



Ministerio de Producción
Presidencia de la Nación

Ministerio de Educación y Deportes

Subsecretaría de Servicios Tecnológicos y Productivos



**PROGRAMACIÓN ORIENTADA
A OBJETOS**



Programación Orientada a Objetos

Temas:

- Palabra clave **this**
- Herencia

Uso de la palabra clave **this**

Es muy común usar la palabra clave **this** dentro de los métodos de instancia de una clase, para referirse al objeto que está ejecutando el código.

¿Por qué usar el **this**?

Típicamente dentro del cuerpo de un método nos podemos referir directamente a las variables miembros de un objeto por su nombre, sin embargo, a veces una variable miembro está oculta por un parámetro de un método que tiene el mismo nombre.

si no usamos el **this**, no asignará al atributo de la clase Persona el valor que recibe como parámetro.

```
package ejemplo;

public class Persona {
    private String apellido;
    . . .
    public void setApellido(String apellido){
        this.apellido = apellido;
    }
    . . .resto de los setters y getters
}
```

Uso de la palabra clave **this**

Si quiero hacer referencia a
algún atributo o método propio del objeto lo hago utilizando **this**

this.nombre = "Martin"



con esto lo que conceptualmente
estoy haciendo es:

objetoReceptor.nombre = "Martin"

Uso de la palabra clave **this**

La palabra clave **this** también es útil para pasar el objeto actual a otro método como parámetro.

```
public class A{  
  
    String nombre;  
  
    public boolean cumple(B b){  
        return b.acepta(this);  
    }  
  
    public String getNombre(){  
        return nombre;  
    }  
}
```

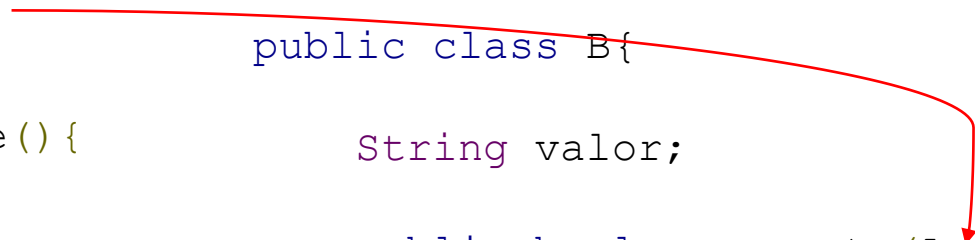
```
public class B{  
  
    String valor;  
  
    public boolean acepta(A a){  
        return valor.equals(a.getNombre());  
    }  
  
}
```

Uso de la palabra clave this

La palabra clave **this** también es útil para pasar el objeto actual a otro método como parámetro.

```
public class A{  
  
    String nombre;  
  
    public boolean cumple(B b){  
        return b.acepta(this);  
    }  
  
    public String getNombre(){  
        return nombre;  
    }  
}  
  
public class B{  
  
    String valor;  
  
    public boolean acepta(A a){  
        return valor.equals(a.getNombre());  
    }  
}
```

this está representando al objeto de tipo A sobre el cual se invocó el método cumple



Uso de la palabra clave **this**

La palabra clave **this** también es útil para invocar constructores desde otros constructores.

- Cuando se definen múltiples constructores para una clase, hay ocasiones en las que se desea invocar un constructor desde otro para evitar duplicar código. Eso se puede hacer utilizando la palabra **this**.
- En los constructores, la palabra **this** no hace referencia a un objeto, sino que hace un llamado explícito al constructor que coincide con la lista de parámetros dada.

```
public class Producto{  
  
    String nombre;  
    int id;  
    int stock;  
  
    public Producto(String nombre, int id, int stock){  
        this.nombre = nombre;  
        this.id = id;  
        this.stock = stock;  
    }  
  
    public Producto(String nombre, int id){  
        this(nombre,id,0);  
    }  
}
```

Uso de la palabra clave **this**

La palabra clave **this** también es útil para invocar constructores desde otros constructores.

- Cuando se definen múltiples constructores para una clase, hay ocasiones en las que se desea invocar un constructor desde otro para evitar duplicar código. Eso se puede hacer utilizando la palabra **this**.
- En los constructores, la palabra **this** no hace referencia a un objeto, sino que hace un llamado explícito al constructor que coincide con la lista de parámetros dada.

```
public class Producto{  
  
    String nombre;  
    int id;  
    int stock;  
  
    public Producto(String nombre, int id, int stock){  
        this.nombre = nombre;  
        this.id = id;  
        this.stock = stock;  
    }  
  
    public Producto(String nombre, int id){  
        this(nombre,id,0);  
    }  
}
```

Desde el constructor que recibe 2 parámetros e está invocando al constructor que recibe 3 parámetros

Uso de la palabra clave this

```
Search Project Run Window Help

Vehiculo.java *Auto.java X Camion.java Moto.java TestVehiculo.java

package claseCuatro.herencia;

public class Auto extends Vehiculo {

    int cantPuertas;

    public Auto(String numeroPatente, String propietario) {
        super(numeroPatente, propietario);
    }

    public Auto(String numeroPatente, String propietario, int cantPuertas) {
        super(numeroPatente, propietario);
        cantPuertas = cantPuertas;
        this.cantPuertas = cantPuertas;
    }

    public int getCantPuertas() {
        return cantPuertas;
    }

    public void setCantPuertas(int cantPuertas) {
        this.cantPuertas = cantPuertas;
    }
}
```

idénticos nombres

parámetro =
parámetro
se asigna así mismo

variable = parámetro



Herencia

El concepto de **herencia** refiere al hecho de transmitir “algo” desde un organismo a otro.

Supongamos que María tiene un hijo entonces esperamos que el hijo de María, herede “cosas” de ella.

¿Qué cosas esperamos que herede?



Herencia

El concepto de **herencia** refiere al hecho de transmitir “algo” desde un organismo a otro.

Supongamos que María tiene un hijo entonces esperamos que el hijo de María, herede “cosas” de ella.

¿Qué cosas esperamos que herede?

Rasgos Físicos

Color de Piel

Color de Ojos



Herencia

El concepto de **herencia** refiere al hecho de transmitir “algo” desde un organismo a otro.

Supongamos que María tiene un hijo entonces esperamos que el hijo de María, herede “cosas” de ella.

