Programación Orientada a Objetos

Centro de Investigación en Computación

Sesión # 02: Sábado 31 de Enero 2015

Prof. Miriam Pescador Rojas
miriam.pescador@gmail.com

Agenda para el día de hoy

- Introducción a la POO
- Atributos
- Métodos
- Constructores
  - Omisión
  - Por parámetros
- Métodos getters y setters
- Relaciones entre clases
  - Herencia
  - Composición
  - Agregación
  - Asociación
  - Dependencia

INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS

Clases de Objetos

Las clases de objetos son los elementos básicos de la programación orientada a objetos y representan conceptos o entidades significativos de un problema determinado.

CLASE: Modelo o plantilla que describe las características comunes de un conjunto de objetos

OBJETO: Abstracción de la vida real que define **características** y **comportamiento.** Ejemplar o instancia de una clase con datos propios.

Clases de Objetos

.

Una clase viene descrita por dos tipos de elementos:

- **Atributos** (variables de clase o miembros dato). Describen el estado interno de cada objeto

- **Operaciones** (métodos o miembros funcionales). Describen lo que se puede hacer con el objeto, los servicios que proporciona

Práctica #1

Descripción / Propósito(s):

- Definición de clases
- Estándar del lenguaje

Implementación de la clase Persona.java

```
public class Persona
{
    private String nombre, paterno, materno;
    private String telefono, correo, tipoContacto;

    /*Constructor por parametros:
        método especial que permite inicializar las variables de un objeto,
        no tiene tipo de retorno y solo se manda a llamar cuando
        se crea la instancia de la clase
    */
    public Persona(String n, String p, String m, String t, String c, String tc)
    {
        nombre = n;
        paterno = p;
        materno = m;
        telefono = t;
        correo = c;
        tipoContacto = tc;
    }

    public String nombreCompleto()
    {
        return nombre + ' ' + paterno + ' ' + materno;
    }
```

Características de las clases

- Los datos de una clase pueden ser de cualquier tipo definido por el lenguaje de programación
- La implementación de las operaciones se realiza en el interior de la definición de la clase, justo tras su declaración.
- El acceso a los atributos de la clase desde la implementación de las operaciones se realiza de forma directa

Diferencias entre clase y objeto

- La diferencia entre un objeto y una clase es que un objeto es una entidad concreta que existe en tiempo y espacio, mientras que una clase representa una abstracción, la "esencia" de un objeto.

Un objeto consta de:

- **Tiempo de vida:** La duración de un objeto en un programa siempre está limitada en el tiempo.
- La mayoría de los objetos sólo existen durante una parte de la ejecución del programa.
- Los objetos son creados mediante un mecanismo denominado *instanciación, y cuando dejan de* existir se dice que son *destruídos.*

Un objeto consta de:

- **Estado:** Todo objeto posee un estado, definido por sus *atributos. Con él se definen las* propiedades del objeto, y el estado en que se encuentra en un momento determinado de su existencia.

- **Comportamiento:** Todo objeto ha de presentar una interfaz, definida por sus *métodos, para que* el resto de objetos que componen los programas puedan interactuar con él.

