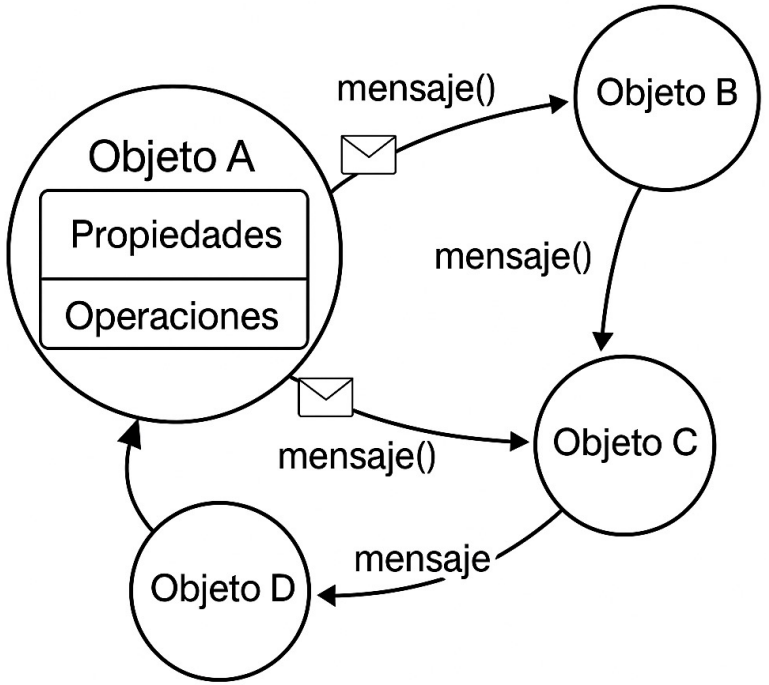

Fundamentos de la Orientación a Objetos (algunos)
ORIENTACIÓN A OBJETOS 1

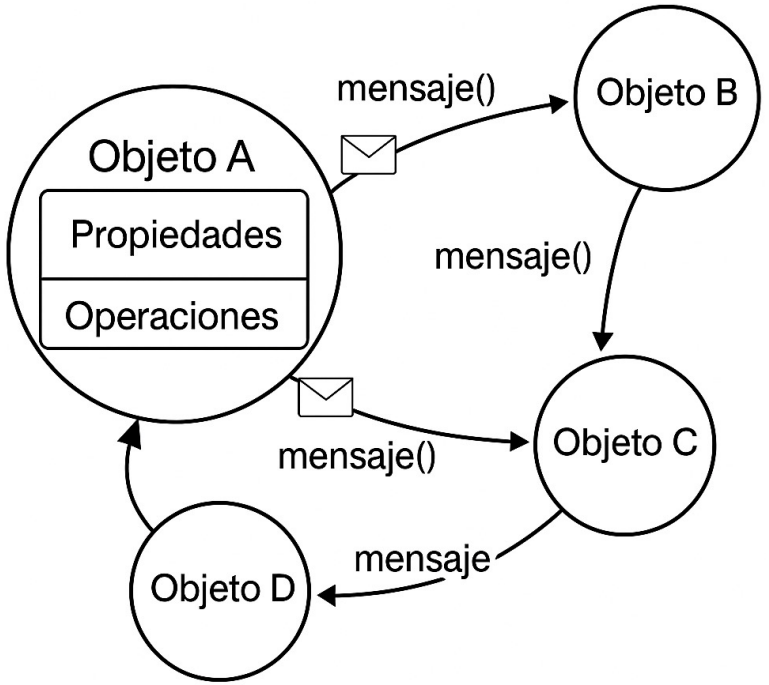
El sistema orientado a objetos

- Red (grafo) de objetos
- Cada objeto encapsula
  - Propiedades (atributos, variables de instancia)
  - Operaciones (métodos)
- Todo cómputo ocurre en algún objeto
- No hay un objeto "main".
- Cómputo y datos ya no se piensan por separado



Pensamos diferente …

- En lugar de una jerarquía "procedimientos y sub procedimientos" tenemos:
  - una red de objetos
  - posibles jerarquías de composición
  - posibles jerarquías de subclasificación
- Pensamos que "cosas" hay en nuestro software (los objetos) y cómo se comunican/relacionan entre sí.
- Mientras que la estructura sintáctica es "lineal" el programa en ejecución no lo es
- Hay un "shift" mental crítico en forma en la cuál pensamos el software como objetos



