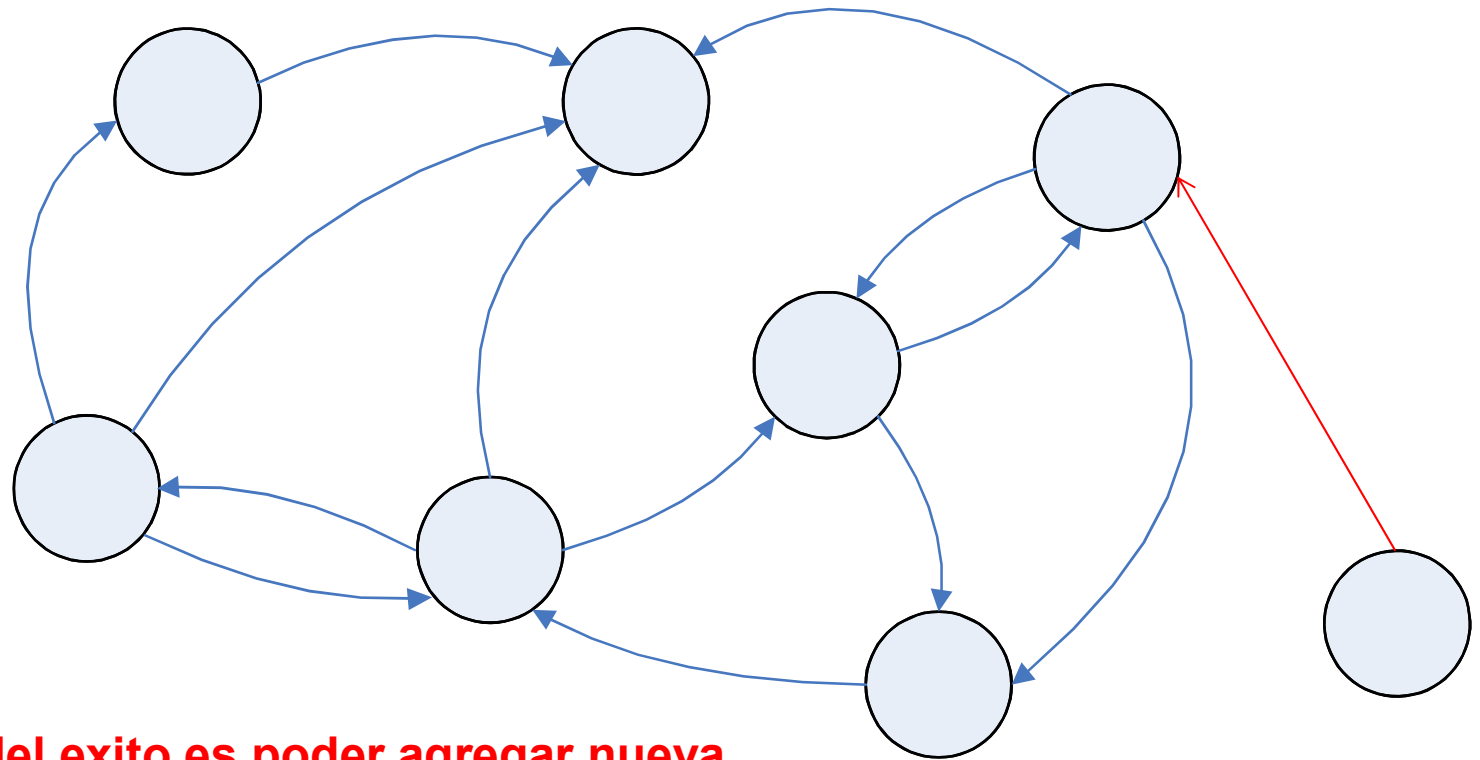


# Programa o Sistema Orientado a Objetos

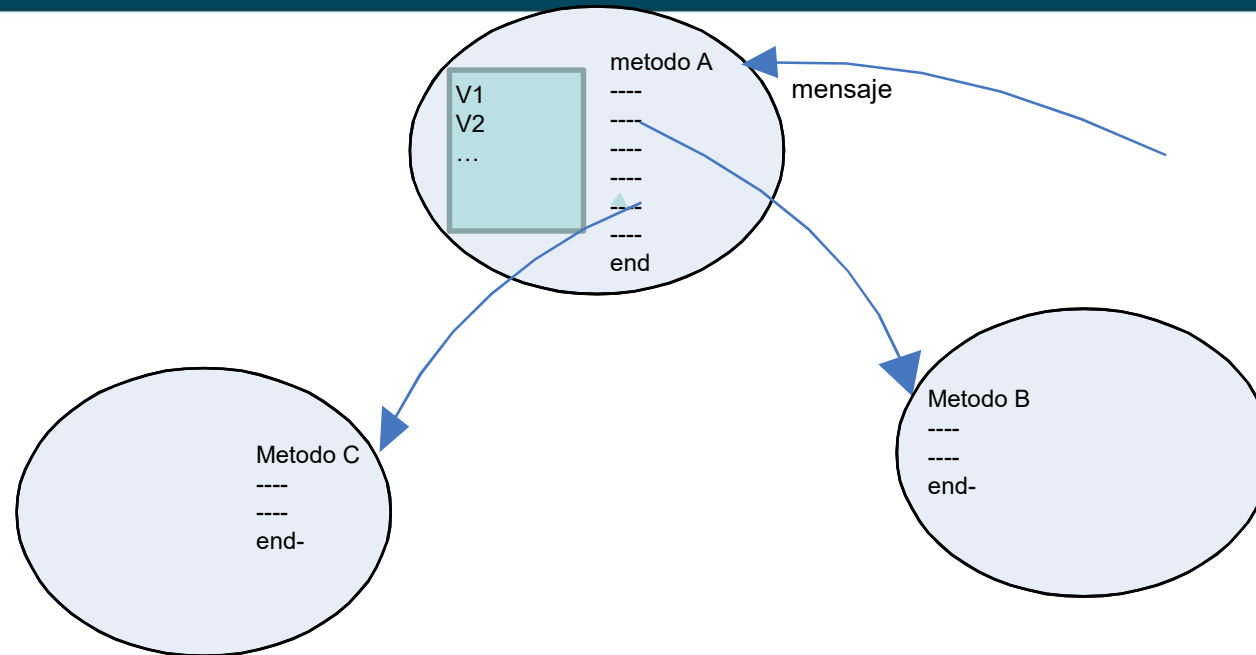
- ¿Como es un software construido con objetos?

Un conjunto de **objetos** que **colaboran** enviándose **mensajes**. **Todo computo ocurre “dentro” de los objetos**



**La clave del éxito es poder agregar nueva funcionalidad (no prevista originalmente), reemplazar objetos o modificar objetos y que el sistema “no se entere”, ni se rompa.**

## Más en detalle



- Los sistemas están compuestos (solamente) por un conjunto de **objetos** que **colaboran** para llevar a cabo sus responsabilidades.
- Los objetos son **responsables** de:
  - conocer sus propiedades,
  - conocer otros objetos (con los que colaboran) y
  - llevar a cabo ciertas acciones.

## Aspectos de interés en esta definición

- No hay un objeto “main”
- Algoritmos y datos ya no se piensan por separado
- Cuando codificamos, describimos clases
- Cuando se ejecuta el programa lo que tenemos son objetos que cooperan y que se crean dinámicamente durante la ejecución del programa
- Una jerarquía de clases no indica lo mismo que la jerarquía top-down

## Impacto en como “pensamos” el software

- La estructura general cambia: en vez de una jerarquía: Main/procedures/sub-procedures tenemos una red de “cosas” que se comunican
- Pensamos en que “cosas” hay en nuestro software (los objetos) y como se comunican entre sí.
- Hay un “shift” mental crítico en forma en la cuál pensamos el software como objetos
- Mientras que la estructura sintáctica es “lineal” el programa en ejecución no lo es

## ¿Qué es un objeto?

- Es una *abstracción* de una *entidad* del *dominio del problema*. Ejemplos: Persona, Producto, Cuenta Bancaria, Auto, Plan de Estudios,....
- *Puede representar también conceptos del espacio de la solución (estructuras de datos, tipos “básicos”, archivos, ventanas, conexiones, iconos, adaptadores, ...)*

# Características de los Objetos

- Un objeto tiene:

- ***Identidad.***

- para distinguir un objeto de otro (independiente de sus propiedades)