Práctica #2

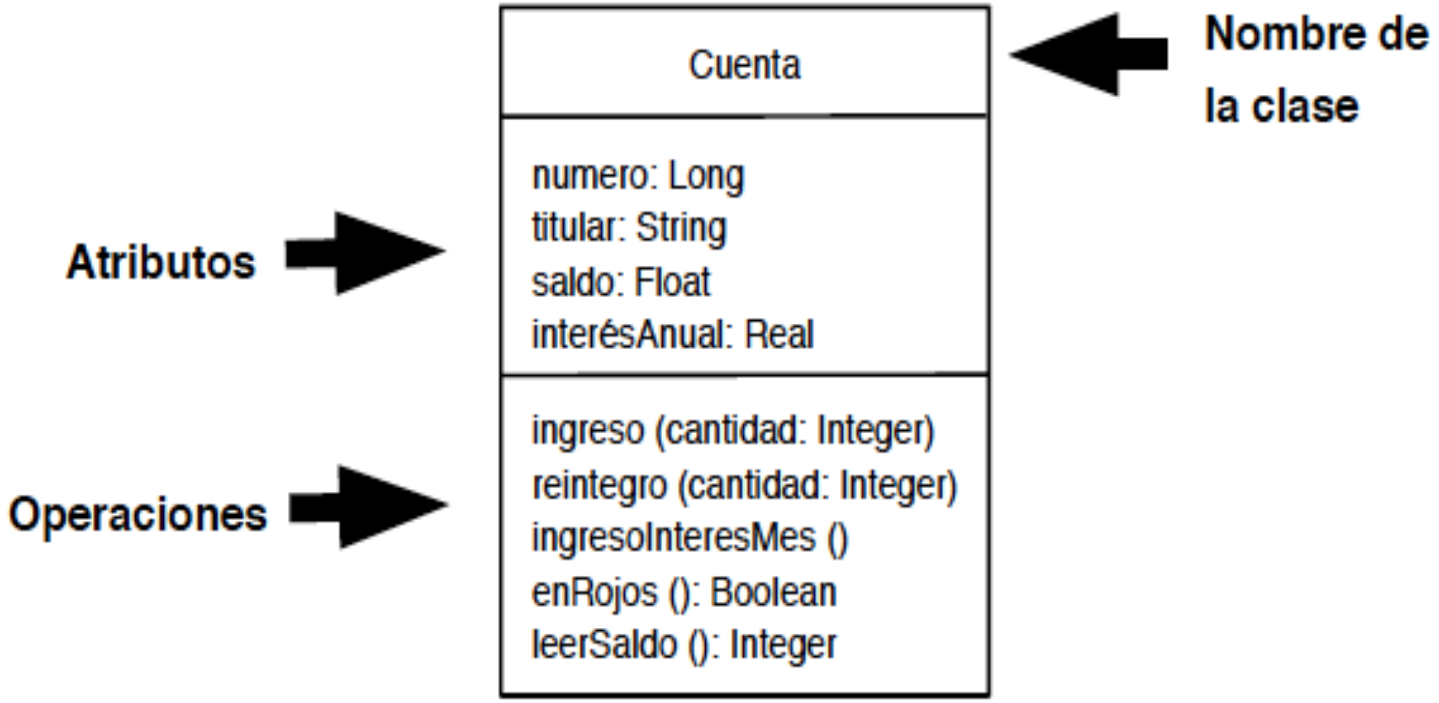
Descripción / Propósito(s):