¿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, …)
- Un objeto tiene:
  - **Conocimiento (o estado interno)**: en base a sus relaciones con otros objetos y su estado interno
  - **Comportamiento**: conjunto de mensajes que un objeto sabe responder
  - **Identidad**: para distinguir un objeto de otro (independiente de sus propiedades)

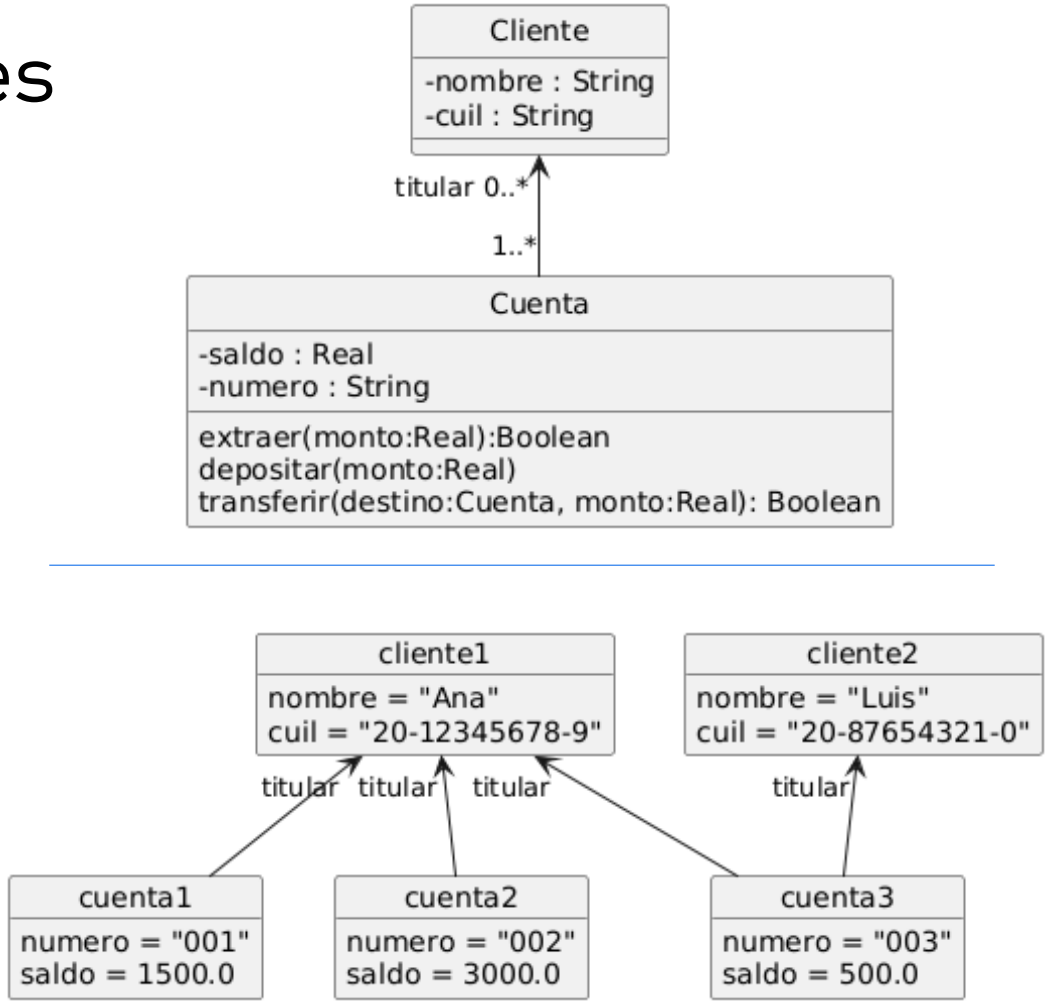
Encapsulamiento y ocultamiento

- Encapsulamiento: Implica agrupar en un mismo módulo (objeto) los datos y el comportamiento que opera sobre esos datos. El objeto es el "envoltorio" que asegura cohesión.
- Ocultamiento de información: Un aspecto dentro del encapsulamiento. Las decisiones de diseño que puedan cambiar (representación interna, estructuras de datos, algoritmos) están protegidas tras interfaces estables. Así se logra bajo acoplamiento.

encapsulamiento = "poner juntos datos + operaciones", y **ocultamiento** = "proteger detalles detrás de la interfaz pública".

