

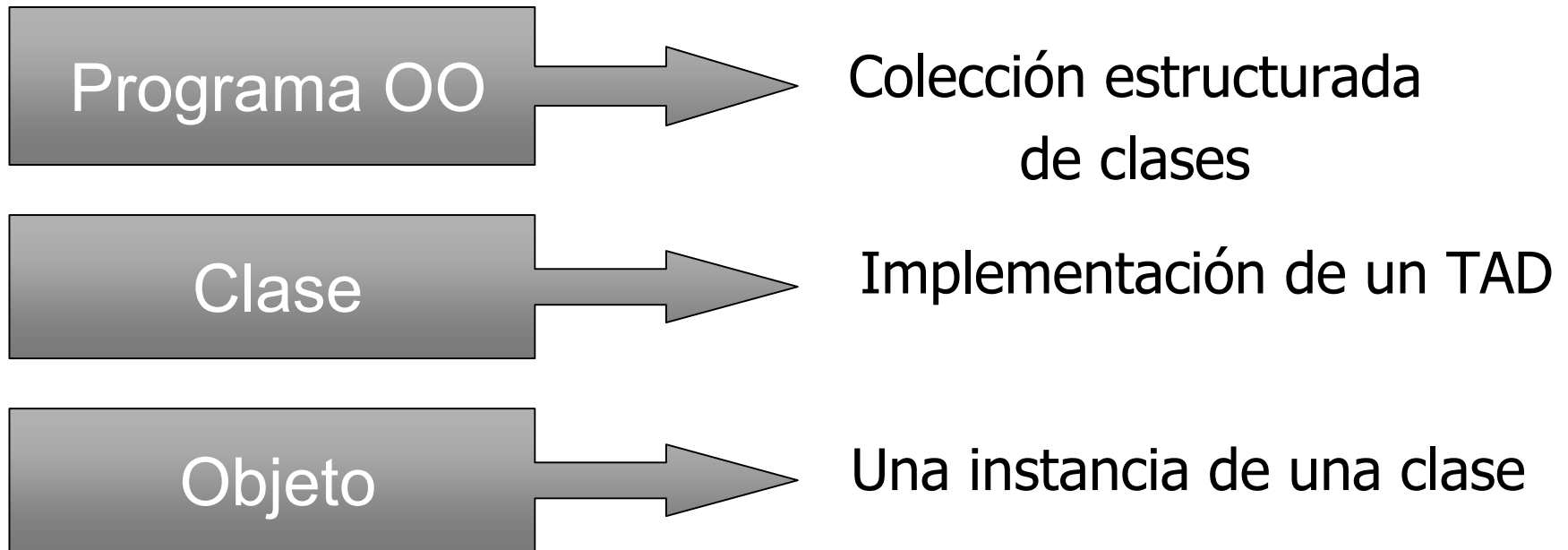
Tema 2: Clases y Objetos

Programación Orientada a Objetos
Curso 2014/2015

Contenido

- ❑ Clases.
- ❑ Objetos.
- ❑ Tipos del lenguaje.
- ❑ Relación de clientela.
- ❑ Semántica referencia.
- ❑ Métodos y mensajes.
- ❑ Instancia actual.
- ❑ Modelo de ejecución OO.
- ❑ Diseño de clases.

Introducción



Los objetos se comunican mediante **mensajes**

Clases

- **Definición:** implementación de un Tipo Abstracto de Datos (**TAD**).
- Construcción proporcionada por los lenguajes OO para la **definición de objetos que tienen la misma estructura y comportamiento.**
- Doble naturaleza: **Tipo + Módulo**
 - **Tipo:** define un nuevo tipo de datos.
 - **Módulo:** organiza el código que implementa el tipo de datos.

Componentes de una clase

- **Atributos:**

- Definen la estructura de datos que representa a los objetos.

- **Métodos:**

- Operaciones aplicables sobre los objetos.
- Encargados de acceder a los atributos.

- **Constructores:**

- Operaciones encargadas de inicializar correctamente los objetos.

- Ejemplo: en una aplicación bancaria, los objetos *cuenta* tienen en común:

- Atributos: *saldo, titular, etc.*
- Métodos: *ingreso, reintegro, etc.*

Ejemplo – Clase Cuenta

Cuenta
titular: String; saldo: double;
reintegro (valor:double); ingreso (valor:double);

Definición de la clase

Atributos

Métodos

Tiempo de ejecución

"José Martínez"	titular
1200.0	saldo

Objeto Cuenta

Clase Cuenta en Java

```
class Cuenta {  
    double saldo;  
    String titular;  
  
    void ingreso (double cantidad) {  
  
        saldo = saldo + cantidad;  
    }  
    void reintegro(double cantidad) {  
  
        if (cantidad <= saldo)  
            saldo = saldo - cantidad;  
    }  
}
```

Clases en Java

- El código se organiza en torno al concepto de clase (**unidad modular**).
- Una clase se define en un fichero de código fuente con el mismo nombre de la clase.
- Las clases permiten agrupar una estructura de datos con las rutinas que trabajan sobre ella.

Ocultación de la información

□ **Ocultación de la información:**

- A las declaraciones de una clase puede aplicarse un **modificador de visibilidad**.
- **Pública** (`public`) → visible para todo el código.
- **Privada** (`private`) → sólo visible para el código de la clase.

□ **Principio:** la estructura de datos está sujeta a más variaciones que las operaciones

- Los **atributos** se ocultan aplicando **visibilidad privada**.
- Los métodos pueden ofrecerse con distintos niveles de visibilidad.

