Universidad
Carlos III de Madrid

Tema 8

Introducción a la Programación Orientada a Objetos (POO)

Programación

2015-2016

Tema 8. Prog. Orientada a Objetos

- **Tipos abstractos de datos.**
- **Encapsulación, Herencia y Polimorfismo.**
- **Objetos.**

Introducción

- Hemos realizado una primera incursión en Java pero para proseguir se nos hace indispensable hablar de conceptos fundamentales de la programación orientada a objetos: **objetos y clases**. Estos términos parecen resultarnos familiares.

- En la vida diaria podemos pensar en objetos como una manzana o un libro y podemos distinguir clases de cosas: por ejemplo clases de plantas.

- Al escribir un programa en un lenguaje orientado a objetos tratamos de modelar un problema del mundo real pensando en objetos que forman parte del problema y que se relacionan entre sí.

Concepto de objeto y clase

- Podemos decir como primera aproximación:
  - **Objeto**: entidad existente en la memoria del ordenador que tiene unas propiedades (atributos o datos sobre sí mismo almacenados por el objeto) y unas operaciones disponibles específicas (métodos).
  - **Clase**: abstracción que define un tipo de objeto especificando qué propiedades (atributos) y operaciones disponibles va a tener.

Tipos abstractos de datos

- Un tipo abstracto de datos es aquel cuyas operaciones relacionadas (**interfaz**) pueden describirse independientemente de su representación interna (**implementación**).

Tipos abstractos de datos

- Supongamos que al crear un programa creamos una ciudad. Disponemos de normas de urbanismo. Cada norma vamos a decir que es una interface: nos dice qué debemos cumplir para que al construir un edificio (clase) se pueda calificar con un nombre determinado. Supongamos una norma denominada “Edificio a dos aguas”, cuyo contenido incluye:
  - El edificio habrá de tener cuatro paredes.
  - El edificio habrá de tener un tejado formado por dos planos.
  - Otras especificaciones.
- Si al construir un edificio (implementación) se cumplen las condiciones de las normas de urbanismo, podremos decir que se trata de un edificio a dos aguas. Un edificio con forma de pentágono y cinco paredes no se denominará “Edificio a dos aguas”. Por el contrario, sería posible denominar edificio a dos aguas a una capilla que cumpla la norma, también a una vivienda unifamiliar que la cumpla, o a una biblioteca que la cumpla.

Clases

- DEFINICIÓN:
  - Implementación total o parcial de un Tipo Abstracto de Datos.
- Entidad sintáctica que **describe objetos** que van a tener la **misma estructura y el mismo comportamiento**.
- Doble naturaleza: Módulo + Tipo de Datos
  - Módulo (concepto sintáctico)
    - Mecanismo para **organizar el software**
  - Tipo (concepto semántico)
    - Mecanismo de definición de **nuevos tipos de datos:** describe una estructura de datos (objetos) y las operaciones aplicables.

Componentes de una clase

- **Atributos**:
  - Determinan una estructura de almacenamiento para cada objeto de la clase (variables)
- **Métodos**:
  - Operaciones aplicables a los objetos
  - Único modo de acceder a los atributos

Tema 8. Prog. Orientada a Objetos

- Tipos abstractos de datos.
- **Encapsulación, Herencia y Polimorfismo.**
- Objetos.

Programación Orientada a Objetos

Características de la POO:

- **Encapsulación.**
- **Herencia.**
- **Polimorfismo.**

Encapsulación

Ocultamiento del estado interno de un objeto, de modo que sólo puede ser cambiado mediante las operaciones definidas para ese objeto.

Es decir, no podremos acceder directamente a los atributos de un objeto (modificador *private*). Lo haremos mediante:

- **Métodos modificadores**: llamamos métodos modificadores a aquellos métodos que dan lugar a un cambio en el valor de uno o varios de los atributos del objeto.

  Suelen ir precedidos del prefijo set (setMatricula, setDistrito, etc.) por lo que muchas veces se alude coloquialmente a ellos como “métodos set” o “setters”.
- **Métodos consultores u observadores**: son métodos que devuelven información sobre el contenido de los atributos del objeto sin modificar los valores de estos atributos.