- ***Conocimiento.***

- En base a sus relaciones con otros objetos y su estado interno

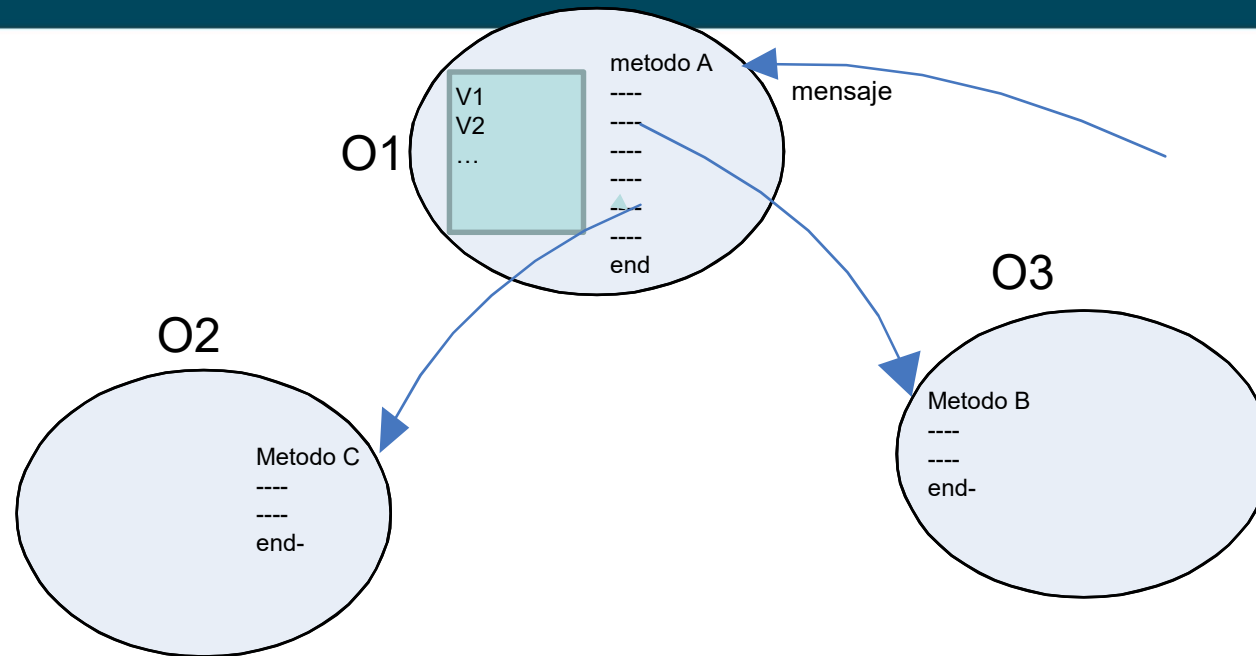
- ***Comportamiento.***

- Conjunto de mensajes que un objeto sabe responder

## El estado interno

- El estado interno de un objeto determina su *conocimiento*.
- El estado interno se mantiene en las *variables de instancia (v.i.)* del objeto.
- Es **privado** del objeto. Ningún otro objeto puede accederlo. (¿Cual es el impacto de esto?)
- El estado interno está dado por:
  - Propiedades básicas (intrínsecas) del objeto.
  - Otros objetos con los cuales colabora para llevar a cabo sus responsabilidades.

# Variables de instancia



- En general las variables son REFERENCIAS (punteros) a otros objetos con los cuales el objeto colabora.
- Algunas pueden ser atributos básicos
- En el gráfico O1 puede mandarle mensajes a O2 y O3 porque “los conoce”, o sea hay una variable en O1 que APUNTA a O2 y otra a O3 (o la misma variable que cambia de valor en diferentes momentos)



## Ejemplos

- Objeto Alumno:

v.i: nombre, dni, **fecha\_nac**,  
**carrera, legajo**

-Las 2 primeras pueden ser tipos “básicos”: string, numero (dependiendo del lenguaje)

-Las otras 3 son referencias a objetos de la Clase Fecha, Carrera y Legajo

- Objeto Carrera:

v.i: nombre, año, **materias**

Materias es una colección (array?) de objetos Materia

- Objeto Materia:

v.i: nombre, año, semestre, **correlativas**

# Comportamiento

- Un objeto se define en términos de su comportamiento.
- El comportamiento indica qué sabe hacer el objeto. Cuáles son sus *responsabilidades*.
- Se especifica a través del conjunto de *mensajes* que el objeto sabe responder: *protocolo*.
- Ejemplo:

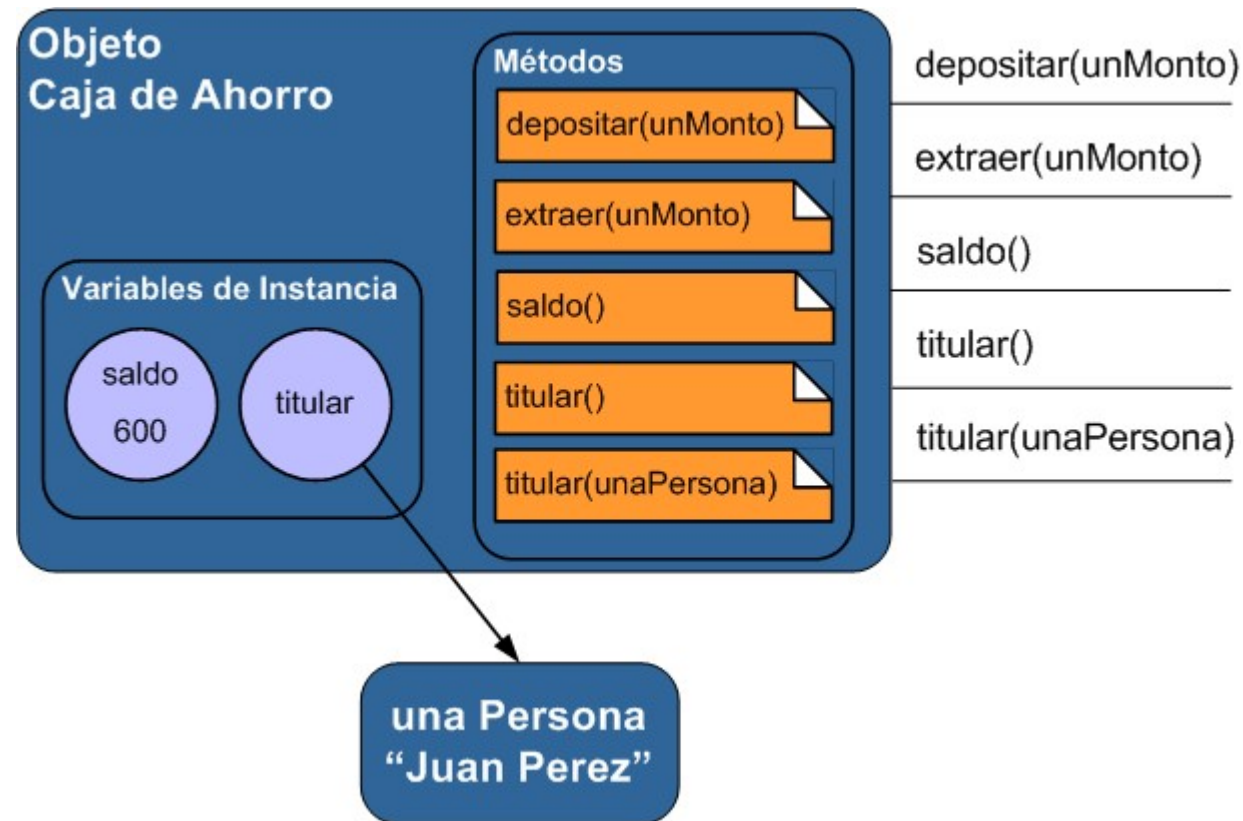


## Comportamiento - implementación

- La **realización** de cada mensaje (es decir, la manera en que un objeto responde a un mensaje) se especifica a través de un ***método***.
- Cuando un **objeto** recibe un **mensaje** responde activando el ***método*** asociado.
- El que envía el mensaje ***delega*** en el receptor la manera de resolverlo, que es ***privada*** del objeto.



# Ejemplo Caja de Ahorro



Observese la variable "titular" apuntando a un objeto Persona

# Encapsulamiento

*“Es la cualidad de los objetos de ocultar los detalles de implementación y su estado interno del mundo exterior”*

- Características:
  - Esconde detalles de implementación.
  - Protege el estado interno de los objetos.
  - Un objeto sólo muestra su “cara visible” por medio de su protocolo.
  - Los métodos y su estado quedan escondidos para cualquier otro objeto. Es el objeto quien decide *qué* se publica.
  - Reduce el acoplamiento, facilita modularidad y reutilización

## Envío de un mensaje

- Para poder enviarle un mensaje a un objeto, **hay que conocerlo**.
- Al enviarle un mensaje a un objeto, éste responde activando el método asociado a ese mensaje (siempre y cuando exista).
- Como resultado del envío de un mensaje puede retornarse un objeto.

# Especificación de un Mensaje

- ¿Cómo se especifica un mensaje?
  - **Nombre:** correspondiente al protocolo del objeto receptor.
  - **Parámetros:** información necesaria para resolver el mensaje.
- Cada lenguaje de programación propone una sintaxis particular para indicar el envío de un mensaje.