¿Qué cosas esperamos que herede?

Rasgos Físicos

Color de Ojos

Color de Piel

Nacionalidad

Apellido

Herencia

La programación orientada a objetos permite a las clases expresar similitudes entre objetos que tienen algunas características y comportamiento común.

Estas similitudes pueden expresarse usando **herencia**.

El término **herencia** se refiere al hecho de que una clase **hereda** los atributos (variables) y el comportamiento (métodos) de otra clase.



Herencia

Si tenemos que modelar una clase **Teléfono**:

- ¿Cómo la modelaríamos?
- ¿Cuáles serían sus atributos?
número - marca
- ¿Cuáles sería su comportamiento?
sonar - llamar



Herencia

Si tenemos que modelar una clase **Teléfono Celular**:

- ¿Cómo la modelaríamos?
- ¿Cuáles serían sus atributos?
número - marca - antena
- ¿Cuáles serían sus comportamientos?
sonar - llamar - enviar mensajes



Herencia

Si tenemos que modelar una clase **Teléfono Celular**:

- ¿Cómo la modelaríamos?
- ¿Cuáles serían sus atributos?
número - marca - ubicación
- ¿Cuál sería su comportamiento?
**recibir crédito - llamar
controlar crédito**

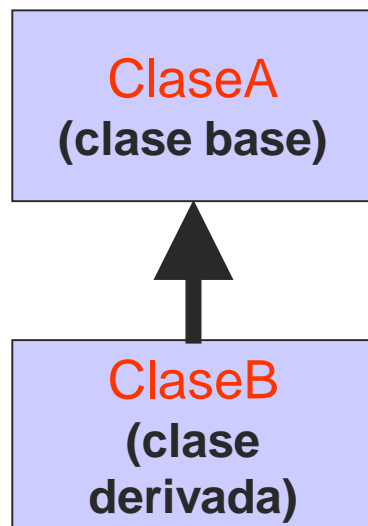


Herencia

Cada uno de estos tipos de teléfonos que hemos enumerados tienen “**características comunes**”.

Podemos observar claramente que tanto Teléfono Celular como Teléfono Público **heredan** atributos y comportamientos del objeto Teléfono.

