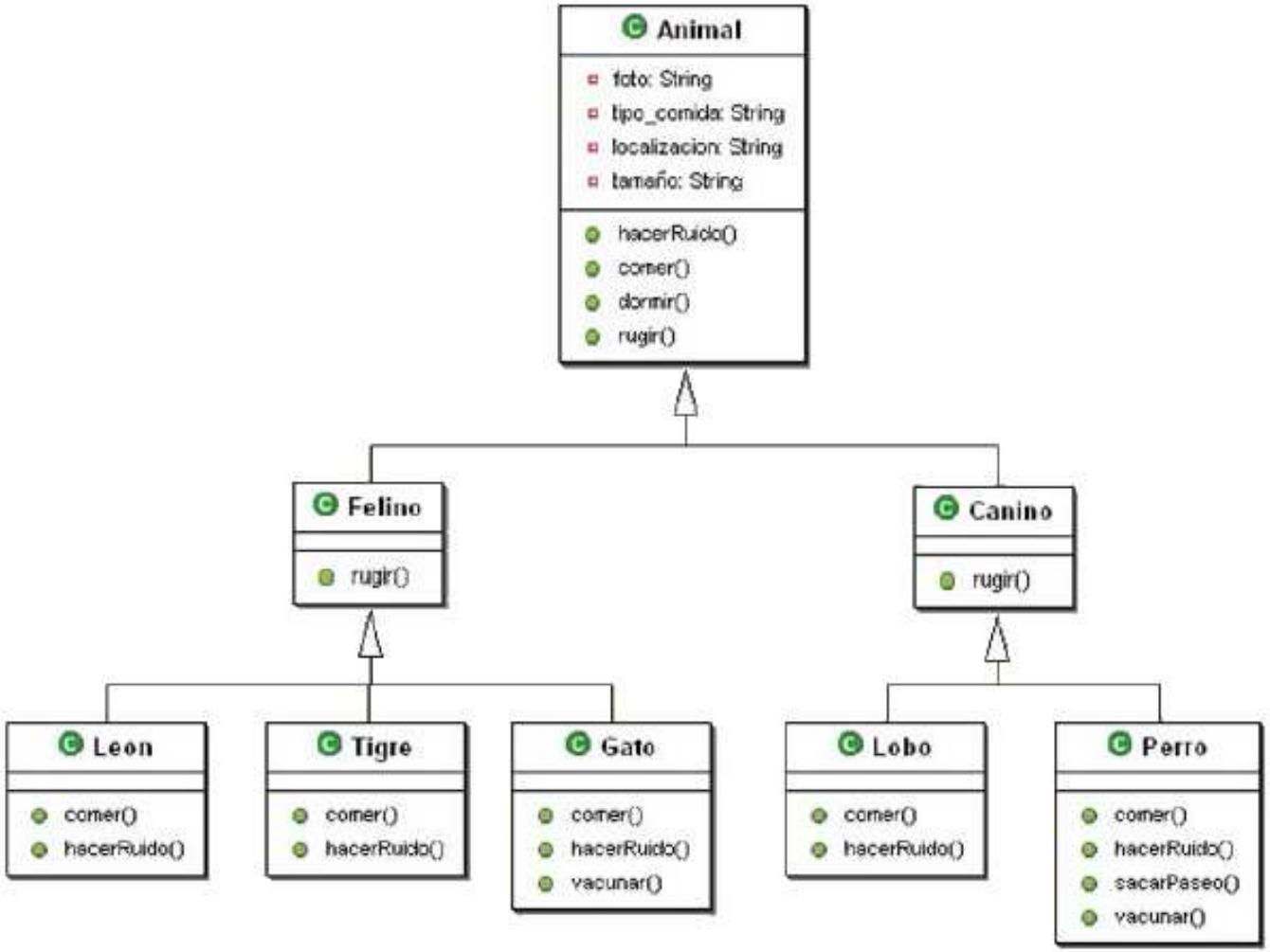
Tipos de herencia:

- Simple.
- Múltiple (no soportada en Java)

Unidad 4: Herencia

Herencia

Ejemplo



Unidad 4: Herencia

Herencia

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

Declaración de la herencia:

modificador_acceso class nom_clase extends nom_clase { }

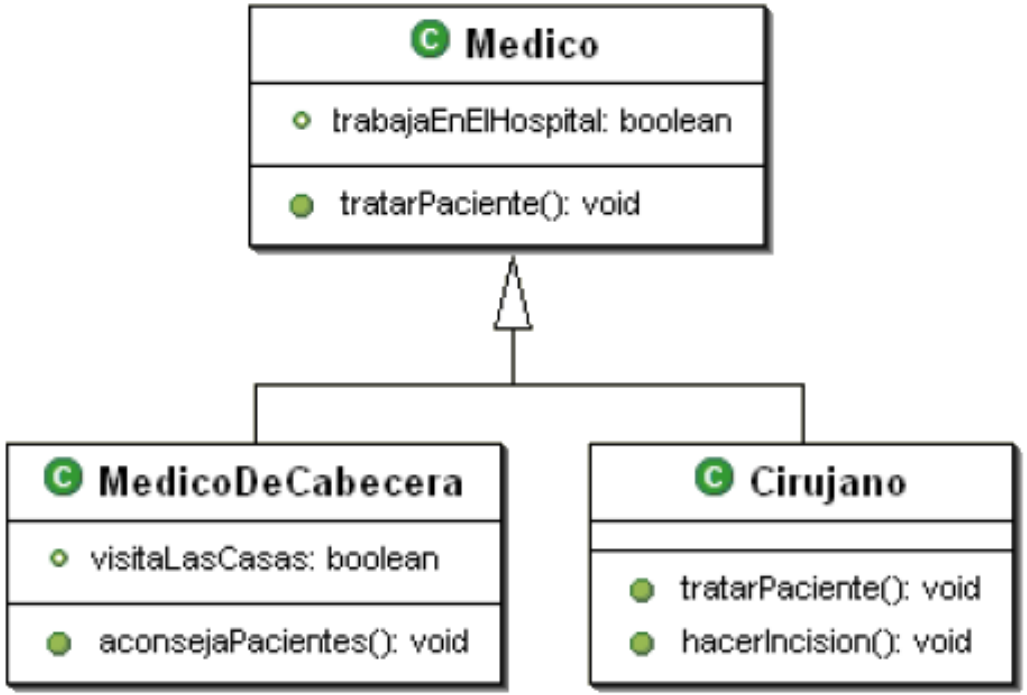
Ejemplo:

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

Unidad 4: Herencia

Herencia

Ejemplo



Unidad 4: Herencia

Herencia

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!).
  }
}
```

Unidad 4: Herencia

Herencia

Contesta a las siguientes preguntas basándote en el ejemplo anterior:

¿Cuántos atributos tiene la clase Cirujano?:__.

¿Cuántos atributos tiene la clase MedicoDeCabecera?:__.

¿Cuántos métodos tiene la clase Medico?:__.

¿Cuántos métodos tiene la clase Cirujano?:__.

¿Cuántos métodos tiene la clase MedicoDeCabecera?:__.

¿Puede un MedicoDeCabecera tratar pacientes?:___.

¿Puede un MedicoDeCabecera hacer incisiones?:___.

Herencia

Contesta a las siguientes preguntas basándote en el ejemplo anterior:

¿Cuántos atributos tiene la clase Cirujano?: 1.

¿Cuántos atributos tiene la clase MedicoDeCabecera?: 2.

¿Cuántos métodos tiene la clase Medico?: 1.

¿Cuántos métodos tiene la clase Cirujano?: 2.

¿Cuántos métodos tiene la clase MedicoDeCabecera?: 2.

¿Puede un MedicoDeCabecera tratar pacientes?: Si.

¿Puede un MedicoDeCabecera hacer incisiones?: No.

Unidad 4: Herencia

Herencia

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.
}
```

Unidad 4: Herencia

Herencia

Esto significa que nuestras clases **siempre** van a contar con los **atributos y métodos** de la clase Object.