- Definición de clases

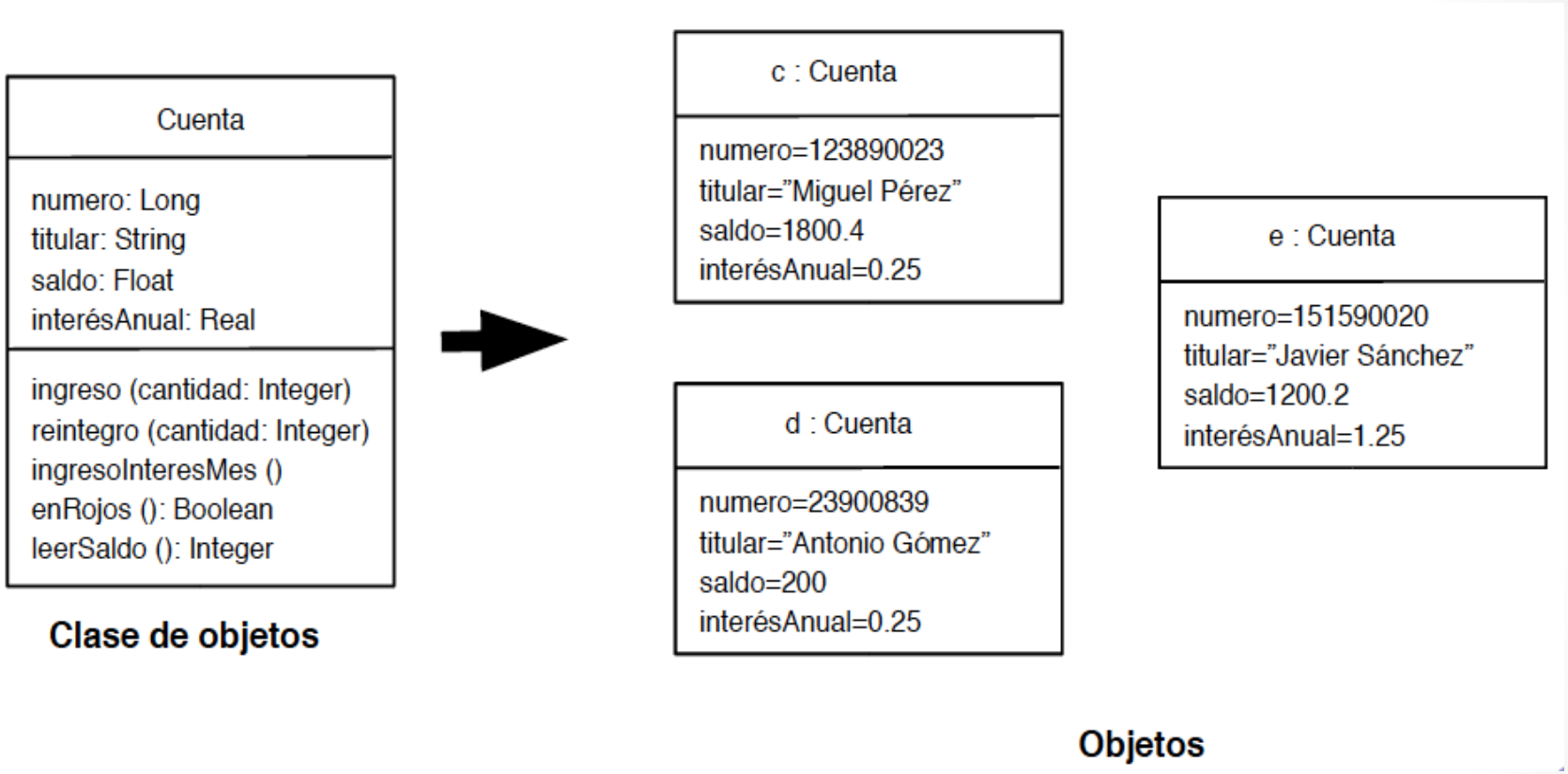
De las siguientes imágenes de objetos, defina las posibles clases

Ejemplo Clase Cuenta

Una clase de objetos describe las **características comunes** a un **conjunto de objetos**.



Instancias de una clase



Durante la ejecución de la aplicación se producirá la **instanciación** de esta clase, es decir, la creación de los objetos que representan cada uno de los individuos con sus características propias, es decir, valores específicos para sus atributos

Implementación

Atributos

Operaciones

```
class Cuenta {
    long numero;
    String titular;
    float saldo;
    float interesAnual;

    void ingreso (float cantidad) { }
    void reintegro (float cantidad) { }
    void ingresoInteresMes () { }
    boolean enRojos () { }
    float leerSaldo () { }
}
```

```
class Cuenta {
    long numero;
    String titular;
    float saldo;
    float interesAnual;

    void ingreso (float cantidad) {
        saldo += cantidad;
    }

    void reintegro (float cantidad) {
        saldo -= cantidad;
    }

    void ingresoInteresMes () {
        saldo += interesAnual * saldo / 1200;
    }

    boolean enRojos () {  return saldo < 0; }
    float leerSaldo () { return saldo; }
}
```