Clase Cuenta

```
class Cuenta {  
    private double saldo;  
    private String titular;  
  
    public void ingreso (double cantidad) {  
  
        saldo = saldo + cantidad;  
    }  
  
    public void reintegro(double cantidad) {  
  
        if (cantidad <= saldo)  
            saldo = saldo - cantidad;  
    }  
}
```

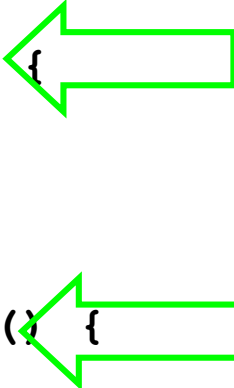
Acceso y modificación de atributos

- ❑ Los **atributos** se declaran **privados**.
- ❑ Si un atributo puede ser consultado se define un **método de acceso** (método *get*).
- ❑ Si un atributo puede ser modificado se define un **método de modificación** (método *set*).
- ❑ El programador decide el nivel de acceso que proporciona a un atributo (ninguno, *get*, *get/set*).

→ Aislamos al cliente de los cambios en la estructura de datos.

Clase Cuenta

```
class Cuenta {  
  
    private double saldo;  
    private String titular;  
  
    ...  
  
    public double getSaldo() {  
        return saldo;  
    }  
  
    public String getTitular() {  
        return titular;  
    }  
}
```



Paquetes

- ❑ Las clases se organizan en **paquetes**.
- ❑ Los paquetes permiten **agrupar código relacionado**.
- ❑ Las declaraciones para las que no se indica visibilidad tienen **visibilidad a nivel de paquete** (*visibilidad por defecto*).
- ❑ **Visibilidad de las clases:**
 - **Públicas:** son visibles desde cualquier paquete.
 - **A nivel de paquete:** sólo son visibles en el paquete al que pertenecen.

Paquetes

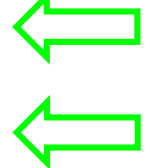
- ❑ El nombre de una clase (tipo) debe ir precedido por el paquete (ruta de paquetes) al que pertenece
→ **Nombre calificado de la clase.**
- ❑ La **pertenencia** de una clase a un **paquete** debe ser especificada antes de la declaración.
- ❑ **Anidamiento de paquetes.**
- ❑ Se utiliza la declaración **import** para poder omitir la ruta de paquetes al nombrar una clase.

Paquetes

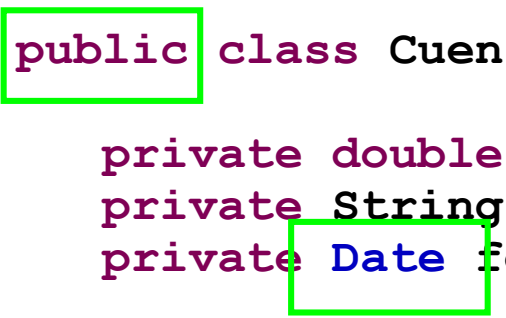
```
package banco.cuentas;
import java.util.Date;
public class Cuenta {
    private double saldo;
    private String titular;
    private Date fecha;

    public void ingreso (double cantidad) { ... }

    public void reintegro (double cantidad) { ... }
}
```



The diagram shows two green arrows pointing left. The top arrow points to the package name 'banco.cuentas' in the 'package' statement. The bottom arrow points to the class name 'Date' in the 'import' statement.



The diagram shows two green boxes. The first box highlights the 'public' keyword in the 'public class' statement. The second box highlights the 'Date' class name in the 'private Date fecha;' statement.

Objetos

- Un **objeto** es una instancia de una clase creada en tiempo de ejecución.
- Está representada por una **estructura de datos en memoria** formada por tantos **campos** como atributos tiene la clase.
- El **estado** de un objeto viene dado por el valor de sus campos.
 - ➔ Los métodos son el único modo de acceder y modificar el estado del objeto.
 - ➔ El estado de un objeto es inicializado por su **constructor**

Declaración y construcción

- ❑ La **declaración** de una variable cuyo tipo sea una clase **no implica la creación de un objeto**.
- ❑ Los objetos se construyen explícitamente con el operador **new** que invoca a un **constructor**.

```
// La variable "cuenta" no está inicializada  
Cuenta cuenta;  
  
// El objeto se crea llamando a un constructor  
cuenta = new Cuenta();
```

Constructores en Java

- ❑ **Un constructor se encarga de la correcta inicialización de los objetos antes de su uso.**
- ❑ Método con el mismo nombre de la clase pero sin valor de retorno.
- ❑ Todas las clases deben tener **al menos un constructor**.
- ❑ Si el programador no *declara* ningún constructor, el compilador incluye un **constructor por defecto** (constructor vacío sin parámetros).

Constructores en Java

- La construcción de un objeto consta de **tres etapas**:
 - Se reserva en memoria **espacio para la estructura de datos** que define la clase.
 - Se realiza la **inicialización de los campos** con los valores por defecto asociados a su tipo de datos.
 - Se realiza la **llamada al constructor** que finaliza la inicialización.

Inicialización de los campos

- Inicialización de los campos por defecto:
 - **Tipos numéricos:** se inicializan a 0.
 - **Carácter:** al carácter 0 ('   ')
 - **Booleano:** false.
 - **Objetos:** valor nulo (**null**).

- Podemos establecer el **valor de inicialización en la declaración de un atributo.**

Clase Cuenta

```
public class Cuenta {  
    private double saldo = 100;  
    private String titular;  
  
    public Cuenta(String nombre) {  
        titular = nombre;  
    }  
  
    public void ingreso (double cantidad) { ... }  
  
    public void reintegro(double cantidad) { ... }  
}
```

Atributos finales

- ❑ 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 **deben ser inicializados** en la declaración o en un constructor.