*Ejemplo: cuenta.depositar(cantidad)*

*figura.dibujar()*

*figuraGrande.rotar(45)*

- cuenta, figura, figuraGrande son variables que apuntan a un objeto que entiende el mensaje correspondiente

# Métodos

- ¿Qué es un método?
  - Es la contraparte funcional del mensaje.
  - Expresa la forma de llevar a cabo la semántica propia de un mensaje particular (el *cómo*).
- Un método puede realizar básicamente 3 cosas:
  - Modificar el estado interno del objeto.
  - Colaborar con otros objetos (enviándoles mensajes).
  - Retornar y terminar



## Ejemplo en Java - Métodos en Cuenta Bancaria

```
public double getSaldo() {  
    return saldo;  
}
```

```
public void depositar (double monto) {  
    saldo = saldo + monto;  
}
```

```
public void extraer(double monto) {  
    saldo = saldo - monto;  
}
```

```
public void transferir(double monto, CajaDeAhorro cuentaDestino) {  
    saldo = saldo - monto;  
    cuentaDestino.depositar(monto);  
}
```

# Métodos

- ¿Qué es un método?
  - Es la contraparte funcional del mensaje.
  - Expresa la forma de llevar a cabo la semántica propia de un mensaje particular (el *cómo*).
- Un método puede realizar básicamente 3 cosas:
  - Modificar el estado interno del objeto.
  - Colaborar con otros objetos (enviándoles mensajes).
  - Retornar y terminar
  - ... ¿y la entrada salida?

## Entrada salida con objetos

- En la actualidad se habla de “lógica de dominio” y “lógica de interfaz” como asuntos separados
- En un sistema diseñado correctamente, un objeto de dominio no debería realizar ninguna operación vinculada a la interfaz (mostrar algo) o a la interacción (esperar un “input”)
- Si no puedo hacer I/O, ¿cómo pruebo mis objetos?
  - Con tests unitarios (próximamente )

# Formas de Conocimiento

- Un objeto solo puede enviar mensajes a otros que conoce
- Para que un objeto conozca a otro lo debe poder “nombrar”. Decimos que se establece una ligadura (binding) entre un nombre y un objeto.
- Podemos identificar tres formas de conocimiento o tipos de relaciones entre objetos.
  - Conocimiento Interno: Variables de instancia.
  - Conocimiento Externo: Parámetros.
  - Conocimiento Temporal: Variables temporales.
- Existe una cuarta forma de conocimiento especial: las pseudo-variables (como “this” o “self” y “super”)

# Clases e instancias

- Una clase es una descripción abstracta de un conjunto de objetos.
- Las clases cumplen tres roles:
  - Agrupan el comportamiento común a sus instancias.
  - Definen la *forma* de sus instancias.
  - *Crean objetos que son instancia de ellas*
- En consecuencia todas las instancias de una clase se comportan de la misma manera.
- Cada instancia mantendrá su propio estado interno.

# Especificación de Clases

- Las clases se especifican por medio de un nombre, el estado o estructura interna que tendrán sus instancias y los métodos asociados que definen el comportamiento
- Gráficamente:

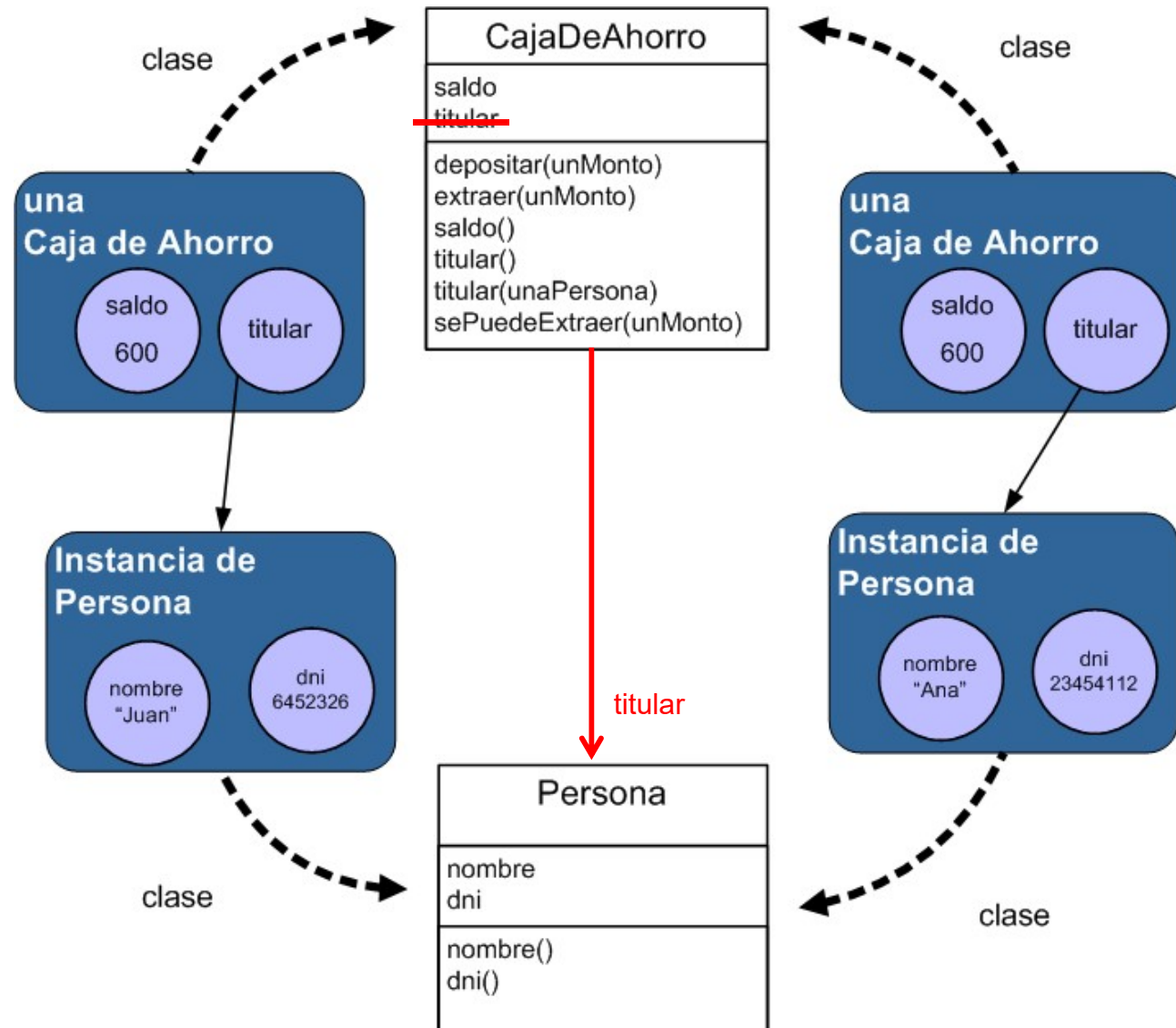
**Variables de Instancia**  
Los nombre de las v.i. se escriben en minúsculas y sin espacios

CajaDeAhorro
saldo titular
depositar(unMonto) extraer(unMonto) saldo() titular() titular(unaPersona) sePuedeExtraer(unMonto)

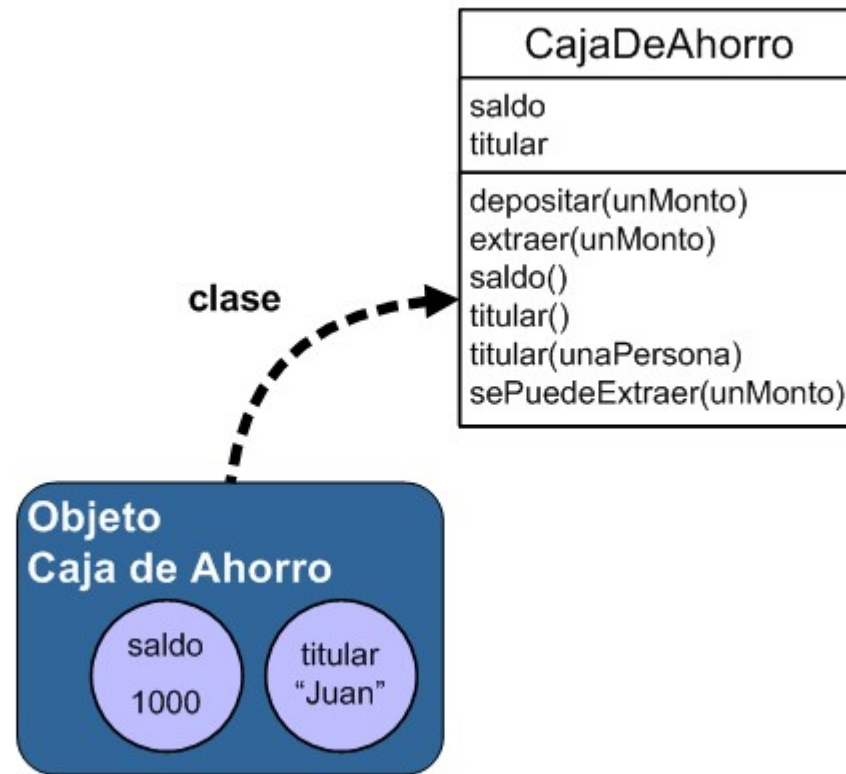
**Nombre de la Clase**  
Comienzan con mayúscula y no posee espacios