Algunos de sus métodos más 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.

Unidad 4: Herencia

Herencia

public int hashCode();

Devuelve un identificador unívoco después de aplicarle un algoritmo hash.

public Object clone();

Devuelve una copia del objeto.

public void finalize();

Un método llamado por el Garbage Collector

Unidad 4: Herencia

Herencia

```java
public class MiClase {
}
```

```java
public class TestMiClase {

    public static void main(String[] args) {
        MiClase mc = new MiClase();
        System.out.println("MiClase: "+mc);
        System.out.println("toString(): "+mc.toString());
        System.out.println("hashCode(): "+mc.hashCode());
        System.out.println("getClass(): "+mc.getClass());
        System.out.println("equals(): "+mc.equals(mc));
    }

}
```

```
<terminated> TestMiClase [Java Applic
MiClase: MiClase@ea30797
toString(): MiClase@ea30797
hashCode(): 245565335
getClass(): class MiClase
equals(): true
```

Unidad 4: Herencia

Herencia

En

Sobrecarga de métodos

Unidad 4: Herencia

Sobrecarga de métodos

Sobrecargar un método es un concepto distinto a sobrescribir un método.

La **sobrecarga** de un método significa tener varias implementaciones del mismo método con parámetros distintos:

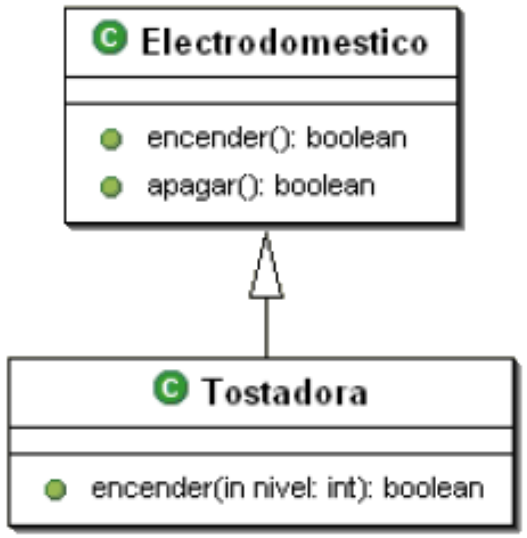
- El nombre ha de ser el mismo.
- El tipo de retorno puede ser distinto.
- Los parámetros tienen que ser distintos.
- El modificador de acceso puede ser distinto.

Sobrecarga de métodos

¿Compila?

```
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.
  }
}
```



Unidad 4: Herencia

Sobrecarga de métodos

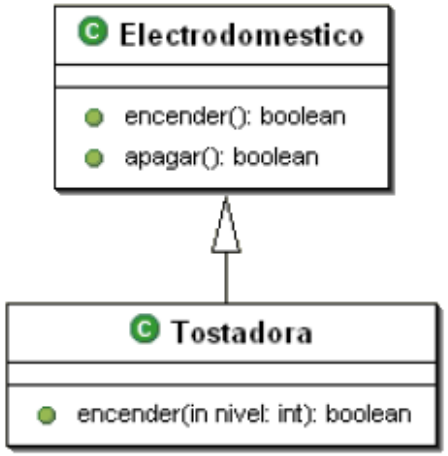
No es sobreescritura.

Los parámetros son distintos.

Es sobrecarga.

```
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.
  }
}
```



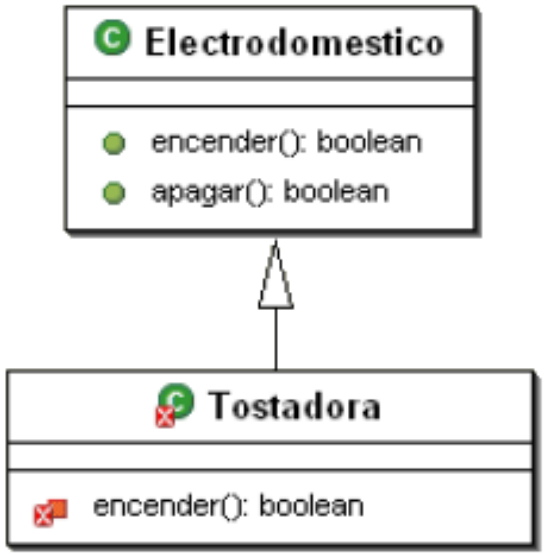
Unidad 4: Herencia

Sobrecarga de métodos

¿Compila?