Clase Cuenta

```
public class Cuenta {  
    private double saldo = 100;  
    private final String titular;  
    public Cuenta(String nombre) {  
        titular = nombre;  
    }  
    ...  
}
```

Reutilización constructores

- Una clase puede tener más de un constructor.
- Habitualmente los constructores utilizan otros constructores de la misma clase (**reutilización**).
- Un constructor puede invocar a otro utilizando **this** (*parámetros*)

Clase Cuenta

```
public class Cuenta {  
  
    ...  
    public Cuenta(String persona, double saldoInicial) {  
  
        saldo = saldoInicial;  
        titular = persona;  
    }  
  
    public Cuenta(String persona) {  
        this(persona, 100.0);  
    }  
    ...  
}
```

Destrucción de objetos

- ❑ En Java **no se destruyen los objetos explícitamente** (no hay operador *delete*).
- ❑ Existe un mecanismo que elimina los objetos que no están en uso (que no son referenciados)
→ **Garbage collector**

Tipos del lenguaje

- El lenguaje Java define **dos categorías** de tipos de datos:
 - **Tipos primitivos.**
 - **Objetos** (definidos por clases).
- Los **tipos primitivos** corresponden a los tipos de datos básicos:
 - Enteros: *byte, short, int, long*
 - Reales: *float, double*
 - Carácter: *char*
 - Booleano: *boolean*
- **String** es una clase, no un tipo primitivo.

Tipos del lenguaje - Enumerados

- Los enumerados son tipos y sus valores son objetos que se definen con la construcción **enum**:

```
public enum EstadoCuenta {  
    OPERATIVA, INMOVILIZADA, NUMEROS_ROJOS;  
}
```

```
public class Cuenta {  
    ...  
    private EstadoCuenta estado;  
  
    public Cuenta(Persona persona) {  
        estado = EstadoCuenta.OPERATIVA;  
        ...  
    }  
}
```

Tipos del lenguaje – Arrays

- ❑ Los arrays son objetos.
- ❑ Tal como sucede con las clases, los arrays no se construyen en su declaración.
→ Es necesario construirlos utilizando el operador **new**.

```
public class Cuenta {  
  
    private double saldo = 100;  
    private final String titular;  
    private double[] ultimasOperaciones;  
  
    public Cuenta(String nombre) {  
  
        titular = nombre;  
        ultimasOperaciones = new double[20];  
    }  
    ...  
}
```

Tipos del lenguaje – Arrays

- Cada elemento del array se inicializa con el valor por defecto asociado a su tipo de datos:
 - Objetos a **null**, enteros a 0, etc.
 - Por tanto, si el tipo del array es una clase, las casillas del array no almacenan ningún objeto.
- Los arrays se indexan a partir del índice 0.
- Se pueden construir arrays de **varias dimensiones**:

```
Cuenta[][] cuentas = new Cuenta[2][3];  
cuentas[0][0] = new Cuenta("Juan");  
cuentas[0][1] = new Cuenta("Pedro");  
...
```

Relación de clientela

- Cuando una clase A declara un atributo cuyo tipo es otra clase B, decimos que la clase A es cliente de B.

- Ejemplo:
 - Definimos la clase **Persona** con las propiedades *nombre* y *dni*.
 - Declaramos el atributo *titular* de tipo **Persona**.

Relación de clientela

```
public class Cuenta {  
  
    ...  
    private final Persona titular;  
    private double[] ultimasOperaciones;  
  
    public Cuenta(Persona persona) {  
  
        titular = persona;  
        ultimasOperaciones = new double[MAX_OPERACIONES];  
    }  
  
    public Persona getTitular() {  
  
        return titular;  
    }  
    ...  
}
```


Semántica referencia

- Los objetos tienen un identificador que diferencia unos de otros (**identificador de objeto**, *oid*)
- En la **asignación de un objeto a una variable**, no se asigna la estructura de datos del objeto, sino el **identificador del objeto**.

→ **En Java los objetos se manejan por referencia.**

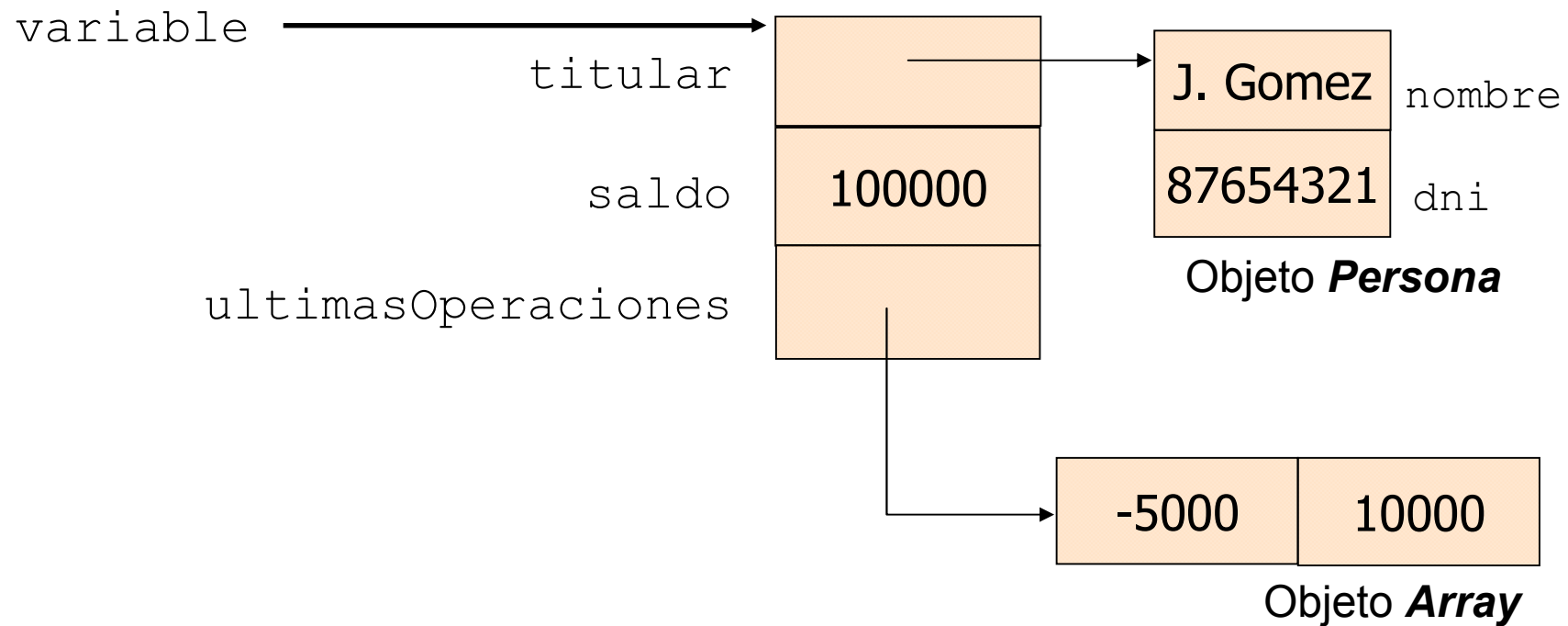
- Una referencia puede tener dos **estados**:
 - *No conectada*: no referencia a ningún objeto y su valor es **null**.
 - *Conectada*: contiene la referencia (*oid*) a un objeto.