**Protocolo**  
Para cada mensaje se debe especificar como mínimo el nombre y los parámetros que recibe

# Ejemplo de clases e instancias



# Envío de mensajes con clases (Method lookup)





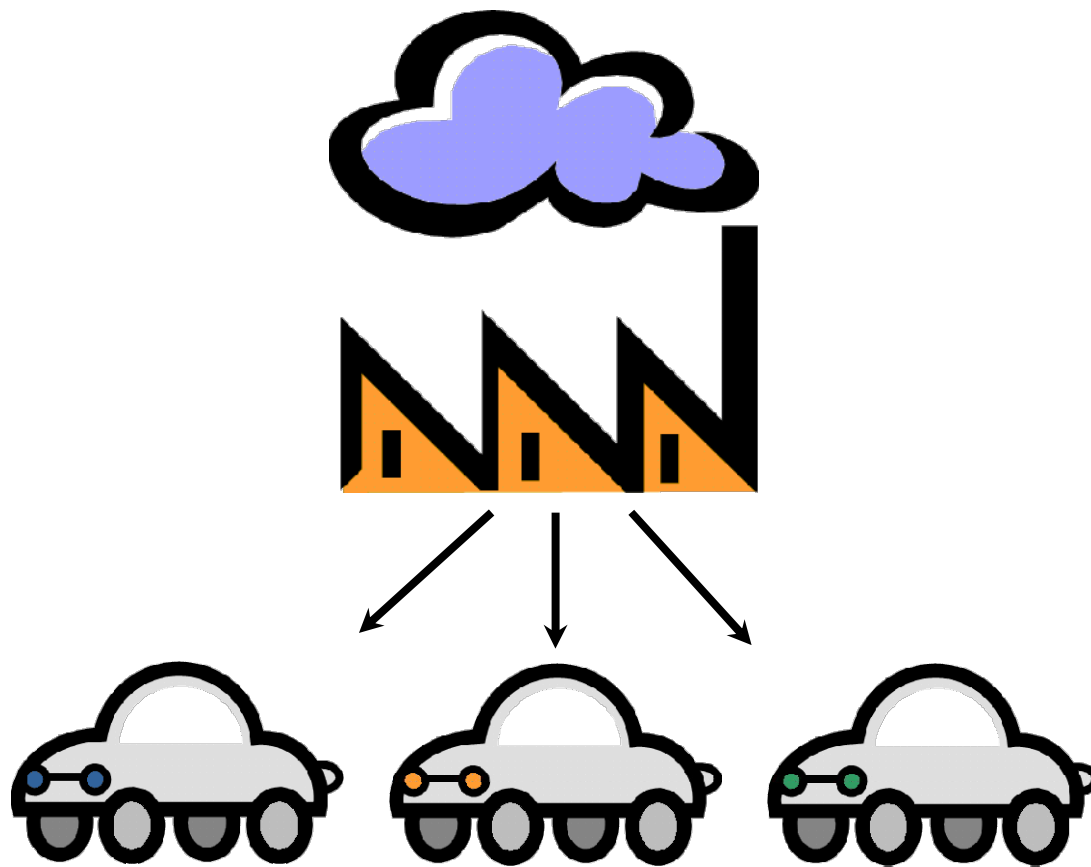
## Method lookup (búsqueda de métodos)

Cuando un **objeto** recibe un mensaje, se busca un método con la firma correspondiente (nombre y parámetros) en la clase de la cual es instancia.

- Si se lo encuentra, se lo ejecuta “en el contexto del objeto”
- Si no se lo encuentra, tendremos un error (o excepción) en tiempo de ejecución
  - Lenguajes estáticamente tipados, como Java, encuentran estos problemas al compilar
- Esto se conoce como “Dynamic Binding” y es clave para la OO

# Creación de Objetos

- ¿Cómo creamos nuevos objetos?
- Instanciación



# Instanciación

- Es el mecanismo de creación de objetos.
- Los objetos se *instancian* a partir de un molde.
- La **clase** funciona como molde.
- Un nuevo objeto es una *instancia* de una clase.
- Todas las instancias de una misma clase
  - Tendrán la misma estructura interna.
  - Responderán al mismo protocolo (los mismos mensajes) de la misma manera (los mismos métodos).