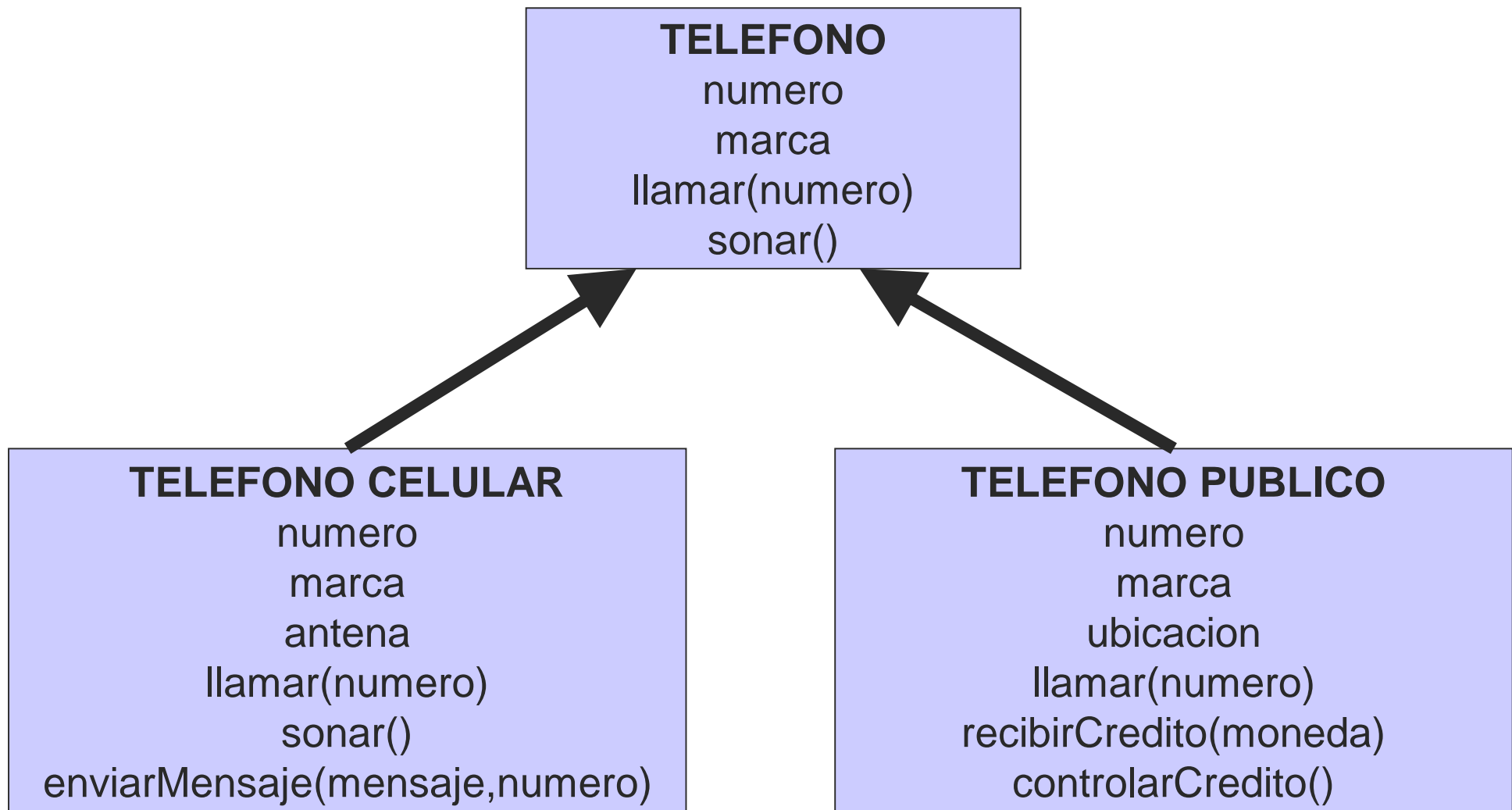
Diagrama de clases con herencia



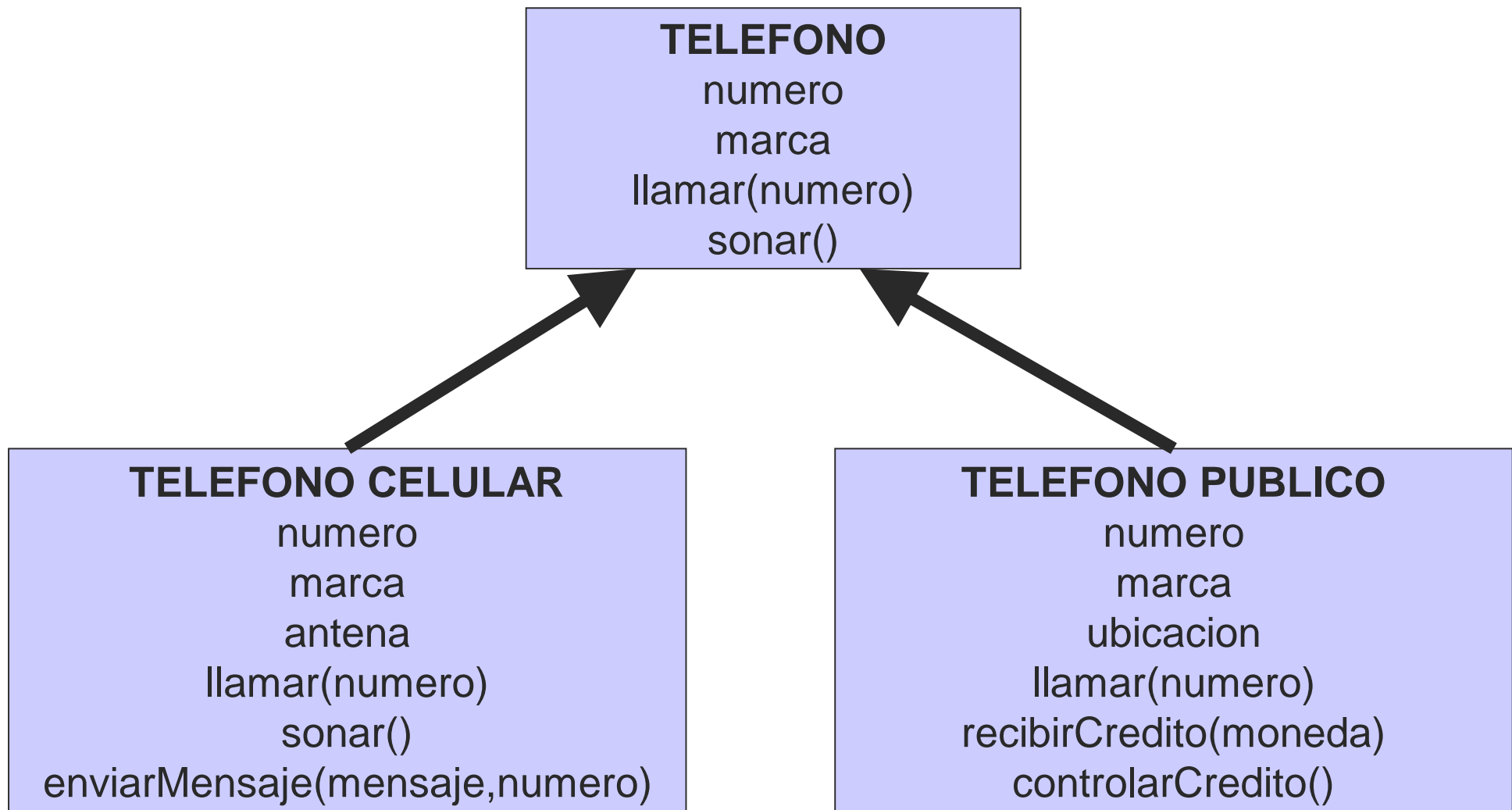
La clase **ClaseB** es *subclase* de la clase **ClaseA**
La clase **ClaseA** es la *superclase* de la clase **ClaseB**

Herencia

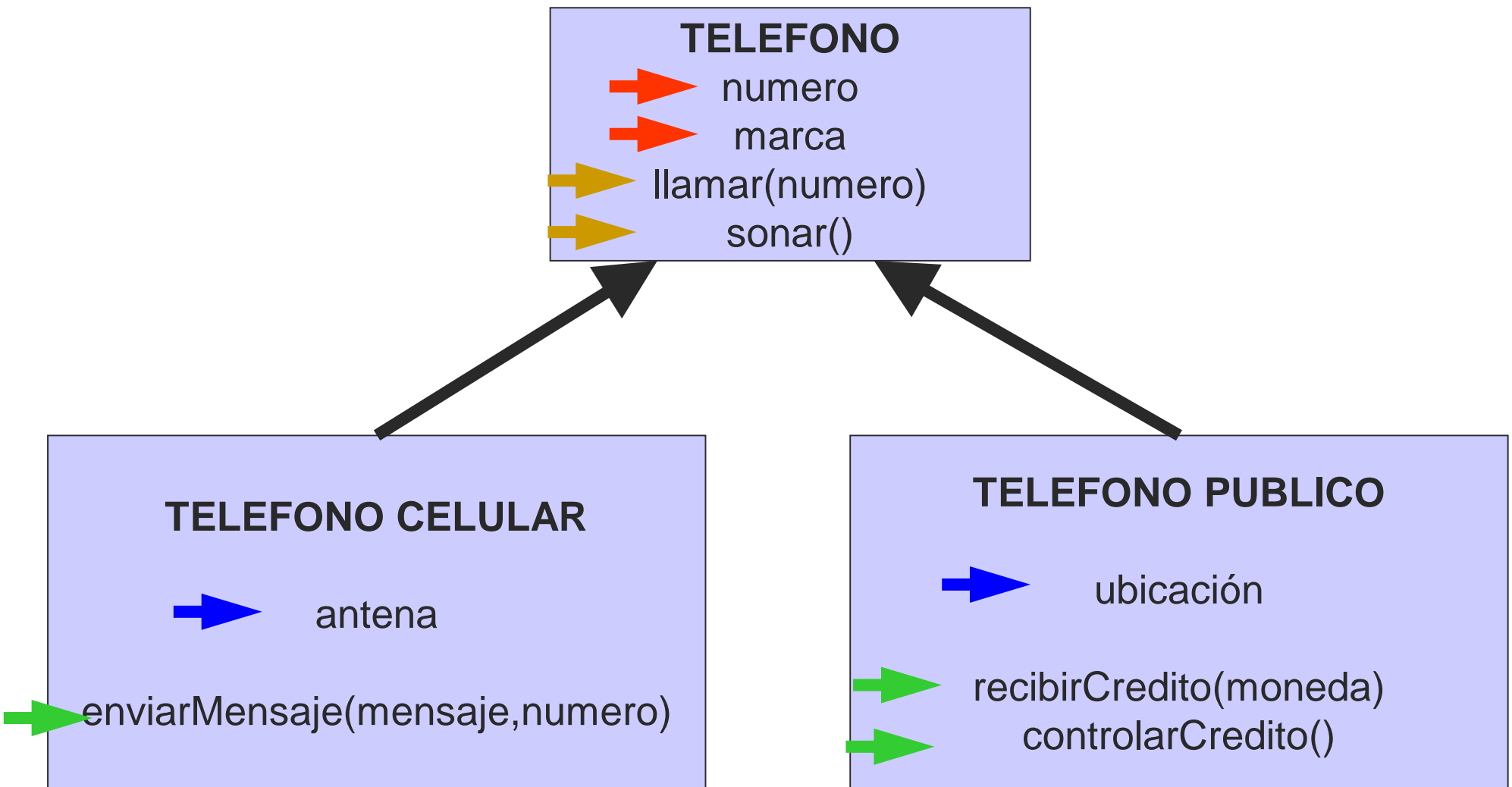
Representemos a los diferentes Objetos Teléfonos que tenemos