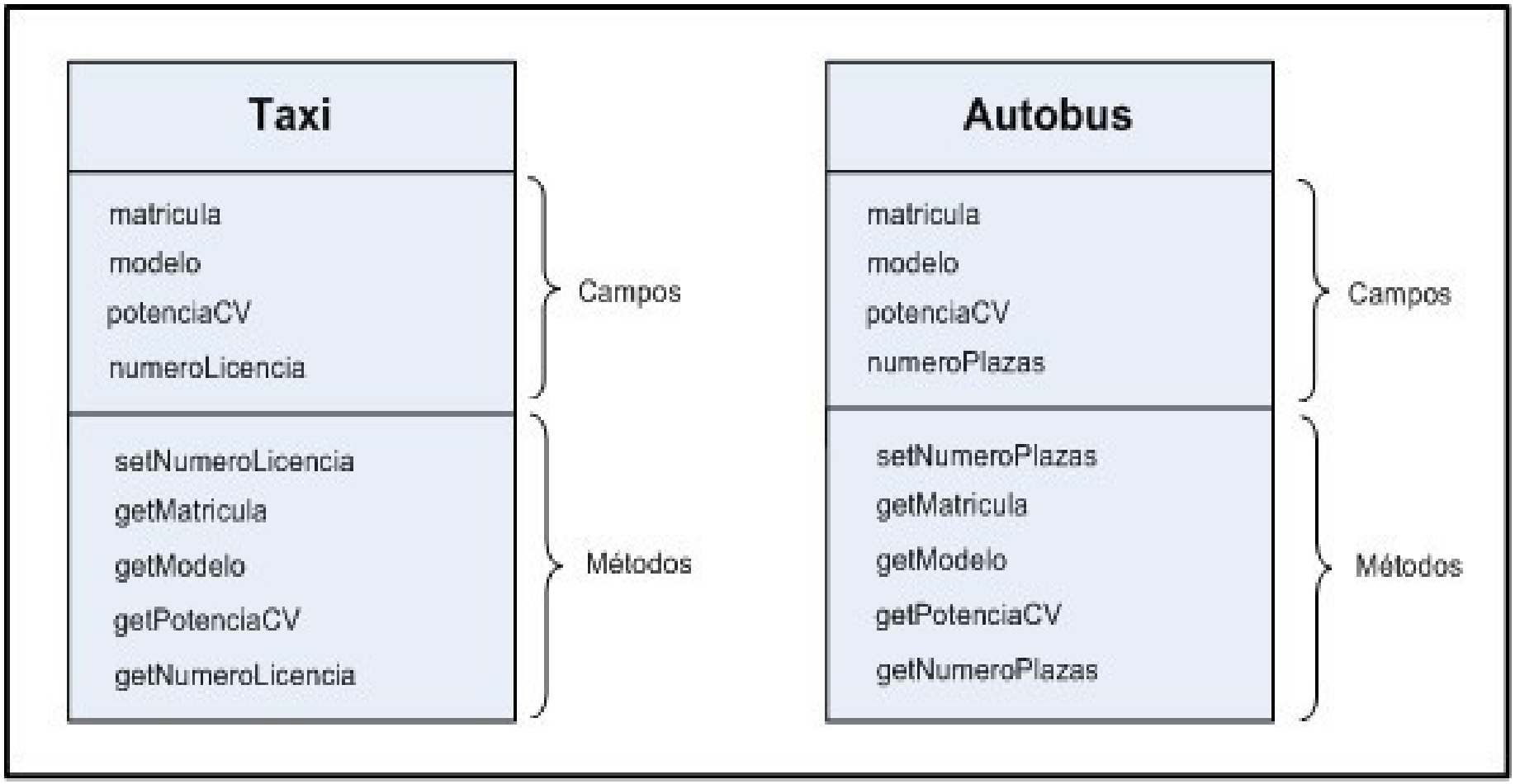
  Estos métodos suelen ir precedidos del prefijo get (getMatricula, getDistrito, etc.) por lo que muchas veces se alude coloquialmente a ellos como “métodos get” o “getters”.