Semántica referencia

- Cuando se declara una variable la referencia no está conectada.
- Al crear el objeto se le asigna un identificador, que es almacenado en la variable

```
// La variable "cuenta" no está conectada, es decir  
// no referencia a ningún objeto (valor null)  
Cuenta cuenta;  
  
// La variable contiene la referencia al nuevo objeto  
cuenta = new Cuenta();
```

Semántica referencia



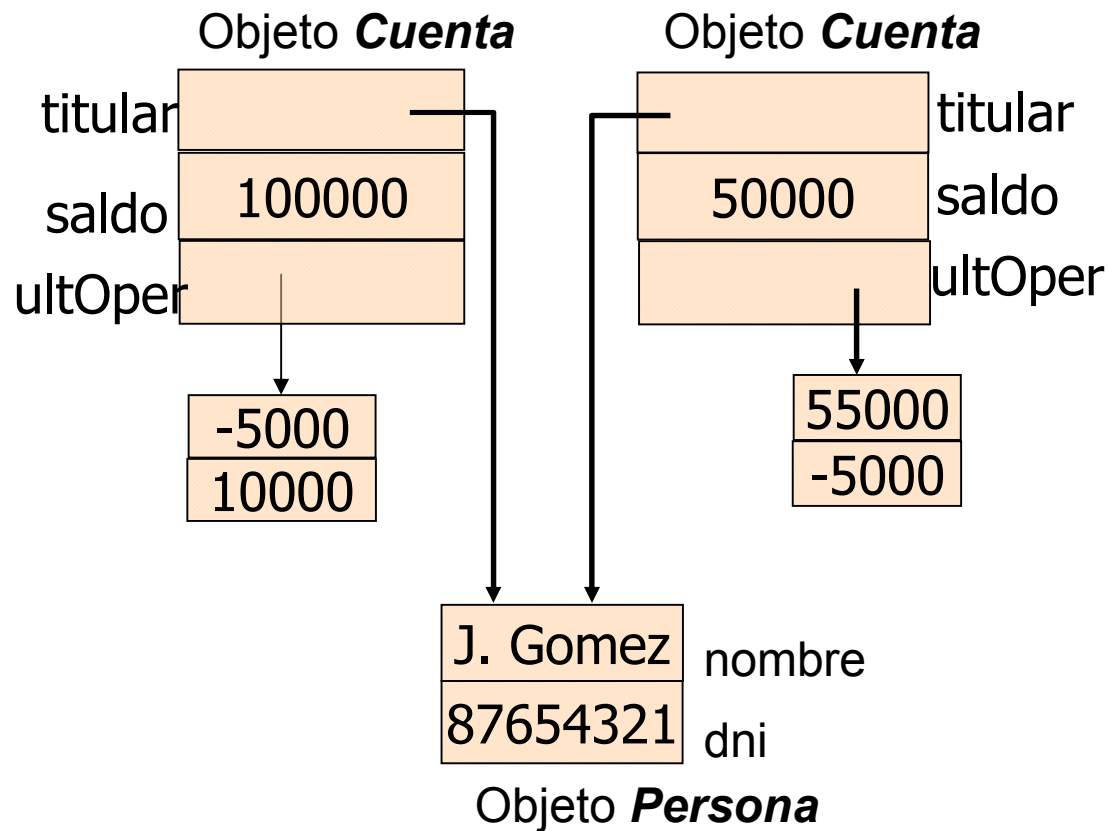
Semántica referencia

- El manejo de los objetos por referencia tiene **ventajas**:
 - Compartición de objetos → **integridad referencial**.
 - **Estructuras recursivas**: objetos que se referencian a sí mismos.
 - Resulta más **eficiente** para el manejo de objetos complejos.
 - Los objetos se crean cuando se necesitan y no en su declaración.

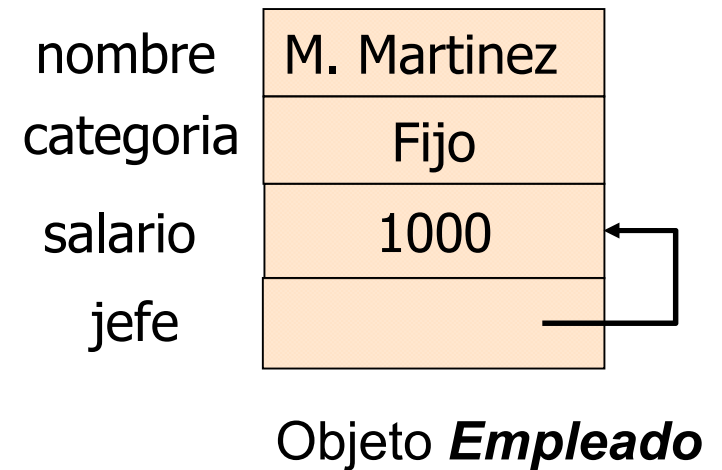
- Pero tiene un **inconveniente**: **aliasing**.