Herencia



Herencia





Ministerio de
Educación y Deportes
Presidencia de la Nación



Ministerio de Producción
Presidencia de la Nación

Herencia

Entonces... ¿En qué consiste?



Herencia

Entonces... ¿En qué consiste?

La herencia toma una clase existente
y construye una versión especializada
reusabilidad de código

En la clase **hija** se definen las diferencias respecto de la clase **padre**.



Herencia

Entonces... ¿En qué consiste?

La herencia toma una clase existente
y construye una versión especializada
reusabilidad de código

En la clase **hija** se definen las diferencias respecto de la clase **padre**.

¿Para qué se usa?

Herencia

Entonces... ¿En qué consiste?

La herencia toma una clase existente
y construye una versión especializada
reusabilidad de código

En la clase **hija** se definen las diferencias respecto de la clase **padre**.

¿Para qué se usa?

- Para extender la funcionalidad de la clase padre.
- Para especializar el comportamiento de la clase padre.

Herencia

Entonces... ¿En qué consiste?

La herencia toma una clase existente
y construye una versión especializada
reusabilidad de código

En la clase **hija** se definen las diferencias respecto de la clase **padre**.

¿Para qué se usa?

- Para extender la funcionalidad de la clase padre.
- Para especializar el comportamiento de la clase padre.

Usando Herencia **SIMPLE** toda clase **SIEMPRE**
HEREDA de una **ÚNICA** clase



Herencia

Ventajas

- Permite reutilizar código extendiendo su funcionalidad.
- Evita duplicar código.

Desventajas

- Puede dificultar la reutilización.
- Un cambio en la clase padre puede tener efectos imprevistos en las clases hijas.
- Un objeto de una clase hija puede tener un comportamiento inconsistente con lo esperado de un objeto de la clase padre.



Herencia

Ventajas

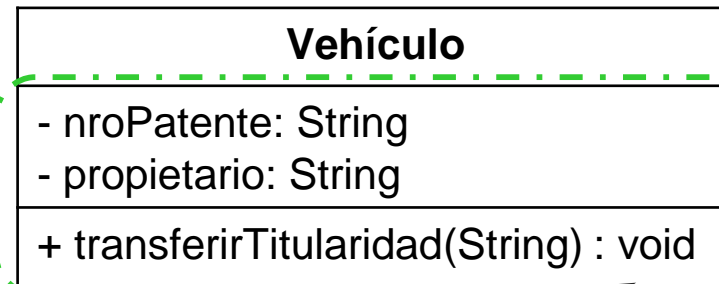
- Permite reutilizar código extendiendo su funcionalidad.
- Evita duplicar código.

Herencia

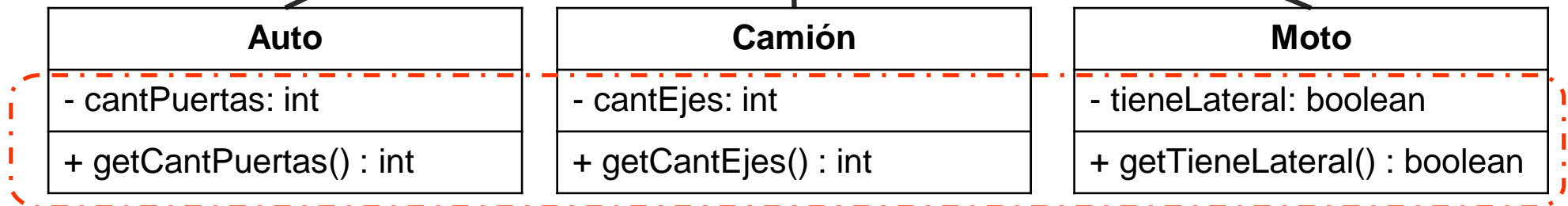
Una subclase puede **agregar** atributos y comportamiento a su superclase y **reemplazar** o **modificar** el comportamiento heredado.