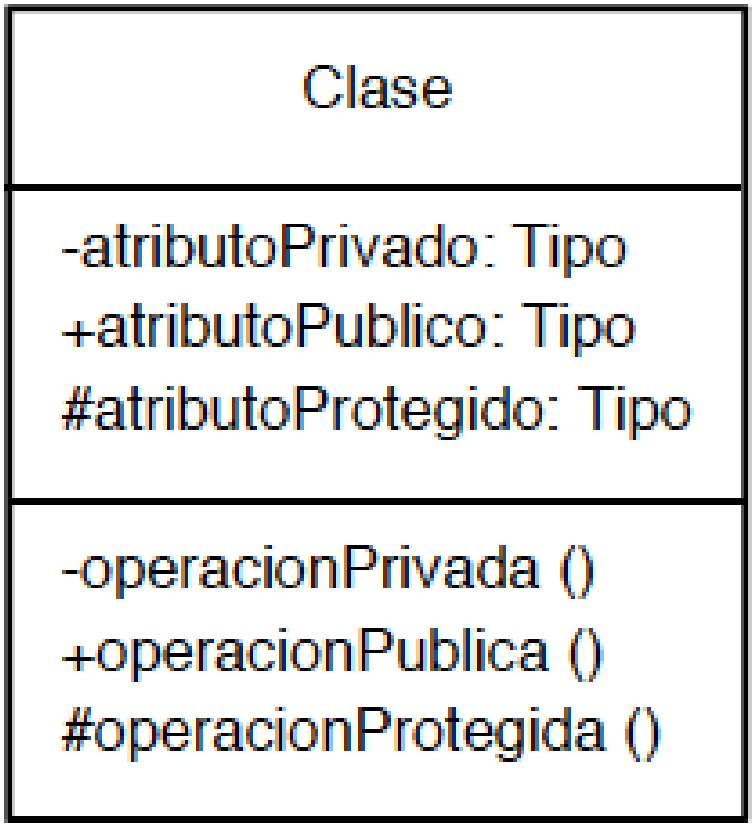
Protección de los Miembros

El principio de ocultación de información se plasma en los lenguajes OO en diversos mecanismos de protección de los miembros de la clase

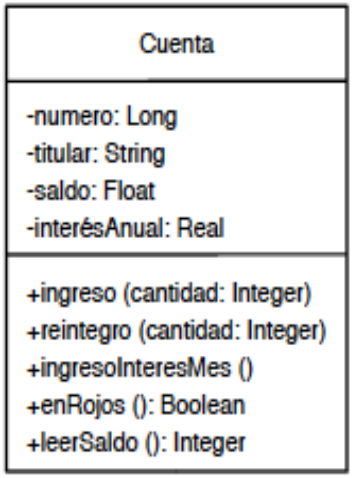
Se definen 3 niveles de protección para cada miembro de la clase en C#:

- **Miembros públicos (+).** Sin ningún tipo de protección especial
- **Miembros privados (-).** Inaccesibles desde el exterior de la clase
- **Miembros protegidos (#).** Similares a los privados aunque se permite su acceso desde las clases descendientes (herencia)

Modificadores de acceso



Ejemplo de implementación




```
class Cuenta {
    private long numero;
    private String titular;
    private float saldo;
    private float interesAnual;

    public void ingreso (float cantidad) {
        saldo += cantidad;
    }

    public void reintegro (float cantidad) {
        saldo -= cantidad;
    }

    public void ingresoInteresMes () {
        saldo += interesAnual * saldo / 1200;
    }

    public boolean enRojos () {  return saldo < 0; }
    public float leerSaldo () { return saldo; }
}
```

Constructores

- Un constructor tiene la función de inicializar las variables de la clase.
- Existen dos tipos de funciones:
  - Por Omisión
  - Por parámetros