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

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



Unidad 4: Herencia

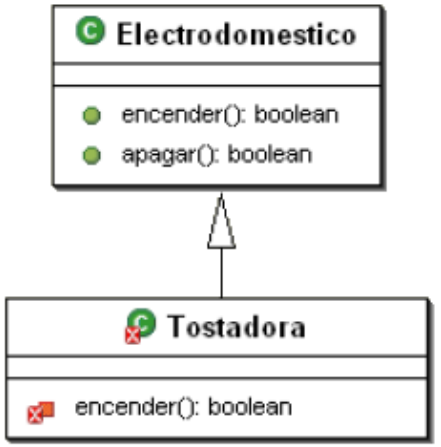
Sobrecarga de métodos

No compila.

Es sobreescritura restringiendo el acceso

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

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



Sobreescritura de métodos

Unidad 4: Herencia

Sobreescritura de métodos

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.

Unidad 4: Herencia

Sobreescritura de métodos

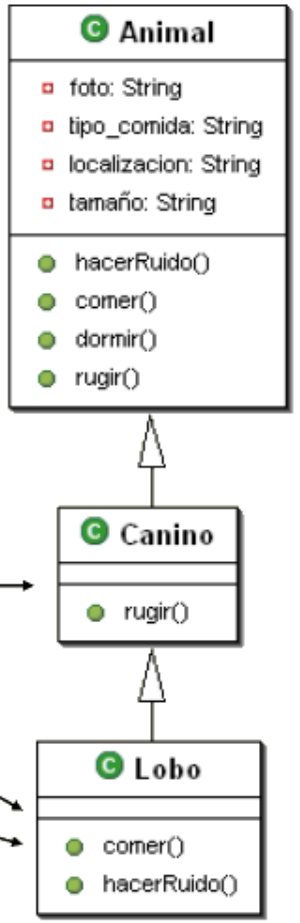
```
public class Test
{
  public static void main (String[] args)
  {
    Lobo lobo = new Lobo();

    lobo.hacerRuido();

    lobo.rugir();

    lobo.comer();

    lobo.dormir();
  }
```



Unidad 4: Herencia

Sobreescritura de métodos

```
public class Punto {
    private int x=0;
    private int y=0;

    public Punto(int valorX, int valorY) {
        this.x = valorX;
        this.y = valorY;
    }

}
```

```
public class TestPunto {

    public static void main(String[] args) {
        Punto punto1 = new Punto(1,2);
        Punto punto2 = new Punto(1,2);
        System.out.println("Punto 1: "+punto1);
        System.out.println("Punto 2: "+punto2);
        if (punto1.equals(punto2))
            System.out.println("Son iguales");
        else
            System.out.println("No son iguales");
    }

}
```

```
Punto 1: Punto@ea30797
Punto 2: Punto@58d25a40
No son iguales
```

Unidad 4: Herencia

Sobreescritura de métodos

```java
public class Punto {
    private int x=0;
    private int y=0;

    public Punto(int valorX, int valorY) {
        this.x = valorX;
        this.y = valorY;
    }

    @Override
    public boolean equals(Object obj) {
        Punto p = (Punto) obj;
        if ((this.x==p.x) && (this.y==p.y))
            return true;
        else
            return false;
    }

    @Override
    public String toString() {
        return "("+this.x+","+this.y+")";
    }

}
```

```java
public class TestPunto {

    public static void main(String[] args) {
        Punto punto1 = new Punto(1,2);
        Punto punto2 = new Punto(1,2);
        System.out.println("Punto 1: "+punto1);
        System.out.println("Punto 2: "+punto2);
        if (punto1.equals(punto2))
            System.out.println("Son iguales");
        else
            System.out.println("No son iguales");
    }

}
```

```
<terminated> TestPunto
Punto 1: (1,2)
Punto 2: (1,2)
Son iguales
```