Encapsulación

- A las características de una clase (atributos y métodos) se les puede asignar un modificador de visibilidad:
  - **public:**
    - Característica pública, accesible desde todas las clases
  - **private:**
    - Característica privada, accesible sólo dentro de la clase donde se define
- Principio de diseño:
  - Todos los atributos de una clase son privados
  - Los métodos pueden tener distintos niveles de visibilidad

Herencia

Muchas veces distintos objetos comparten campos y métodos que hacen aproximadamente lo mismo



Herencia

Si nos fijamos en el planteamiento del problema, encontramos:

- La definición de clases nos permite identifica**r campos y métodos que son comunes** a Taxis y Autobuses. Si implementamos ambas clases, incurriremos en duplicidad de código. Por ejemplo si el campo matricula es en ambas clases un tipo String, el código para gestionar este campo será idéntico en ambas clases.
- La definición de clases nos permite identificar **campos y métodos que difieren** entre una clase y otra. Por ejemplo en la clase Taxi se gestiona información sobre un campo denominado numeroDeLicencia que no existe en la clase Autobus.
- Conceptualmente **podemos imaginar una abstracción** que engloba a Taxis y Autobuses: ambos podríamos englobarlos bajo la denominación de “Vehículos”. Un Taxi sería un tipo de Vehiculo y un Autobus otro tipo de Vehiculo.
- Al implementar tros vehículos como minibuses, tranvías, etc. seguiríamos engrosando la **duplicidad de código**. Por ejemplo, un minibús también tendría matrícula, potencia… y los métodos asociados.

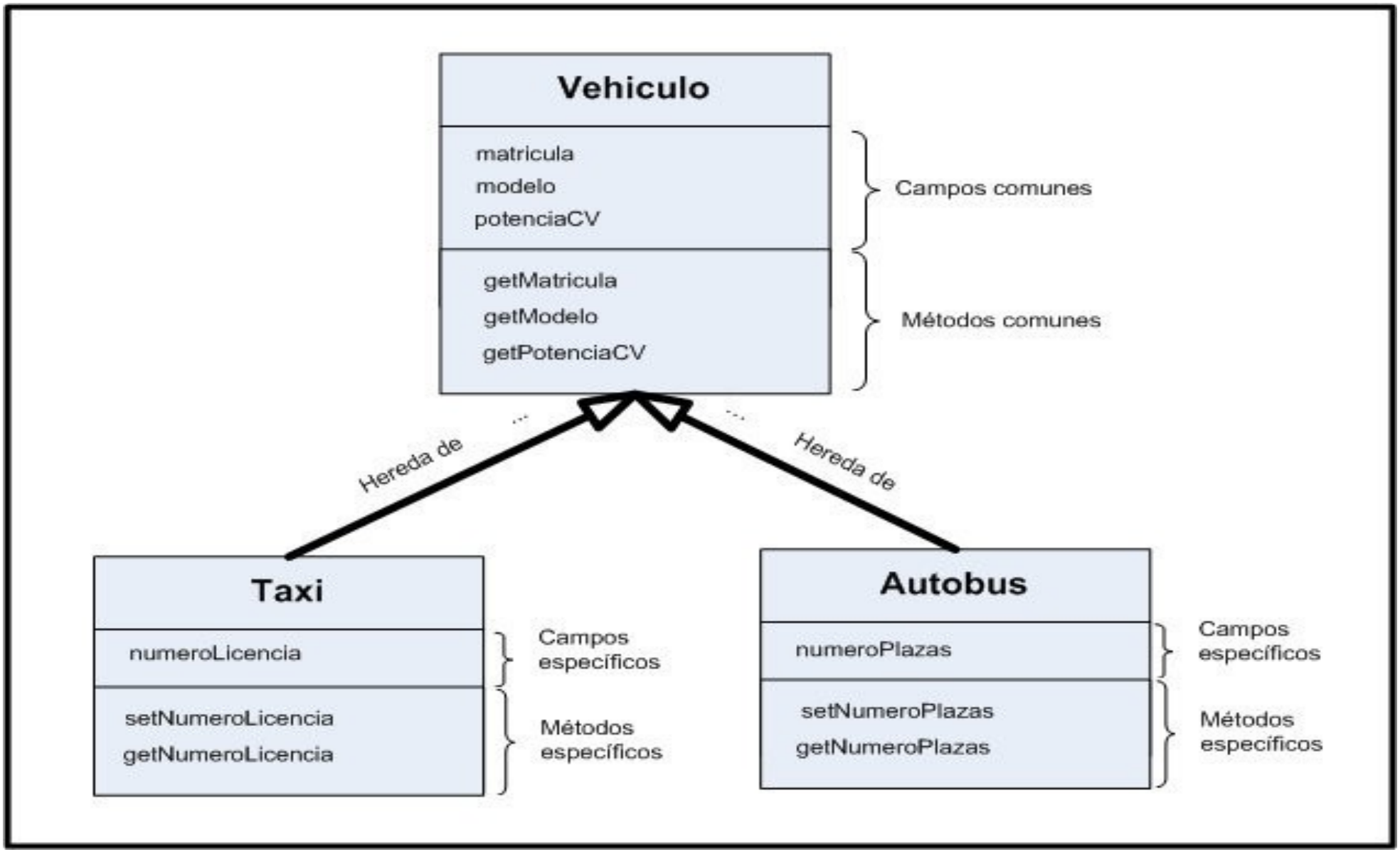
Herencia

¿No sería más razonable, si una propiedad o método va a ser siempre común para varios tipos de objetos, que estuviera localizada en un sitio único al que ambos tipos de objeto accedieran? En los lenguajes con orientación a objetos la solución a esta problemática se llama **herencia**.