Semántica Referencia

a) Compartición

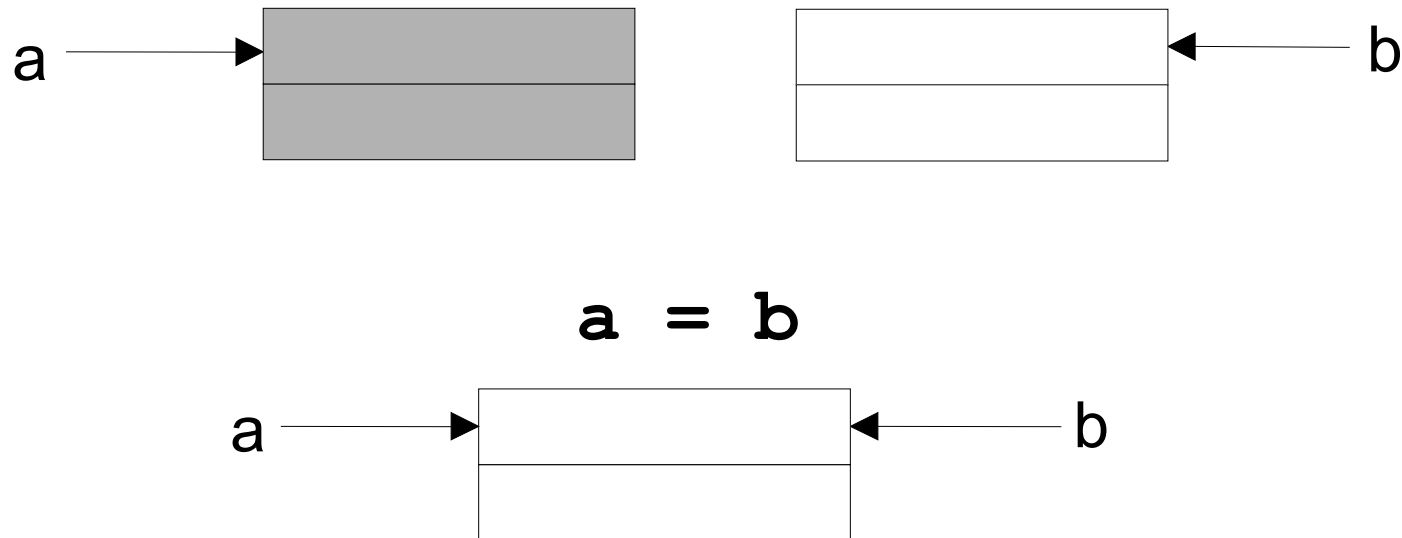


b) Autorreferencias



Asignación vs. copia

- **Asignación** de referencias (operador =):
 - No implica la copia de los objetos, sino los *oid*.
 - Se produce **aliasing**



- ¿Cómo podemos **copiar los objetos**? Método **clone** (se estudia en el próximo tema)

Aliasing - problemas

```
Cuenta cuenta1;  
Cuenta cuenta2;  
...  
  
double saldoCuenta1 = cuenta1.getSaldo();  
cuenta2 = cuenta1;  
cuenta2.reintegro(1000.0);  
  
// cuenta1.getSaldo() != saldoCuenta1 !!
```

Aliasing - Ocultación de la información

- Hay que prestar atención a los **métodos de acceso**, ya que si un atributo es una referencia (objeto), al devolver la referencia **se compromete la integridad del objeto**.
- Ejemplo: método de consulta de las últimas operaciones de la cuenta.

```
public double[] getUltimasOperaciones()  
{  
    return ultimasOperaciones;  
}
```

- ➔ Quien consulte las últimas operaciones podría modificar el array ...

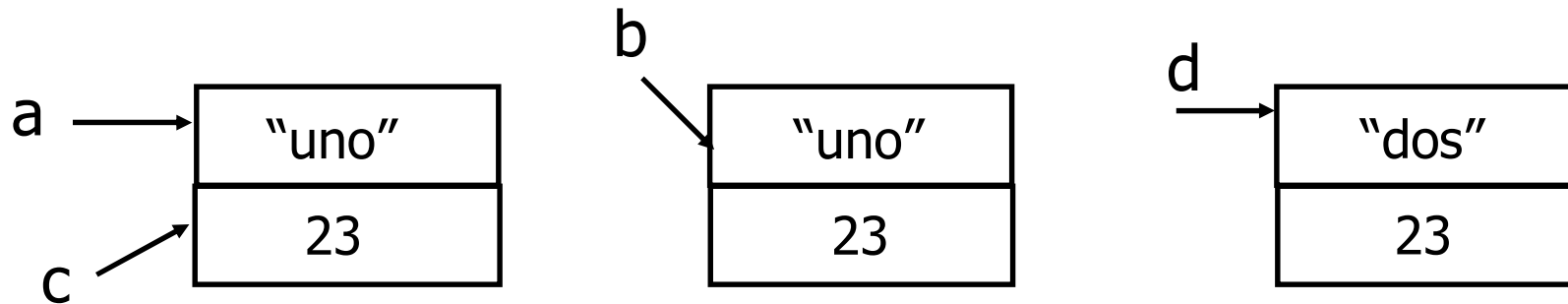
Aliasing - Ocultación de la información

- Los arrays son objetos: se manejan por referencia.
- El cliente que recibe la referencia puede modificar el array afectando al objeto cuenta.