Instancias vs. Clases

- Cuando diseñamos/programamos, describimos clases
- Cuando se ejecuta el programa tenemos objetos (instancias)
  - Se crean dinámicamente durante la ejecución (y eventualmente se guardan)
  - Se obtienen de algún repositorio (o a partir de otro objeto)
  - Reciben mensajes y ejecutan métodos
  - En los métodos, se envían mensajes a otros objetos



¿Qué es un objeto?

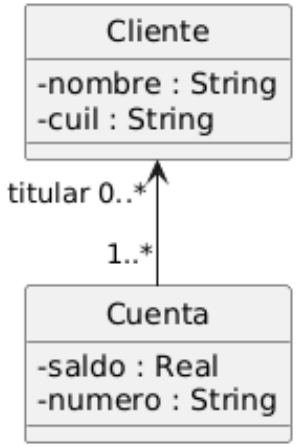
- Es una abstracción de una entidad del dominio del problema. Ejemplos: Persona, Producto, Cuenta Bancaria, Auto, Plan de Estudios,….
- Puede representar también conceptos del espacio de la solución (estructuras de datos, tipos "básicos", archivos, ventanas, conexiones, iconos, adaptadores, ...)
- Un objeto tiene:
  - **Conocimiento (o estado interno)**: en base a sus relaciones con otros objetos y su estado interno
  - **Comportamiento**: conjunto de mensajes que un objeto sabe responder
  - **Identidad**: para distinguir un objeto de otro (independiente de sus propiedades)

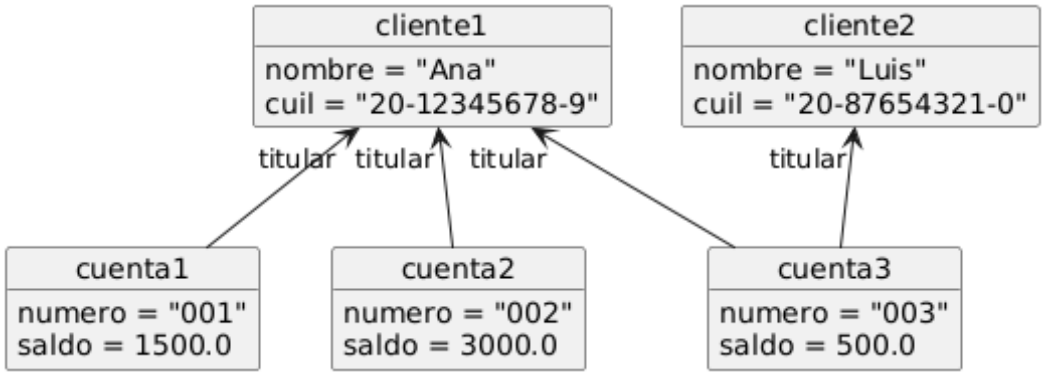
Formas de conocimiento

- Un objeto **solo puede enviar mensajes a otros que conoce**
- Las variables establecen ligaduras entre un nombre y un objeto.
- Podemos identificar tres formas de conocimiento o tipos de relaciones entre objetos.
- Conocimiento Interno: **Variables de instancia / estado del objeto**
- Conocimiento Externo: Parámetros.
- Conocimiento Temporal: Variables temporales.
- Existe una cuarta forma de conocimiento especial: las pseudo-variables (como “this” o “self” y “super”)

Conocimiento (estado interno)

- El estado interno de un objeto está dado por:
  - Propiedades básicas (intrínsecas) del objeto.
  - Otros objetos que necesita
- Es **privado** del objeto; ningún otro objeto debe acceder accederlo
- El estado interno del objeto se mantiene en sus **variables de instancia**
- Todas las instancias de una clase tienen las mismas variables de instancia
- Solo se accede en las operaciones del objeto (ocultamiento de información)



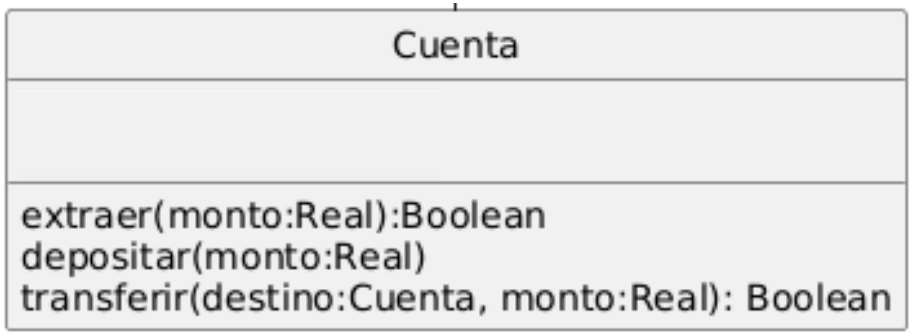


¿Qué es un objeto?