- **La herencia nos permite definir una clase como extensión de otra:**
  - de esta manera decimos “la clase 1.1 tiene todas las características de la clase 1 y además sus características particulares”.
  - Todo lo que es común a ambas clases queda comprendido en la clase “superior”, mientras lo que es específico, queda restringido a las clases “inferiores”.

En nuestro ejemplo definiríamos una clase denominada Vehiculo, de forma que la clase Taxi tuviera todas las propiedades de la clase Vehiculo, más algunas propiedades y métodos específicos. Lo mismo ocurriría con la clase Autobus y otras que pudieran “heredar” de Vehiculo.

Herencia


Vehiculo
matricula
modelo
potenciaCV
getMatricula
getModelo
getPotenciaCV
Campos comunes
Métodos comunes
Hereda de
Hereda de
Taxi
numeroLicencia
setNumeroLicencia
getNumeroLicencia
Campos específicos
Métodos específicos
Autobus
numeroPlazas
setNumeroPlazas
getNumeroPlazas
Campos específicos
Métodos específicos

Herencia

- Para declarar la herencia en Java usamos la palabra clave extends. Ejemplo:
  - public class MiClase2 extends Miclase1.
- Si quisiéramos podríamos escribir para todas las clases **public class NombreDeLaClase extends Object**, aunque como es algo implícito a Java normalmente no lo escribiremos por ser redundante. En los diagramas representativos de la jerarquía de herencia ocurre lo mismo.
- Recordar que en Java los tipos primitivos no son objetos: no son instancias de clase, y por tanto no heredan de la superclase Object.
- Los campos privados de una superclase no son accesibles por la subclase directamente. Decimos que la subclase no tiene derechos de acceso sobre los campos privados de la superclase. Para acceder a ellos tendrá que hacer uso de métodos de acceso o modificación. Una subclase puede invocar a cualquier método público de su superclase como si fuese propio.

Polimorfismo

- Al igual que se forma una jerarquía de clases, el hecho de que las clases definan tipos hace que la herencia dé lugar a una jerarquía de tipos. El tipo que se define mediante una subclase se dice que es un **subtipo del tipo definido en su superclase**.

- Los supertipos pueden usarse para definir operaciones que admitan objetos de distintos subtipos. Por ejemplo, podemos crear una colección que admita objetos de distintos subtipos.

- Al uso de variables de subtipos en lugares donde se espera (o se admite) un objeto de un supertipo se le denomina “sustitución”. Los lenguajes orientados a objetos trabajan con el principio de sustitución: los tipos hijos pueden sustituir a los tipos padres. Sin embargo, la operación contraria no es posible: un tipo padre no puede ocupar el lugar de un tipo hijo.