superclase

Variables y métodos de instancia
comunes a todos los vehículos



subclases

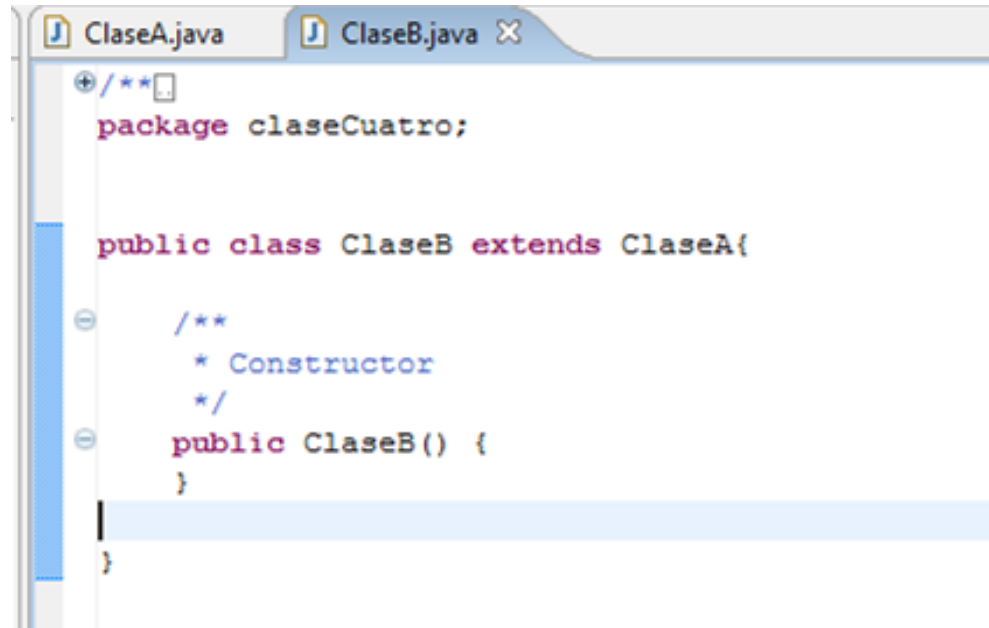
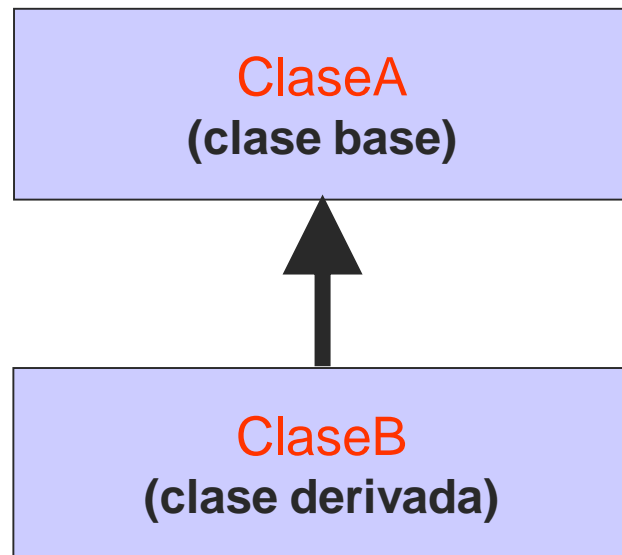


Variables y métodos de instancia **ESPECÍFICOS** de cada
tipo de vehículo

Herencia

¿Cómo indicamos la relación de herencia en el código fuente Java?

Con la palabra clave **extends**



```
ClaseA.java  ClaseB.java X
+/**
package claseCuatro;

public class ClaseB extends ClaseA{

    /**
     * Constructor
     */
    public ClaseB() {
    }
}
```

The screenshot shows a Java IDE with two tabs: "ClaseA.java" and "ClaseB.java". The "ClaseB.java" tab is active, displaying the following code:

Herencia

```
public class Vehiculo {  
    private String nroPatente;  
    private String propietario;  
    public void transferirTitularidad(String nuevoTit) {  
        . . .  
    }  
}
```

clase base

clases derivadas

```
public class Auto extends Vehiculo {  
    private int cantPuertas;  
    public int getCantPuertas() {  
        . . .  
    }  
    . . .  
}
```

```
public class Moto extends Vehiculo {  
    private boolean tieneLateral;  
    public boolean getTieneLateral() {  
        . . .  
    }  
    . . .  
}
```

```
public class Camion extends Vehiculo {  
    private int cantEjes;  
    public int getCantEjes() {  
        . . .  
    }  
    . . .  
}
```

Automáticamente, la subclase obtiene las variables y métodos de la superclase



Herencia

¿A qué pueden acceder las clases hijas?

- Los miembros privados de la clase padre (superclase) no son visibles desde las clases hijas (subclases).
- Los miembros públicos de la superclase son visibles y siguen siendo públicos en la subclase.
- Se puede acceder a los miembros de la superclase usando la palabra reservada **super**.



Herencia

- Los atributos no se pueden redefinir, sólo se ocultan.
 - Si la clase hija define un atributo con el mismo nombre que un atributo de la clase padre, éste no está accesible.
 - El atributo de la superclase todavía existe pero no se puede acceder.
- Un método de la subclase con la misma signatura (nombre y parámetros) que un método de la superclase lo está redefiniendo.
 - Por el contrario, si se cambia el tipo o cantidad de parámetros se está sobrecargando el método original.