Por Omisión: Da valores por omisión a las variables por ejemplo inicializar en 0 las variables o a NULL

Por parámetros. Recibe variables para inicializar las variables de mi clase

Constructor por omisión

- Ejemplo

```
Public class Cuenta
{

    Cuenta()
    {
        numero = 0;
        saldo = 2000.0; // saldo minimo requerido
        titular = null;
    }
}
```

Constructor parámetros

- Ejemplo

```
Public class Cuenta
{
     Cuenta(long n, double s, String t)
    {
          numero = n;
         saldo = s;
         titular = t;
    }
}
```

Creación de Objetos

- Para crear una instancia de clase hacemos uso del operador new y el constructor de la clase.
- El operador new sirve para reservar espacio en memoria cada vez que se crea un nuevo objeto
- El constructor de una clase es un método especial que:
  - solo se manda a llamar cuando se crea el objeto.
  - Va seguido del operador new.
  - Tiene el mismo nombre de la clase
  - No tiene tipo de retorno

Creación de objetos

- SINTAXIS

```
Cuenta c = new Cuenta( );
```

Definición de un constructor (dentro de la clase Cuenta)

```
public class Cuenta
{
    public Cuenta() {}
}
```

Métodos Getter y Setter

- Para tener acceso de lectura y escritura de un atributo de clase desde distintas partes del programa se utilizan métodos públicos

- Estos métodos nos permiten conocer el valor de la variable o establecer un valor

Métodos Getter y Setter

- En C# estos métodos se establecen de la siguiente manera

```
Class Cuenta
{
   //Atributos
    private int numero;
   //Propiedades y métodos
    public Numero
    {
        get { return numero;}
        set { nombre = value; }
    }
}
```

Práctica #2

Descripción / Propósito(s):

- Uso de Netbeans
- Creación de proyectos

a) Crear un nuevo proyecto “Figuras” e implementar las clases:

- Cuadrado.java
- Circulo.java
- Triangulo.java

Práctica #3

Descripción / Propósito(s):

- Manejo de modificadores de acceso
- Definición de métodos getters y setters

Implemente la clase Cuenta y agregue a los métodos get y set las siguientes validaciones:

La cadena String cliente no deberá ser vacía
El número de la cuenta no debe ser menor igual a 0
El saldo de la cuenta debe ser mayor igual a 1000 pesos

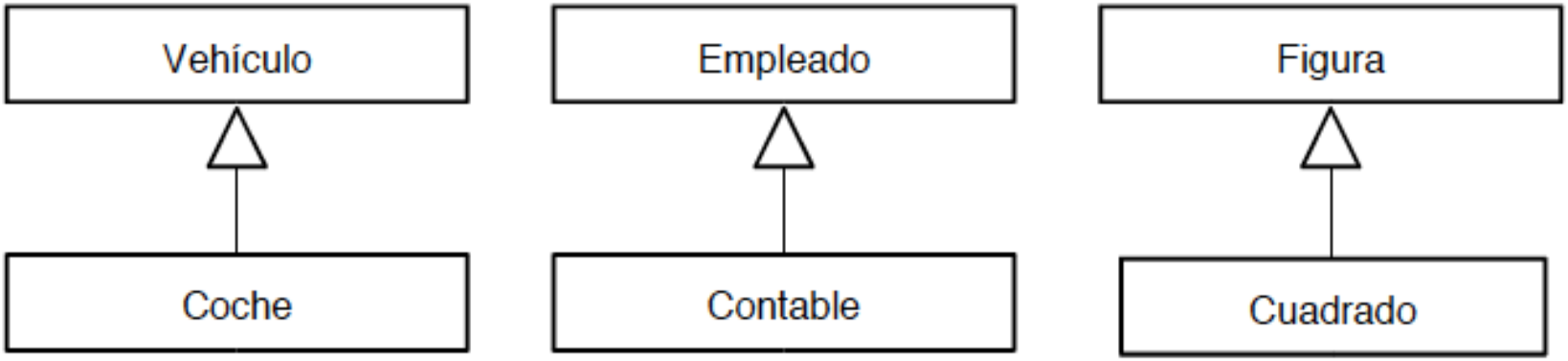
Relaciones entre clases

- Un conjunto de objetos aislados tiene escasa capacidad para resolver un problema. En una aplicación útil los objetos colaboran e intercambian información, mantienen distintos tipos de relaciones entre ellos