- **Recomendación:** cuando se devuelve la referencia a un objeto en un **método público** hay que valorar
 - Si el objeto no puede ser modificado (**objeto inmutable**). Por ejemplo, los objetos *String* no son modificables.
 - En caso de ser modificable, devolver una copia del objeto para la consulta.
 - **Siempre hay que valorar el contexto de la clase.**

Identidad vs. igualdad

- ❑ **Identidad:** dos referencias son idénticas si apuntan al mismo objeto (operador `==`).



- ❑ Identidad entre referencias:
 - `a == c; // True`
 - `a == b; // False`
- ❑ ¿Cómo podemos comprobar la **igualdad**?
Método **equals** (se estudia en el próximo tema)

Métodos y Mensajes

- Un **método** es una operación asociada a una clase de objetos.
- Está compuesto por:
 - **Cabecera**: identificador y parámetros (signatura).
 - **Cuerpo**: secuencia de instrucciones.
- **Mensaje**:
 - Un mensaje consiste en la aplicación de un método sobre un objeto.
 - Mecanismo básico de la computación OO.
 - Está formado por el **objeto receptor**, el **identificador del método** y los **argumentos**.

Métodos y Mensajes

- Declaración del **método** `reintegro` de la clase `Cuenta`:

```
public void reintegro(double cantidad) {  
    if (cantidad <= saldo)  
        saldo = saldo - cantidad;  
}
```

- Aplicación del **mensaje** `reintegro` a un objeto `cuenta`:

```
cuenta.reintegro(6000.0);
```

Métodos – Estructuras de control

- Sentencias condicionales: **if**
- Bucles: **while**, **for**
 - Bucle “for each”

```
public double getGastos() {  
    double total = 0;  
    for (double operacion : ultimasOperaciones) {  
        if (operacion < 0) total += operacion;  
    }  
    return total;  
}
```

- Selecciones múltiples: **switch (opcion)**
 - Hasta la versión 6 la opción sólo podía ser de tipo entero o un enumerado.
 - A partir de la **versión 7** también es posible utilizar cadenas de texto (tipo `String`).

Sobrecarga de métodos

- Posibilidad de definir varios métodos con el mismo identificador, pero con distinta **lista** de tipos de parámetros (se ignora el tipo de retorno).

```
// Pago de una compra en una vez
public void cobrar(Compra ticket){
    reintegro(ticket.getTotal());
}

// Pago a plazos
public boolean cobrar(Compra ticket, boolean aplazado){
    ...
}
```

Paso de parámetros

- El paso de parámetros define el modo en el que se asignan los parámetros reales de una rutina a los parámetros formales (identificadores de los parámetros en la rutina).

```
public void reintegro(double cantidad) {  
    ...  
}
```

Parámetro
formal

```
double v = 3000.0;  
cuenta.reintegro(6000.0);  
cuenta.reintegro(v);
```

Parámetro
real

Paso de parámetros

- En Java se opta por el modo más simple de paso de parámetros: **paso por valor**
 - Se asigna (operador =) el parámetro real en la variable que representa al parámetro formal.

- Implica que si se utilizan variables para el paso de parámetros (variable v en el ejemplo), la **variable no se ve alterada por el método**.

Paso de parámetros

- El paso por valor protege a las variables de ser modificadas, pero **no protege el estado de los objetos**.

```
public void transferencia(Cuenta emisor, Cuenta receptor,  
                          double cantidad) {  
  
    emisor.reintegro(cantidad);  
    receptor.ingreso(cantidad);  
    emisor = null;  
}  
  
Cuenta cuenta1 = new Cuenta(titular1);  
Cuenta cuenta2 = new Cuenta(titular2);  
...  
banco.transferencia(cuenta1, cuenta2, 3000.0);
```

Cambia el estado de los dos objetos

No afectaría a la variable "cuenta1"

Argumentos de tamaño variable

- ❑ Un método o constructor puede definir un argumento de tamaño variable.
- ❑ Dentro del método el parámetro se utiliza como un **array**.
- ❑ Sólo puede haber un argumento de este tipo y debe estar al final de la lista de argumentos.
 - Ejemplo: método que permite ingresar varias cantidades.

```
public void ingreso(double... cantidades) {  
    for (double cantidad : cantidades) {  
        // ...  
    }  
}  
  
cuenta.ingreso(30, 40, 20);  
cuenta.ingreso(); // permite no establecer el parámetro
```

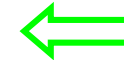
Atributos de clase

- ¿Cómo representamos que el valor de una propiedad sea compartido por todos los objetos de una misma clase?
- Ejemplo:
 - Añadimos a las cuentas un atributo para el código.
 - Es necesario una variable que almacene el último código de cuenta asignado.
- 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.

Atributos de clase

- ❑ Los atributos de clase son **compartidos por todos los objetos de la clase**.
- ❑ Se definen anteponiendo el modificador **static** a la declaración del atributo.



```
public class Cuenta {  
    private static int ultimoCodigo = 0;  
  
    private int codigo;  
    private double saldo = 100;  
    private final String titular;  
  
    public Cuenta(String nombre) {  
        codigo = ++ultimoCodigo;  
        titular = nombre;  
    }  
    ...  
}
```



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 se consideran atributos
➔ No tiene sentido definir métodos de acceso y modificación.
- ❑ El nivel de acceso es controlado por su visibilidad.

Constantes

```
public class Cuenta {  
    private static final int MAX_OPERACIONES = 20;   
    private static int ultimoCodigo = 0;  
  
    private int codigo;  
    private double saldo = 100;  
    private final String titular;  
    private double[] ultimasOperaciones;  
  
    public Cuenta(String nombre) {  
        codigo = ++ultimoCodigo;  
        titular = nombre;  
        ultimasOperaciones = new double[MAX_OPERACIONES];   
    }...  
}
```

Métodos de clase

- ❑ ¿Podemos definir operaciones que NO se apliquen sobre objetos?
- ❑ Ejemplo: consultar el número de cuentas creadas.
- ❑ Un método se define de clase anteponiendo el modificador **static** a su declaración.
- ❑ El cuerpo del método **sólo puede acceder a los parámetros y 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**.