- 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, ...)
- Un objeto tiene:
  - **Conocimiento (o estado interno)**: en base a sus relaciones con otros objetos y su estado interno
  - **Comportamiento**: conjunto de mensajes que un objeto sabe responder
  - **Identidad**: para distinguir un objeto de otro (independiente de sus propiedades)

Comportamiento de un objeto

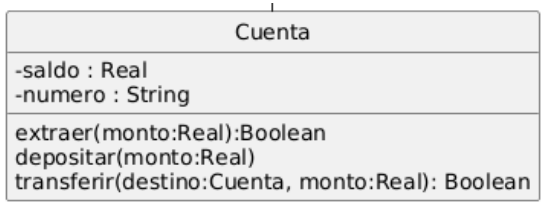
- El comportamiento de un objeto esta dado por el conjunto de **mensajes que entiende**
- Otros objetos le envían mensajes para que haga algo, o para obtener algo
- Cuando un objeto recibe un mensaje, **ejecuta un método**
- Todas las instancias de una clase entienden los mismos mensajes
- Todas las instancias de una clase ejecutan un mismo método, para un mismo mensaje




```
cuentaB.depositar(500);
cuentaA.depositar(1000);
cuentaA.transferir(cuentaB,200);
cuentaB.extraer(700);
```

Método (unidad mínima de comportamiento)

- Un método es una operación definida en el contexto de una clase
- Lo ejecuta un objeto (instancia de una clase) al recibir un mensaje
- Puede recibir parámetros (otros objetos)
- Puede devolver algo (un objeto) o nada
- Puede hacer uso de las variables del objeto receptor
- Puede modificar al objeto que lo ejecuta
- Puede enviar mensajes (incluso al propio objeto)




```
public boolean transferir(Cuenta d, double m) {
    if (saldo >= m) {
        this.extraer(m);
        d.depositar(m);
        return true;
    } else {
        return false;
    }
}
```

```
cuentaA.transferir(cuentaB,200);
```

Envío de mensajes

```
cuentaSueldo.transferir(cuentaDeAhorros, 1000);
```

- Una variable (de instancia, temporal, parámetro) actúa como receptor del mensaje
  - cuentaSueldo
- El objeto al que apunta cuentaSueldo es instancia de alguna clase (no sabemos cuál)
- El mensaje indica lo que se quiere que el objeto receptor haga
  - transferir
- El mensaje incluye parámetros
  - una variable que referencia a un objeto (**cuentaDeAhorros**), y un número (1000)
- **No sabemos** que método ejecuta el objeto al recibir el mensaje transferir, **y no nos importa**

Binding dinámico / búsqueda de método

- 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 receptor”
- 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”
- Permite desacoplar “lo que quiero hacer” (invocaciones) de las “formas de hacerlo” (implementaciones)
- Por eso ”no podemos” invocar métodos; solo “enviamos mensajes” y el binding dinámico decide que método se ejecuta

Instanciación (creación de objetos)

- Es el mecanismo de creación de objetos.
- Los objetos se instancian a partir de un molde (una clase)
- Todo objeto es una **instancia** de una clase (su clase)
- Un objeto nunca cambia de clase
- Todas las instancias de una misma clase
  - Tienen la misma estructura interna (mismas variables de instancia)
  - Responden al mismo protocolo (entienden los mismos mensajes) de la misma manera (ejecutando los los mismos métodos)

Observación: noten como usamos objeto e instancia como sinónimos

Instanciación (creación de objetos)

- Comúnmente se utiliza la palabra reservada new para instanciar nuevos objetos.
- ¿Quién crea objetos? ¿Cuándo 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 (constructores)

- Para que un objeto esté listo para recibir mensajes, hace falta inicializarlo.
- Los valores iniciales pueden asumirse (en algunos casis), o recibirse como parámetros