Herencia

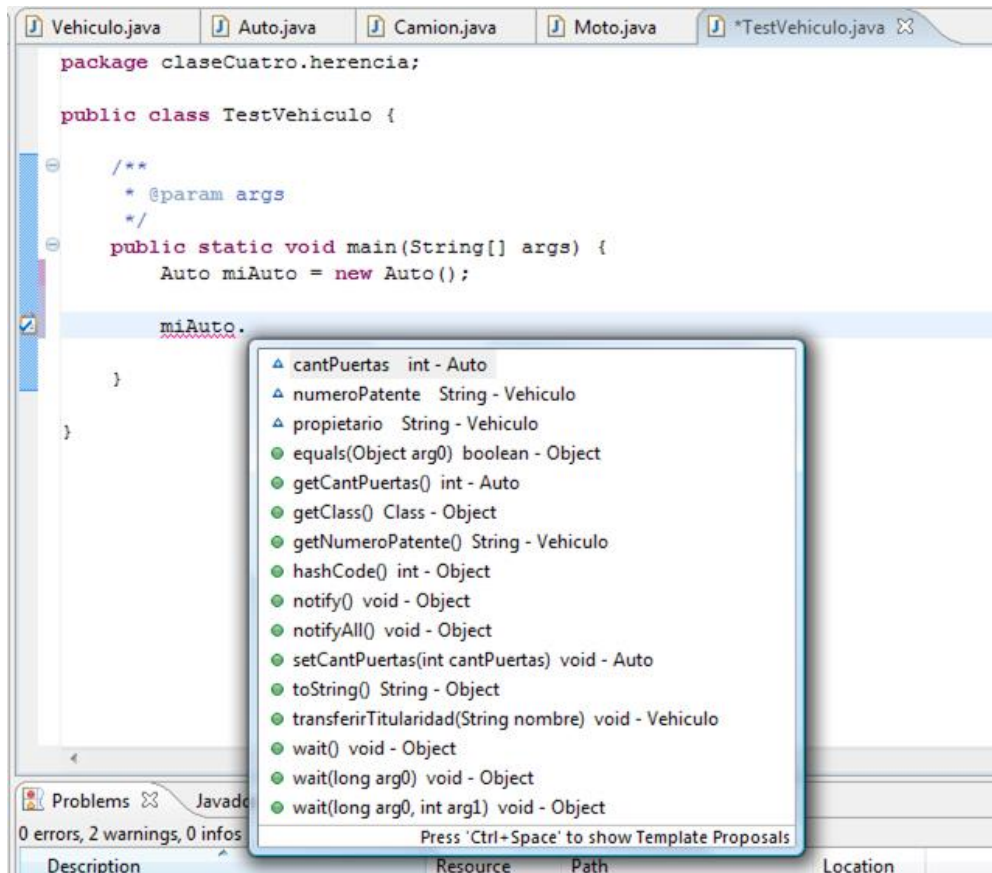
- Los atributos no se pueden redefinir, sólo se ocultan.
 - Si la clase hija define un atributo con el mismo nombre que un atributo de la clase padre, éste no está accesible.
 - El atributo de la superclase todavía existe pero no se puede acceder.
- Un método de la subclase con la misma signatura (nombre y parámetros) que un método de la superclase lo está redefiniendo.
 - Por el contrario, si se cambia el tipo o cantidad de parámetros se está sobrecargando el método original.

Una subclase puede redefinir un método de la superclase por dos motivos:

- **Reemplazo.** Se sustituye completamente la implementación del método heredado manteniendo la semántica.
- **Refinamiento.** Se añade nueva funcionalidad al comportamiento heredado.
 - Resulta útil invocar a la versión heredada del método, utilizando la palabra reservada **super**.

Herencia

Observemos en nuestro ejemplo como se ve la herencia



```
package claseCuatro.herencia;

public class TestVehiculo {

    /**
     * @param args
     */
    public static void main(String[] args) {
        Auto miAuto = new Auto();

        miAuto.

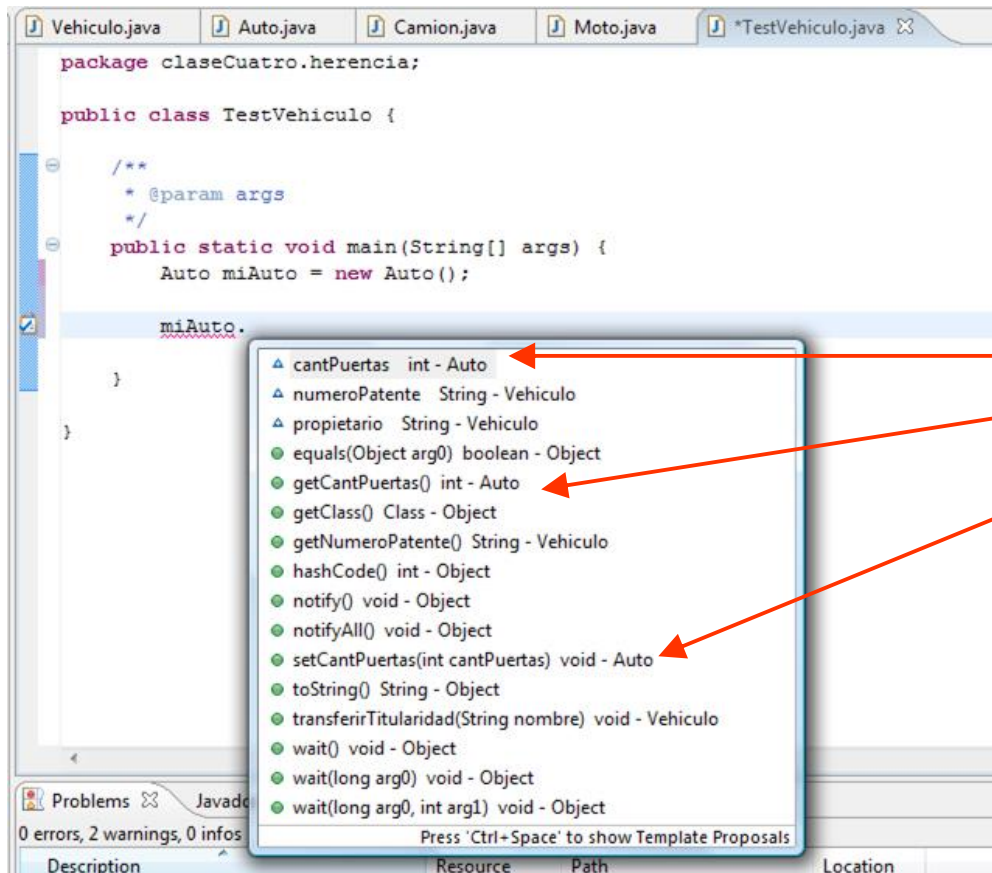
    }
}
```

- ▲ cantPuertas int - Auto
- ▲ numeroPatente String - Vehiculo
- ▲ propietario String - Vehiculo
- equals(Object arg0) boolean - Object
- getCantPuertas() int - Auto
- getClass() Class - Object
- getNumeroPatente() String - Vehiculo
- hashCode() int - Object
- notify() void - Object
- notifyAll() void - Object
- setCantPuertas(int cantPuertas) void - Auto
- toString() String - Object
- transferirTitularidad(String nombre) void - Vehiculo
- wait() void - Object
- wait(long arg0) void - Object
- wait(long arg0, int arg1) void - Object

Press 'Ctrl+Space' to show Template Proposals

Herencia

Observemos en nuestro ejemplo como se ve la herencia



```
package claseCuatro.herencia;

public class TestVehiculo {

    /**
     * @param args
     */
    public static void main(String[] args) {
        Auto miAuto = new Auto();

        miAuto.

    }
}
```

Methods in TestVehiculo:

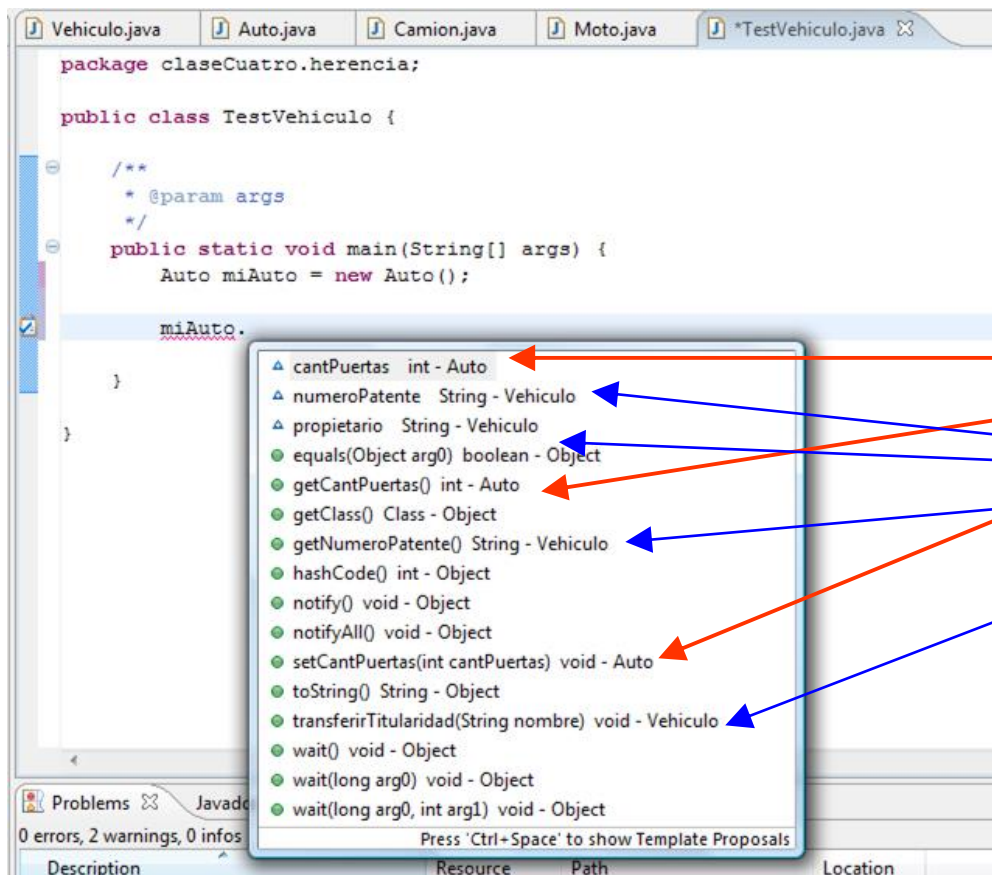
- ▲ cantPuertas int - Auto
- ▲ numeroPatente String - Vehiculo
- ▲ propietario String - Vehiculo
- equals(Object arg0) boolean - Object
- getCantPuertas() int - Auto
- getClass() Class - Object
- getNumeroPatente() String - Vehiculo
- hashCode() int - Object
- notify() void - Object
- notifyAll() void - Object
- setCantPuertas(int cantPuertas) void - Auto
- toString() String - Object
- transferirTitularidad(String nombre) void - Vehiculo
- wait() void - Object
- wait(long arg0) void - Object
- wait(long arg0, int arg1) void - Object

Press 'Ctrl+Space' to show Template Proposals

Propio de la clase
Auto

Herencia

Observemos en nuestro ejemplo como se ve la herencia



The screenshot shows an IDE with several Java files open: Vehiculo.java, Auto.java, Camion.java, Moto.java, and *TestVehiculo.java. The active file is TestVehiculo.java, which contains the following code:

```
package claseCuatro.herencia;

public class TestVehiculo {

    /**
     * @param args
     */
    public static void main(String[] args) {
        Auto miAuto = new Auto();

        miAuto.
    }
}
```

A popup window displays the class hierarchy for the `miAuto` object. The hierarchy is as follows:

- `cantPuertas int - Auto` (Red arrow from "Propio de la clase Auto")
- `numeroPatente String - Vehiculo` (Blue arrow from "Heredado de Vehículo")
- `propietario String - Vehiculo` (Blue arrow from "Heredado de Vehículo")
- `equals(Object arg0) boolean - Object` (Blue arrow from "Heredado de Vehículo")
- `getCantPuertas() int - Auto` (Red arrow from "Propio de la clase Auto")
- `getClass() Class - Object` (Blue arrow from "Heredado de Vehículo")
- `getNumeroPatente() String - Vehiculo` (Blue arrow from "Heredado de Vehículo")
- `hashCode() int - Object` (Blue arrow from "Heredado de Vehículo")
- `notify() void - Object` (Blue arrow from "Heredado de Vehículo")
- `notifyAll() void - Object` (Blue arrow from "Heredado de Vehículo")
- `setCantPuertas(int cantPuertas) void - Auto` (Red arrow from "Propio de la clase Auto")
- `toString() String - Object` (Blue arrow from "Heredado de Vehículo")
- `transferirTitularidad(String nombre) void - Vehiculo` (Blue arrow from "Heredado de Vehículo")
- `wait() void - Object` (Blue arrow from "Heredado de Vehículo")
- `wait(long arg0) void - Object` (Blue arrow from "Heredado de Vehículo")
- `wait(long arg0, int arg1) void - Object` (Blue arrow from "Heredado de Vehículo")