- A nivel de diseño, podemos distinguir entre 5 tipos de relaciones básicas entre clases de objetos:
  - dependencia, asociación, agregación, composición y herencia*

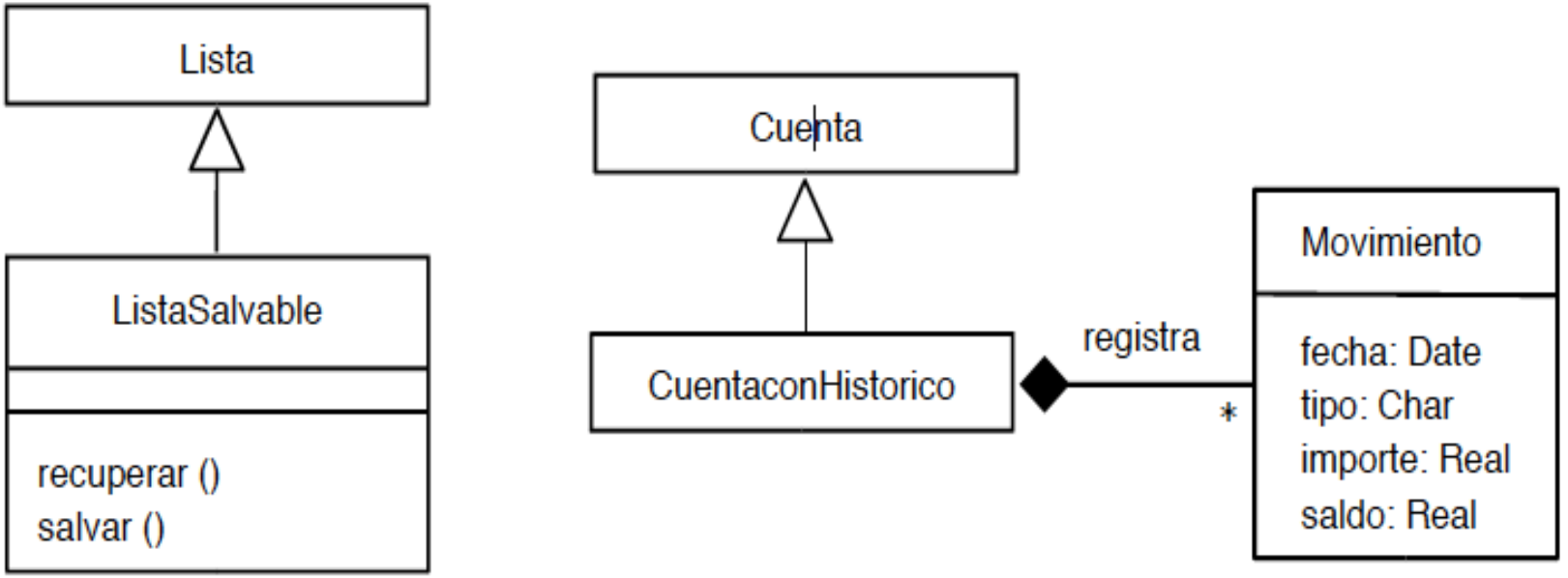
Aplicación de la Herencia

- Existen diferentes situaciones en las que puede aplicarse herencia:
- Especialización. Dado un concepto B y otro concepto A que representa una especialización de A, entonces puede establecerse una relación de herencia entre las clases de objetos que representan a A y B. En estas situaciones, el enunciado “A es un B” suele ser aplicable