Sobreescritura de métodos

Java 5.0 añadió una novedad al respecto. Se permite la sobreescritura de métodos cambiando también el tipo de retorno, pero siempre que:

- El **método** que se está sobrescribiendo **sea de una clase padre** (de la que heredamos directa o indirectamente).
- El **nuevo tipo de retorno sea hijo del tipo de retorno del método original** (es decir, que herede de él directa o indirectamente).

Por tanto, no es válido para tipos primitivos

super y this

Unidad 4: Herencia

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**.

Unidad 4: Herencia

super y this: Acceso a un atributo

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

```java
public class ClaseHija extends ClasePadre {
    boolean atributo = false;

    public void imprimir() {
        System.out.println(atributo);
        System.out.println(super.atributo);
    }
}
```

```java
public static void main(String[] args) {
    ClaseHija ch = new ClaseHija();
    ch.imprimir();
}
```

```
<terminated> Test (1)
false
true
```

Unidad 4: Herencia

super y this: Acceso a un constructor

```java
public class ClasePadre {
    public ClasePadre(int p1) {
        System.out.println(p1);
    }
}
```

```java
public class ClaseHija extends ClasePadre {

    public ClaseHija(int p1) {
        super(p1+2);
        System.out.println(p1);
    }
}
```

```java
public static void main(String[] args) {
    ClaseHija ch = new ClaseHija(7);
}
```

```
<terminated> Te
9
7
```

Unidad 4: Herencia

super y this: Acceso a un método

```java
public class ClasePadre {
    public void imprimir() {
        System.out.println("Método del padre");
    }
}
```

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

```java
public static void main(String[] args) {
    ClaseHija ch = new ClaseHija();
    ch.imprimir();
}
```

```
<terminated> Test (1) [Ja
Método del padre
Método del hijo
```

Unidad 4: Herencia

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.

Unidad 4: Herencia

super y this: 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);
    }
}


public static void main(String[] args) {
    MiClase mc = new MiClase();
    mc.setX(3);
}
```

```
<terminated> MiClase [Jav
x local vale: 3
x atributo vale: 5
x atributo vale: 3
```

Unidad 4: Herencia

super y this: Acceso a un constructor

```
public class MiClase {

    public MiClase() {
        this(2);
        System.out.println("Constructor sin parámetros");
    }

    public MiClase(int p) {
        System.out.println("Constructor con parámetros");
    }
}
```

```
public static void main(String[] args) {
    MiClase mc = new MiClase();
}
```

```
<terminated> Test (1) [Java Applicatio
Constructor con parámetros
Constructor sin parámetros
```

Modificadores de acceso

Unidad 4: Herencia

Modificadores de acceso

Existen cuatro tipos de modificadores de acceso y por tanto cuatro keywords:

public -> (público).
protected -> (protegido).
-> (paquete, identificado por la ausencia de keyword).
private -> (privado).

Están ordenados de **menor a mayor** restricción.

El modificador de acceso indica quién puede acceder a dicha clase, atributo o método.

Unidad 4: Herencia

Modificadores de acceso

Acceso a…	public	protected	package	private
Clases del mismo paquete	Si	Si	Si	No
Subclases de mismo paquete	Si	Si	Si	No
Clases de otros paquetes	Si	No	No	No
Subclases de otros paquetes	Si	Si	No	No

Unidad 4: Herencia

Modificadores de acceso

Los modificadores de acceso se utilizan en las definiciones de:

Clases e interfaces: solo se permiten public y package.

Atributos: se permiten cualquiera de los cuatro.

Métodos: se permiten cualquiera de los cuatro.