Polimorfismo

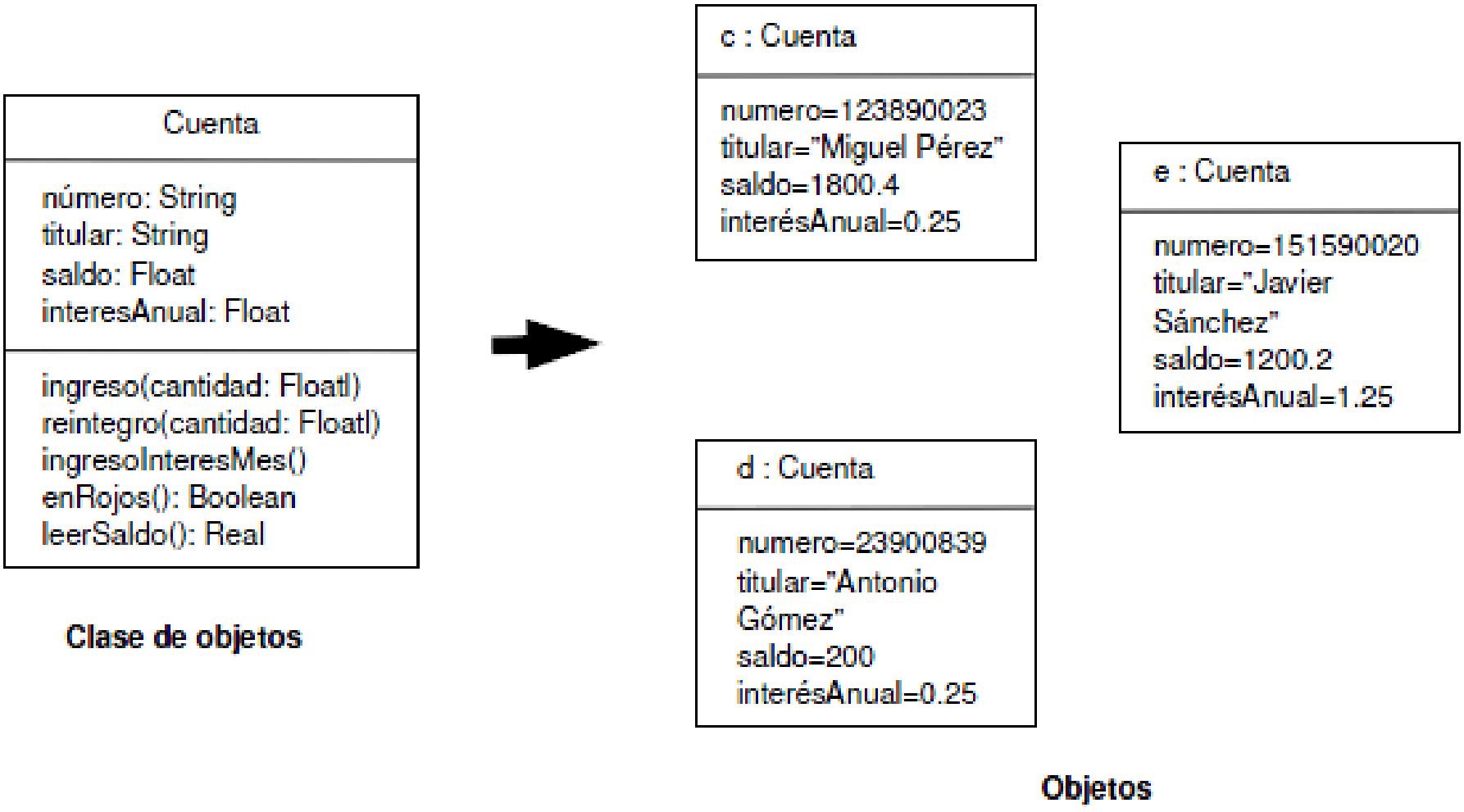
- Una variable que apunta a un objeto de un supertipo puede contener objetos de ese supertipo (si es que es coherente que existan) o de cualquier subtipo en escalas dependientes dentro de la jerarquía de tipos. Así resultarían válidas declaraciones como estas:
  - Persona p1 = new Persona();
  - Persona p1 = new Profesor();
  - Persona p3 = new ProfesorInterino();
- Esto sería erróneo: ProfesorInterino p1 = new Persona();.
  La persona puede ser un profesor interino o no: existe la incertidumbre de que lo sea o no lo sea. Java no puede saber si la persona es profesor interino o no, por lo que diremos que esta asignación no es válida en Java.
- También sería erróneo declarar ProfesorInterino p1 = new ProfesorTitular();. Obviamente esto no tiene sentido.

- Tipos abstractos de datos.
- Encapsulación, Herencia y Polimorfismo.
- **Objetos.**

Objetos

- Un objeto es una **instancia de una clase,** creada en tiempo de ejecución.
- Es una estructura de datos formada por tantos campos como atributos tiene la clase.
- El **estado** de un objeto viene dado por el valor de los campos.
- Los métodos permiten consultar y modificar el estado del objeto.