Vehículo
Coche
Empleado
Contable
Figura
Cuadrado

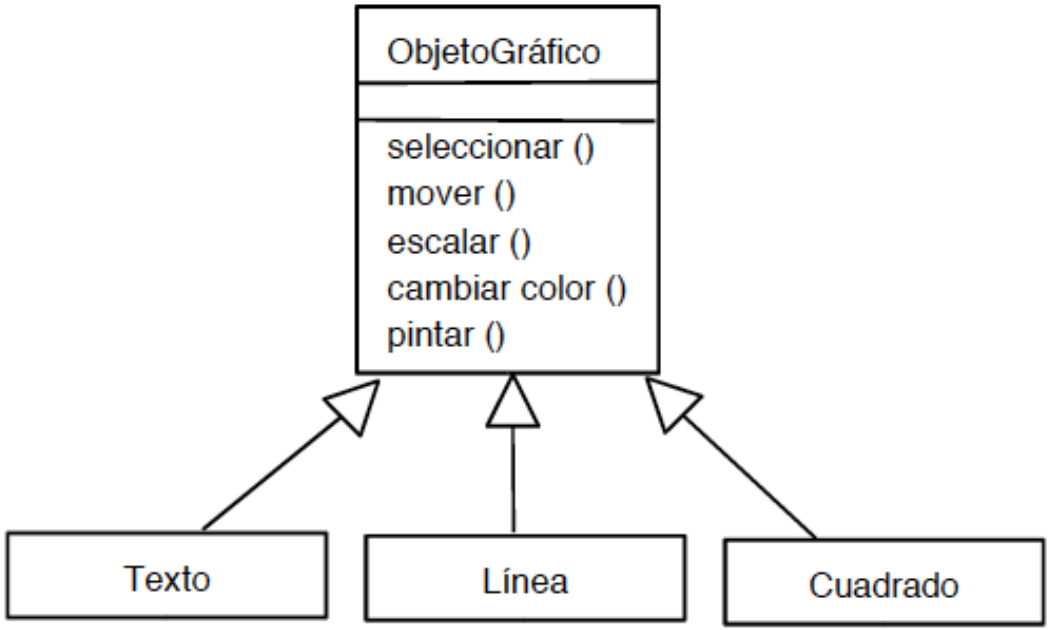
Aplicación de la Herencia

- Extensión. Una clase puede servir para extender la funcionalidad de una superclase sin que represente necesariamente un concepto más específico



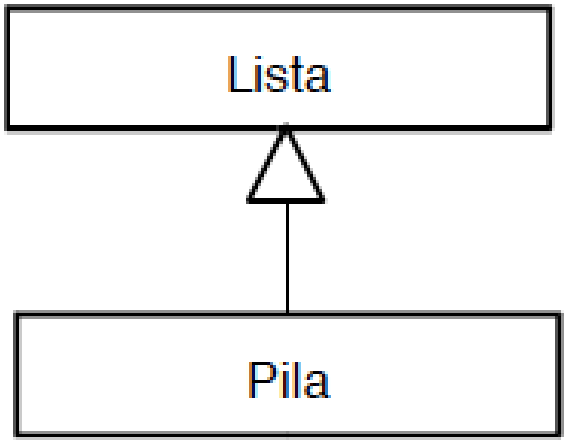
Aplicación de la Herencia

- Especificación. Una superclase puede servir para especificar la funcionalidad mínima común de un conjunto de descendientes. Existen mecanismos para obligar a la implementación de una serie de operaciones en estos descendientes