Métodos de clase

```
public class Cuenta {  
    private static int ultimoCodigo = 0;  
    ...  
    public Cuenta(Persona persona) {  
        codigo = ++ultimoCodigo;  
        ...  
    }  
    public static int getNumeroCuentas() {  
        return ultimoCodigo;  
    }  
}
```

```
Cuenta.getNumeroCuentas();
```


Importación declaraciones de clase

- Se puede utilizar cuando se necesita acceso frecuente a declaraciones `static` de una clase.
 - Por ejemplo, funciones matemáticas (clase `Math`)
 - `double r = Math.cos(Math.PI * theta);`
- Se importa un elemento estático de una clase para ser utilizado sin calificación:
 - `import static java.lang.Math.PI;`
 - `import static java.lang.Math.*;`
- La expresión anterior utilizando la importación de clase sería:
 - `double r = cos(PI * theta);`

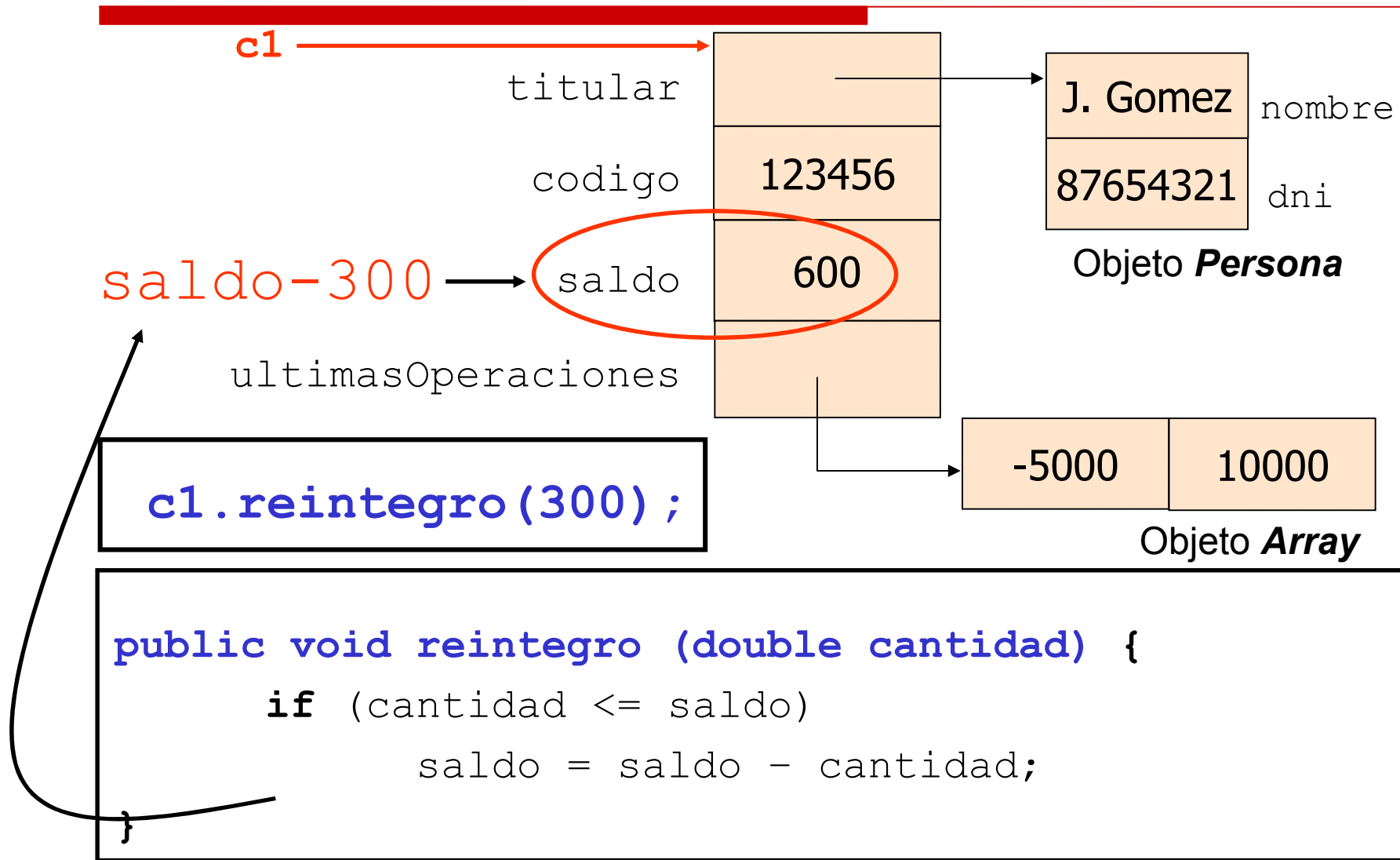
Instancia actual

- ¿A qué objeto **Cuenta** se refiere el código del método **reintegro**?

```
public void reintegro(double cantidad) {  
    if (cantidad <= saldo)  
        saldo = saldo - cantidad;  
}
```

- El cuerpo de un método trabaja sobre la instancia (objeto) sobre la que se aplica el mensaje (**instancia actual**).