Instanciación



Tipos de los atributos

Tipos de datos primitivos:

- Enteros: byte, short, int, long
- Reales: float, double
- Carácter: char
- Booleano: boolean

Referencias:

- Sus valores son objetos de tipos no básicos,pertenecientes a otras clases.
- **Enumerados:** son clases que representan un conjunto finito de valores

Enumerados

```
enum EstadoCuenta{
    OPERATIVA, INMOVILIZADA, NUM_ROJOS;
    }

public class Cuenta{
    private String titular;
    private double saldo;
    private EstadoCuenta estado;
    …
}
```

Métodos y mensajes

Un **método** está compuesto por:

- Cabecera: Identificador y Parámetros
- Cuerpo: Secuencia de instrucciones

Mensaje:

- Mecanismo básico de la computación OO.
- Invocación de la aplicación de un método sobre un objeto.
- Un mensaje está formado por tres partes
  - **Objeto receptor**
  - Identificador del método a aplicar
  - Argumentos

Ejemplo método vs. mensaje

Método reintegro en la clase Cuenta:

```
public double reintegro (double cantidad) {
        if (puedoSacar(cantidad))
        saldo = saldo – cantidad;
}
```

Mensaje, aplica el método reintegro sobre un objeto cuenta:

```
cuenta.reintegro(600.0);
```

Instancia actual

Cada operación de un programa OO es relativa a cierto objeto, **la instancia actual,** en el momento de la ejecución de la operación.

- ¿A qué objeto Cuenta se refiere el texto de la rutina reintegro?

El cuerpo de una rutina se refiere a la instancia sobre la que se aplica

La instancia actual es el receptor de la llamada actual, **el objeto receptor del mensaje**

Instancia actual

Si se aplica un método y no se especifica el objeto receptor, se asume que es la instancia actual.

```
public double reintegro (double cantidad) {
        if (puedoSacar(cantidad))
        saldo = saldo – cantidad;
}
```