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

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

¿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, ...)
- Un objeto tiene:
  - **Conocimiento (o estado interno)**: en base a sus relaciones con otros objetos y su estado interno
  - **Comportamiento**: conjunto de mensajes que un objeto sabe responder
  - **Identidad**: para distinguir un objeto de otro (independiente de sus propiedades)

Identidad

- Las variables son punteros a objetos → mas de una variable pueden apuntar a un mismo objeto
- Para saber si dos variables apuntan al mismo objeto , utilizo "==", que por ser un operador no puede redefinirse.

```
// quiero saber si dos autos pertenecen a la misma persona
if (unAuto.getPropietario() == otroAuto.getPropietario()) {
    // los autos pertenecen a la misma persona
}
```

- El ejemplo asume que solo hay un objeto que representa a cada persona (suele ser el caso)

Igualdad

- Dos objetos pueden ser iguales; para saber si lo son uso **equals()**
- La igualdad se define en función del dominio → **equals()** puede redefinirse en cada clase

```
// quiero saber si dos autos son iguales
if (unAuto.equals(otroAuto)) {
    // los autos son iguales
}
```

En algún lugar

En la clase Automovil

```
public boolean equals(Automovil otroAuto) {
    return marca.equals(otroAuto.getMarca()) &&
           modelo.equals(otroAuto.getModelo());
}
```

Igualdad e identidad (ejemplo)

```java
// implementación "aproximada" de equals en la clase Color
public boolean equals(Color otro) {
    return this.getRed() == otro.getRed() &&
           this.getGreen() == otro.getGreen() &&
            this.getBlue() == otro.getBlue();
}
```

En la clase Color

En algún otro lugar

```java
Color blanco = new Color(255, 255, 255);
Color otroBlanco = new Color(255, 255, 255);
Color unColor = blanco;

System.out.println(blanco == blanco); // true
System.out.println(blanco == unColor); // true
System.out.println(blanco == otroBlanco); // false
System.out.println(blanco.equals(unColor)); // true
System.out.println(blanco.equals(otroBlanco)); // true
System.out.println(blanco.equals(new Color(0,0,0))); // false
```

Identidad e igualdad: implicancias

- La identidad de un objeto se define en el momento de su creación (cuando comienza a existir)
- La identidad de un objeto deja de existir cuando el objeto es destruido (garbage collected)
- Todas sus propiedades de un objeto pueden cambiar, pero no su identidad
- Dos objetos (dos instancias) pueden ser iguales, pero no pueden ser la misma (idénticas)
- Dos variables (que son referencias a objetos), pueden:
  - Apuntar al mismo objeto (a la misma identidad)
  - Apuntar a dos objetos que no son el mismo, pero son iguales → equals()
  - Apuntar a dos objetos que no son el mismo, y no son iguales → ! equals()

Dado que las variables son punteros a objetos (identidades), y los objetos se relacionan por variables, no necesito identificadores únicos / claves primarias para mantener relaciones

Anatomía de una clase en Java

```java
// package ...

// import ...

public class CajaDeAhorro {

    private double saldo;
    private Cliente titular;

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

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

    public double getSaldo() {
```

Nombre de la case (coincide con el nombre del archivo)

Variables de instancia (conocimiento)

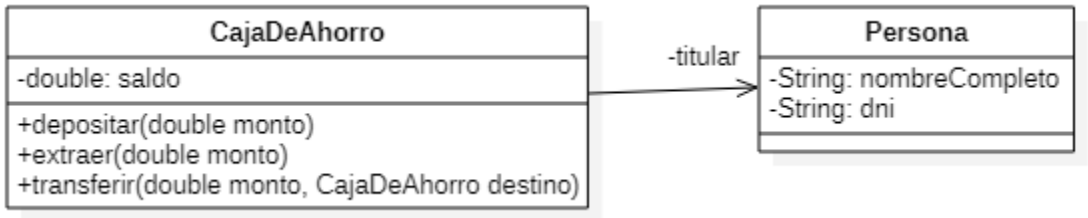
Constructores (inicialización)

```java
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);
}

Run | Debug
public static void main(String[] args) {
    Cliente cliente = new Cliente(nombre:"Juan Carlos Batman", cuil:"30-23537234-5");
    CajaDeAhorro cuentaA = new CajaDeAhorro(cliente);
    CajaDeAhorro cuentaB = new CajaDeAhorro(cliente);
}
}
```

Métodos (comportamiento)

Método estático main – solo útil en casos muy específicos – no vamos a usarlo

Pretendamos ser objetos (role playing)




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

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

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