# Creación de Objetos

- Comúnmente se utiliza la palabra reservada **new** para instanciar nuevos objetos.
- Quien crea objetos? Cuando los crea?

```
/** Método en alguna otra clase del sistema **/  
public CajaDeAhorro abrirCajaDeAhorro(Cliente titular  
    , double depositoInicial)  
{  
    CajaDeAhorro nuevaCuenta = new CajaDeAhorro(titular);  
    nuevaCuenta.depositar(depositoInicial);  
    cuentas.add(nuevaCuenta);  
    return nuevaCuenta;  
}
```

# Inicialización

- Para que un objeto esté listo para llevar a cabo sus responsabilidades hace falta *inicializarlo*.
- ¿De dónde sacamos esos valores iniciales?

```
public CajaDeAhorro(Cliente unCliente) {  
    saldo = 0;  
    titular = unCliente;  
}
```

```
public CajaDeAhorro(Cliente unCliente,  
                    double saldoInicial) {  
    saldo = saldoInicial;  
    titular = unCliente;  
}
```

# Identidad

- Las variables (de instancia, temporales, parámetros) son apuntadores a objetos
- En cualquier momento, mas de una variable pueden apuntar a un mismo objeto
- Para saber si apuntan al mismo objeto, utilizo “==”

```
public void asignarNuevoResponsable(Persona alguien) {  
    if (alguien == responsable) {  
        // Emitir un error u excepción  
        // El nuevo responsable debe ser otra persona  
    }  
}
```

- ¿Que pasa si antes de asignarlo, cambiamos su nombre y dni?

# Igualdad

- Dos objetos (por ende, no idénticos) pueden ser iguales
- La igualdad se define en función del dominio
- Para saber si dos objetos son iguales, uso “equals()”

```
public void asignarNuevoResponsable(Persona alguien) {  
    if (alguien.equals(responsable)) {  
        // Emitir un error u excepción  
        // El nuevo responsable debe ser otra persona  
    }  
}
```

- ¿Que pasa si antes de asignarlo, cambiamos su nombre y dni?
- El equals() puede redefinirse, como cualquier método. El “==” no puede redefinirse.

# this

- **this** (o en algunos lenguajes self) es una “pseudo-variable”
  - no puedo asignarle valor
  - Toma valor automáticamente cuando un objeto comienza a ejecutar un método
- **this** hace referencia al objeto que ejecuta el método (al receptor del mensaje que resultó en la ejecución del método)
- Se utiliza para:
  - Descomponer métodos largos (top down)
  - Reutilizar comportamiento repetido en varios métodos
  - Aprovechar comportamiento heredado (proximamente ...)
- En algunos lenguajes (p.e. Java):
  - Puede obviarse (es implícito), aunque en OO1 preferimos no hacerlo
  - Para desambiguar referencia a las variables de instancia del objeto



# Reutilizar comportamiento repetido

*// Moverse*

```
public void translateBy(Point2D point) {  
    this.position.translateBy(point);  
    this.disableShields();  
    this.energy -= 1;  
}
```



*//Disparar*


```
public void fireUpon(Ship ship) {  
    ship.takeFireFrom(this.position);  
    this.disableShields();  
    this.energy -= 1;  
}
```



# Reutilizar comportamiento repetido

```
// Moverse
public void translateBy(Point2D point) {
    this.position.translateBy(point);
    this.disableShields();
    this.energy -= 1;
}

//Disparar
public void fireUpon(Ship ship) {
    ship.takeFireFrom(this.position);
    this.disableShields();
    this.energy -= 1;
}
```



The diagram illustrates code reuse. Two red arrows point from the `this.disableShields();` and `this.energy -= 1;` lines in the `translateBy` and `fireUpon` methods to the corresponding lines in the `private void payThePrice()` method. Red brackets are placed next to the `this.disableShields();` and `this.energy -= 1;` lines in both the `translateBy` and `fireUpon` methods, indicating the shared behavior.

```
private void payThePrice() {
    this.disableShields();
    this.energy -= 1;
}
```

# Reutilizar comportamiento repetido

```
// Move
public void translateBy(Point2D point) {
    this.position.translateBy(point);
    this.payThePrice();
}
```

```
//Disparar
public void fireUpon(Ship ship) {
    ship.takeFireFrom(this.position);
    this.payThePrice();
}
```

```
private void payThePrice() {
    this.disableShields();
    this.energy -= 1;
}
```

# Simulemos ser objetos



```
public void depositar (double monto) {  
    saldo = saldo + monto;  
}
```

```
public void extraer(double monto) {  
    saldo = saldo - monto;  
}
```

```
public void transferir(double monto, CajaDeAhorro cuentaDestino) {  
    saldo = saldo - monto;  
    cuentaDestino.depositar(monto);  
}
```