El objeto receptor de puedoSacar será el objeto receptor del reintegro

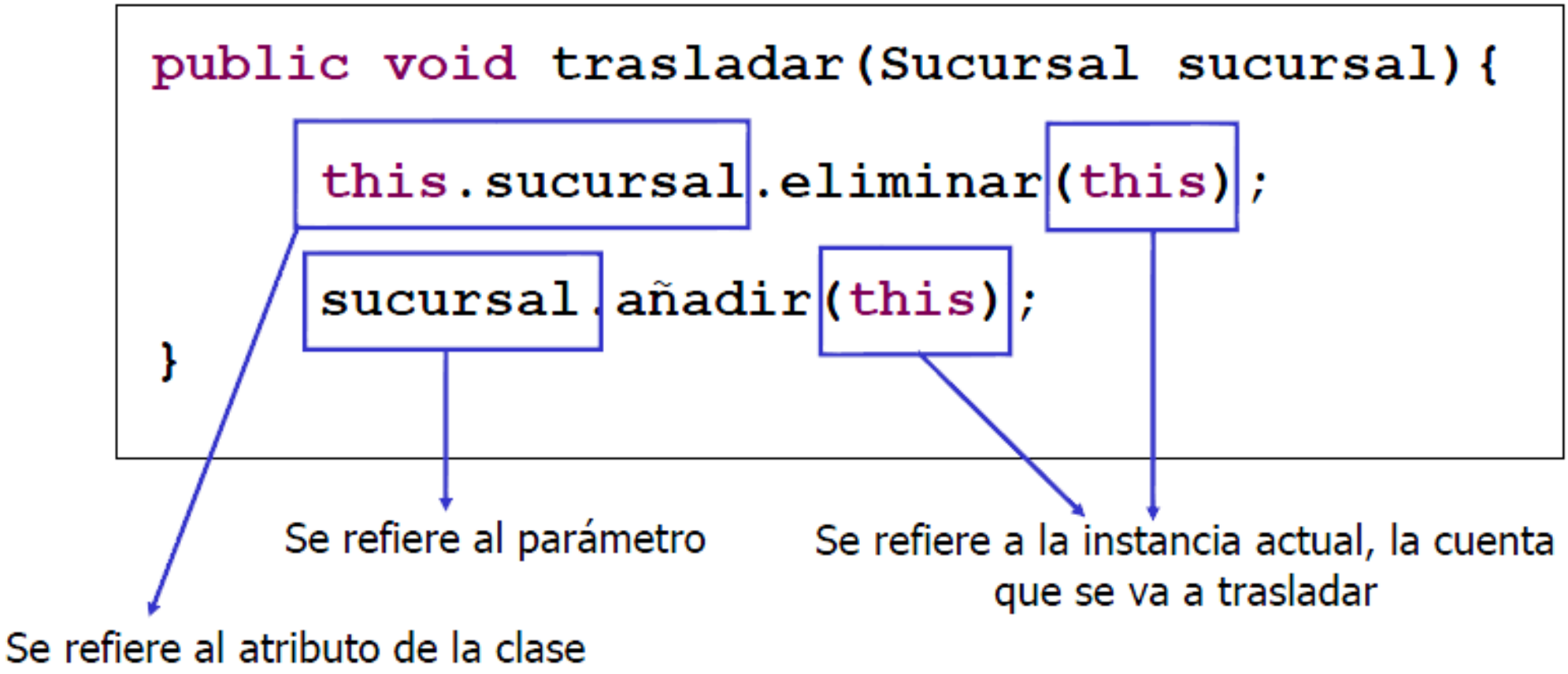
Referencia this

El lenguaje Java proporciona la palabra clave **this** que **referencia a la instancia actual**.

Utilidad:

- Distinguir los atributos de los parámetros y variables locales dentro de la implementación de un método.
- Aplicar un mensaje a otro objeto estableciendo como parámetro la referencia al objeto actual.

Referencia this



Creación de Objetos

La **declaración de una variable cuyo tipo sea** una clase **no implica la creación del objeto.**

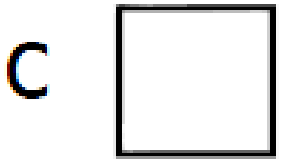
Se necesita un mecanismo explícito de creación de objetos: **new**

¿Por qué?

- Evitar cadena de creaciones antes de empezar a hacer nada útil
- Estructuras recursivas
- Los objetos se crean cuando se necesitan (referencias vacías, compartir objeto)

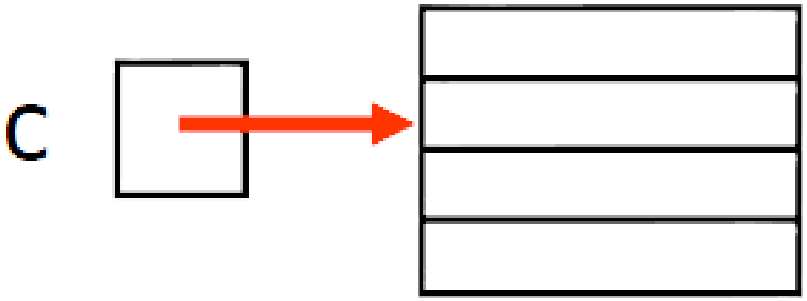
Declaración vs. Creación

```
Cuenta c;                    //declaración
```



Estado `null`

```
c = new Cuenta(...) //creación explícita
```



objeto CUENTA

- Estado "**conectado**"
- `c` contiene la referencia al objeto
- `c` almacena el oid asignado al objeto al crearse

Constructores

- Método encargado de inicializar correctamente los objetos
- Métodos con el mismo nombre de la clase pero sin valor de retorno
- No se pueden invocar una vez que el objeto se ha creado
- Permite **sobrecarga para especificar formas distintas** de inicializar los objetos
- Toda clase tiene que tener al menos un constructor
- Si no se define ningún constructor, el compilador crea uno **por defecto sin argumentos, vacío, que inicializa** los atributos a los valores por defecto.

Ejemplo de Constructores

```
public Cuenta(Persona quien){
//Utilizamos this para invocar al otro constructor
// → reutilización de código
this(quien, 100);
}

public Cuenta(Persona quien, double saldoInicial){
    titular = quien;
    saldo = saldoInicial;
    ultimasOperaciones = new double[20];
}
```

El array hay que crearlo!!!

Creación de objetos

La construcción de un objeto consta de tres etapas:

- Se reserva espacio en memoria para la estructura de datos que define la clase.
- Inicializa los campos de la instancia con los valores por defecto
  - Garantiza que cada atributo de una clase tenga un valor inicial antes de la llamada al constructor
- Se aplica sobre la instancia el constructor que se invoca

Atributos invariantes

- Java permite especificar que el valor de un atributo no podrá variar una vez construido el objeto.

- Un atributo se declara de sólo consulta anteponiendo el modificador **final** a su declaración.

- Los atributos finales sólo pueden ser inicializados en la declaración o en el constructor

Atributo final

```
public class Cuenta {

        //Los atributos se pueden inicializar
        //en el momento de la declaración
        private double saldo = 100;

        private final Persona titular;

        public Cuenta(Persona persona) {
                titular = persona;
        }

        ...
        public void setTitular(Persona persona){
                titular = persona;
        }
}
```

Error en tiempo de compilación

Atributos de clase

Representa una propiedad cuyo valor es **compartido** por todos los objetos de una misma clase

Ejemplo:

- Si tenemos una clase Cuentas con un atributo para el código de cuenta (en cada instancia).
- Es necesario una variable que almacene el último código de cuenta asignado (de manera global).