Instancia actual



Instancia actual

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

```
public void cobrar(Compra ticket){  
    reintegro(ticket.getTotal());  
}
```

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.

```
public void trasladar(Oficina oficina) {  
    this.oficina.eliminarCuenta(this);  
    oficina.añadirCuenta(this);  
    this.oficina = oficina;  
}
```

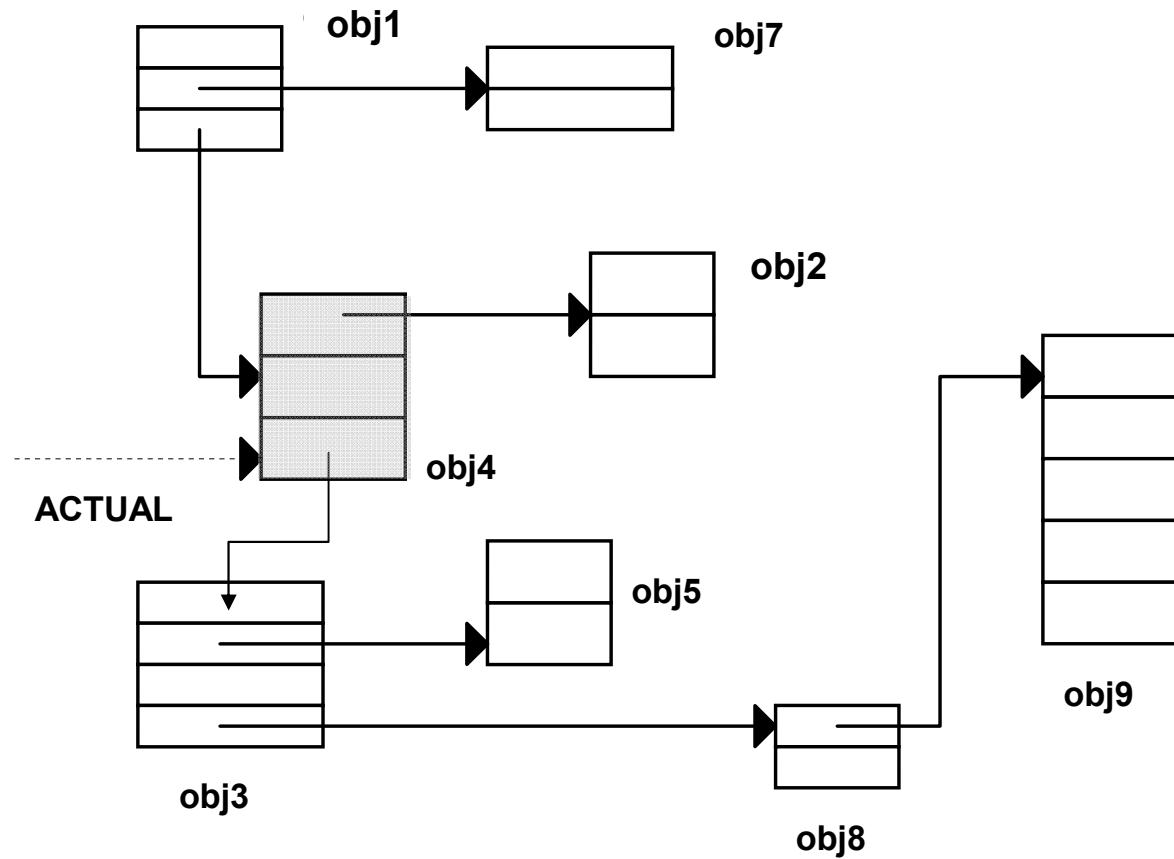
Modelo de ejecución OO

- Una **aplicación OO** se organiza como un **conjunto de clases relacionadas** entre sí (clientela, herencia).
- Una de las clases representa la **raíz de la aplicación** y contiene un método encargado de poner en marcha la aplicación.
- La **ejecución de un programa OO** consiste en:
 - Creación dinámica de objetos.
 - Envío de mensajes entre objetos.
 - No hay *programa principal*, aunque sí un método encargado de arrancar la aplicación.

Modelo de ejecución OO

- En **tiempo de ejecución**, el flujo de ejecución siempre se encuentra en uno de los siguientes estados:
 - **Aplicando un método sobre algún objeto (instancia actual)**
 - Ejecutando alguna instrucción imperativa (asignación, creación, condicional, recorrido, etc.).
- **Nota:** en una aplicación concurrente puede haber varios flujos de ejecución

Modelo de ejecución OO



Método main de Java

- ❑ En Java, la rutina de código encargada de arrancar la aplicación se denomina **método main**.
- ❑ Se define como un **método de clase** (*static*) que tiene como parámetro la lista de argumentos del programa.
- ❑ **La ejecución de una aplicación Java debe comenzar por un método main.**

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

Características avanzadas

- Un objeto puede ser notificado antes de ser eliminado por el Garbage Collector implementando el método **finalize** ().
 - Sólo interesa cuando hace uso de recursos que quedan fuera del proceso que ejecuta la aplicación: conexiones de bases de datos, etc.
- Los **parámetros formales** de un método podrían declararse con el modificador **final** indicando que no pueden ser modificados en el cuerpo del método.
- Las variables locales también se pueden declarar **final**.

Diseño de Clases

- **Ocultación de la información:**
 - Los atributos deben ser privados.

- No todos los atributos necesitan **métodos de acceso/modificación**.
 - Para la modificación de un atributo no siempre es adecuado un método *set*. Por ejemplo, el saldo de una cuenta se modifica con las operaciones de ingreso y reintegro.

- No todas las **propiedades** de un objeto deben representarse como atributos. Algunas propiedades son **calculadas**.

Diseño de Clases

- Evitar clases con muchos **atributos de tipo primitivo**:
 - Suele ser síntoma de que es necesario definir otra clase que incluya parte de esos atributos.
 - Ejemplo: datos del cliente de una *Cuenta* definidos en la clase *Persona*.

- Patrón de diseño **Experto en información**:
 - Asignar una funcionalidad (método) a la clase que tiene la información necesaria para llevarla a cabo.

- Los **identificadores** de clases y métodos deben ser **significativos**.

Seminario 1

- El seminario virtual 1 complementa los contenidos desarrollados en el tema.
- Incluye contenidos que no se desarrollan en teoría:
 - Documentación del código **JavaDoc**.
 - **Convención de nombrado** en Java.