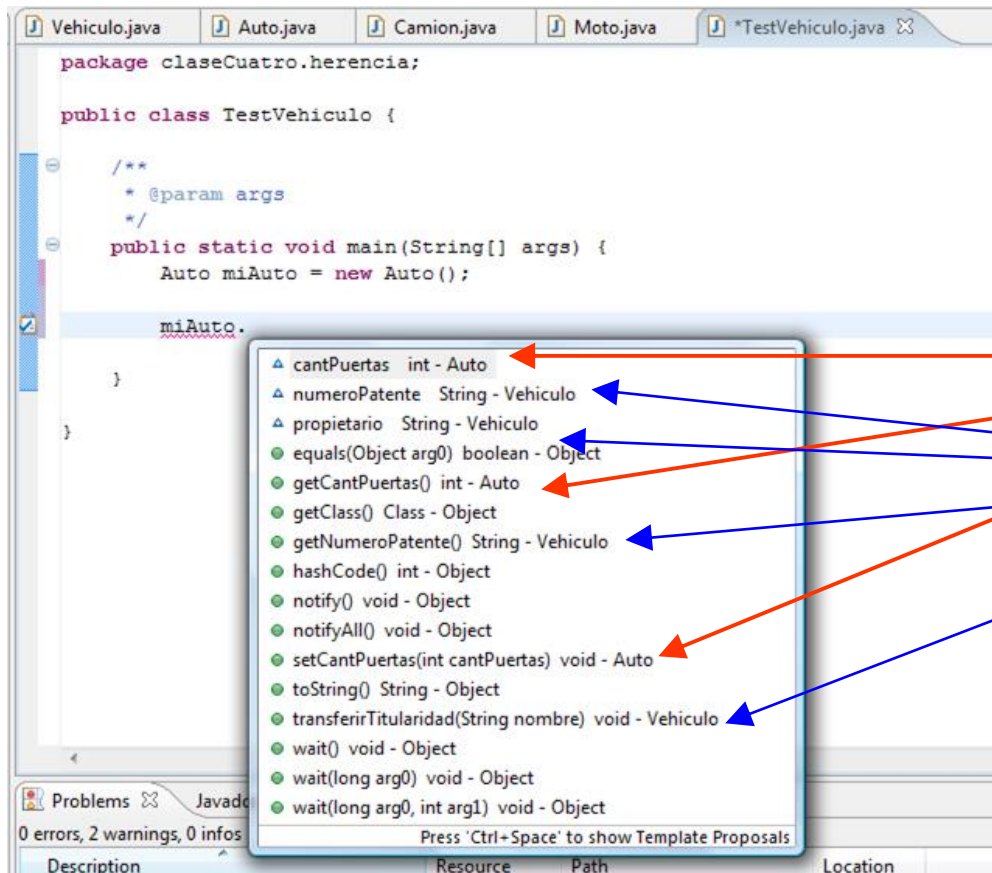
At the bottom of the popup, it says "Press 'Ctrl+Space' to show Template Proposals".

Propio de la clase
Auto

Heredado de Vehículo

Herencia

Observemos en nuestro ejemplo como se ve la herencia



```
package claseCuatro.herencia;

public class TestVehiculo {

    /**
     * @param args
     */
    public static void main(String[] args) {
        Auto miAuto = new Auto();

        miAuto.
    }
}
```

Methods in the tooltip:

- ▲ cantPuertas int - Auto
- ▲ numeroPatente String - Vehiculo
- ▲ propietario String - Vehiculo
- equals(Object arg0) boolean - Object
- getCantPuertas() int - Auto
- getClass() Class - Object
- getNumeroPatente() String - Vehiculo
- hashCode() int - Object
- notify() void - Object
- notifyAll() void - Object
- setCantPuertas(int cantPuertas) void - Auto
- toString() String - Object
- transferirTitularidad(String nombre) void - Vehiculo
- wait() void - Object
- wait(long arg0) void - Object
- wait(long arg0, int arg1) void - Object

Propio de la clase
Auto

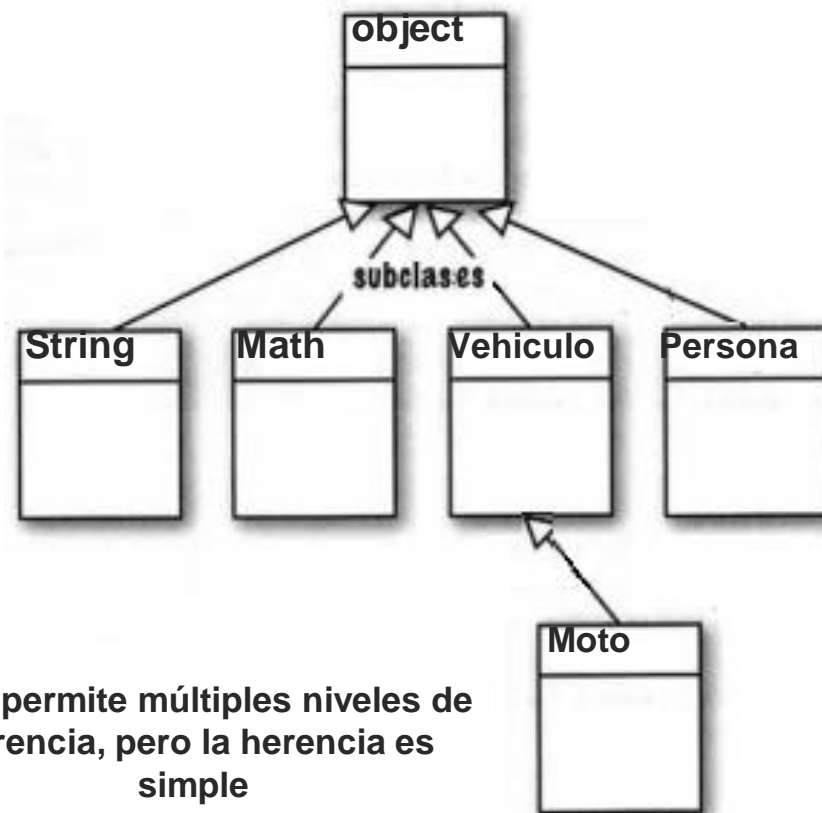
Heredado de Vehículo

Heredado de Object
(el resto de los
métodos)

Herencia

La clase `java.lang.Object` es la raíz de la jerarquía de clases en Java.

Cualquier clase que no especifique un padre directo, será subclase directa de `Object`.



Java permite múltiples niveles de herencia, pero la herencia es simple

```
public class Persona {  
    private String apellido;  
    private String nombres;  
    . . .  
}
```



Referencias

- Pensando en Java 3a edición Español, Bruce Eckel. Capítulo 6.



Ministerio de Producción
Presidencia de la Nación

Ministerio de Educación y Deportes

Subsecretaría de Servicios Tecnológicos y Productivos



**PROGRAMACIÓN ORIENTADA
A OBJETOS**