En un lenguaje imperativo se declararía una variable global.

Java es un lenguaje OO puro que no permite declaraciones fuera del ámbito de una clase.

Ejemplo de Atributos de clase

```
public class Cuenta {
        private static int ultimoCodigo = 0;

        private int codigo;
        private double saldo;
        private final Persona titular;
        private double [] ultimasOperaciones;

        public Cuenta(Persona nombre, double saldoInicial) {
                codigo = ++ultimoCodigo;
                titular = nombre;
                saldo = saldoInicial;
                ultimasOperaciones = new double[20];
        }
}
```

A los atributos de clase se tiene acceso desde cualquier método de la clase

Constantes

En Java **no hay una declaración específica para las constantes.**

Se consigue el mismo resultado definiendo un **atributo de clase y final**.

Las constantes no pueden ser modificadas.

No tiene sentido definir métodos de acceso y modificación.

El nivel de acceso es controlado por su visibilidad.

Ejemplo de Constantes

```
public class Cuenta {
        private static final int MAX_OPERACIONES = 20;
        private static final double SALDO_MINIMO = 100;

        private static int ultimoCodigo = 0;

        private int codigo;
        private double saldo;
        private final Persona titular;
        private double[] ultimasOperaciones;

        public Cuenta(Persona persona) {
                codigo = ++ultimoCodigo;
                titular = persona;
                saldo = SALDO_MINIMO;
                ultimasOperaciones = new double[MAX_OPERACIONES];
        }…
}
```

Métodos de clase

¿Cómo definimos operaciones que manejan atributos de clase?

Un método se define de clase anteponiendo el identificador **static** a su declaración

En el cuerpo del método de clase **sólo se puede acceder a los atributos de clase**

Para la **aplicación** de un método de clase no se hace uso de ningún objeto receptor, sino del **nombre de la clase**

Ejemplo de Métodos de clase

```
public class Cuenta {
    private static int ultimoCodigo = 0;
    …
    public static int getNumeroCuentas() {
        return ultimoCodigo;
    }
}
```

Cuenta.*getNumeroCuentas();*

El ciclo de vida de un objeto

- Empieza por su declaración, su instanciación y su uso en un programa Java hasta que finalmente desaparece.
- Cuando un objeto deja de ser utilizado, Java libera la memoria asignada al objeto y la reutiliza.
- El entorno de ejecución de Java decide cuándo puede reutilizar la memoria de un objeto que ha dejado de ser útil en un programa.
- El programador no debe preocuparse de liberar la memoriautilizada por los objetos. A este proceso se le conoce como recolección de basura (Garbage Collector).

Destrucción de objetos

- Java cuenta con un sistema recolector de basura (Garbage Collector) que se encarga deliberar los objetos y los espacios de memoria que ocupan cuando éstos dejan de ser utilizados en un programa (objetos no referenciados).
- En la clase Object existe un método finalize()
  - Este método se invocará justo antes de la recolección de basura
  - Por defecto no hace nada, pero se puede sobreescribir (@Override), por ejemplo para liberar recursos.
  - Su ejecuciń no está garantizada: En C++ todos los objetos se destruyen (en un programa sin errores), mientras que en Java no siempre se “recolectan”.

Modelo de ejecución OO

Para obtener un código ejecutable se deben ensamblar las clases para formar sistemas (cerrado).

Un sistema viene dado por:

- Un conjunto de clases
- La clase raíz
- El procedimiento de creación de la clase raíz.

La ejecución de un programa OO consiste en:

- Creación dinámica de objetos
- Envío de mensajes entre los objetos creados, siguiendo un patrón impredecible en tiempo de compilación

Ausencia de programa principal

Inicio de un programa

Debemos proporcionar el nombre de la clase que conduzca la aplicación.

Cuando ejecutamos un programa, el sistema localizará esta clase y ejecutará el **main** que contenga

El método **main** es un método de clase que recibe como parámetro un array de cadenas de texto que son los parámetros del programa

El método main

Definición del método main

```
public class Eco{
    public static void main(String[] args){
        for(int i=0; i < args.length; i++)
            System.out.println(args[i]+” ”);
    }
}
```

Parámetros del programa:

c:\ java Eco estamos aquí

–> SALIDA: estamos aquí