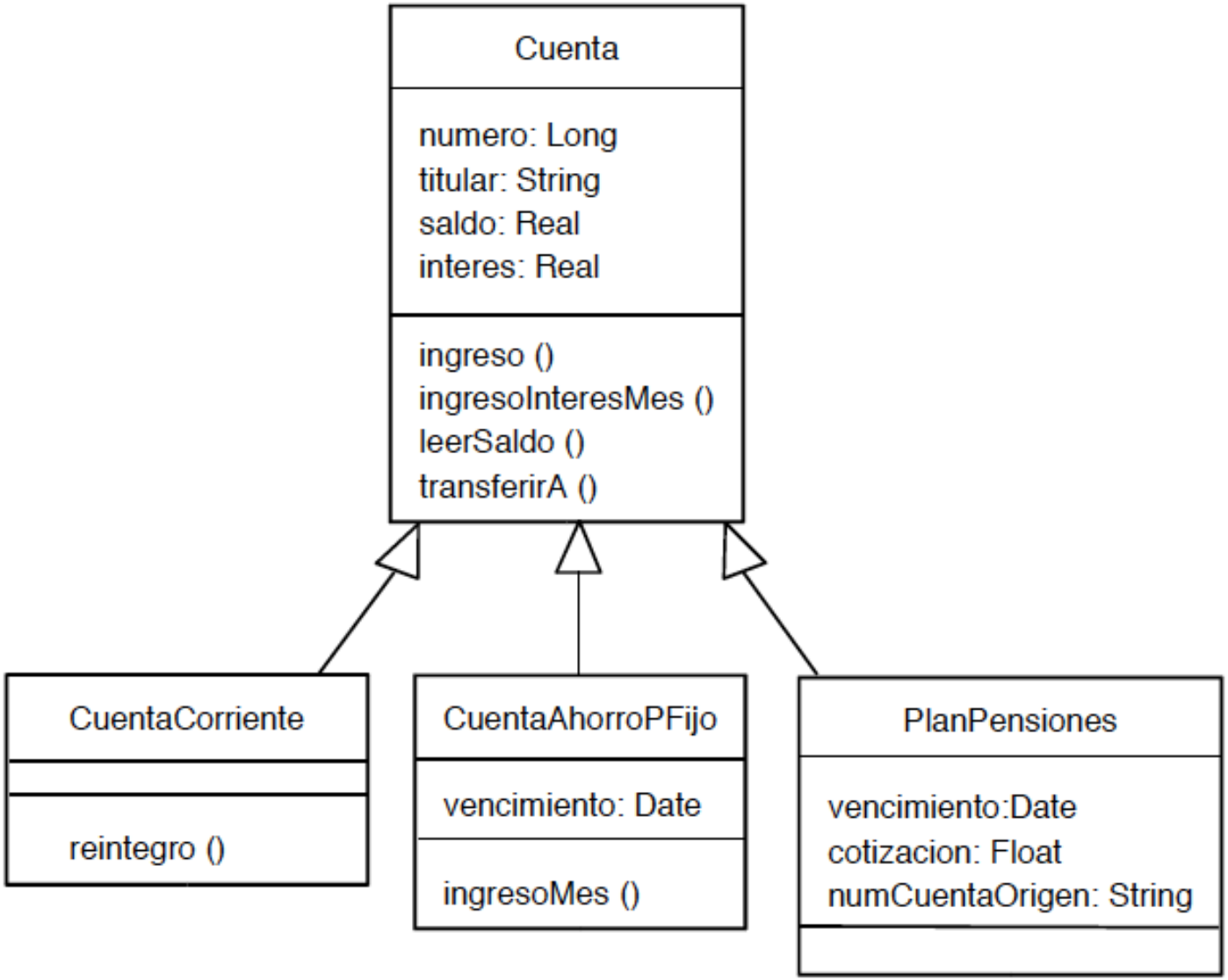


Aplicación de la herencia

- Construcción. Una clase puede construirse a partir de otra, simplemente porque la hija puede aprovechar internamente parte o toda la funcionalidad del padre, aunque representen entidades sin conexión alguna



Ejemplos de Herencia



Práctica #4

Descripción / Propósito(s):

- Definición de clases abstractas

Realice los siguientes pasos:

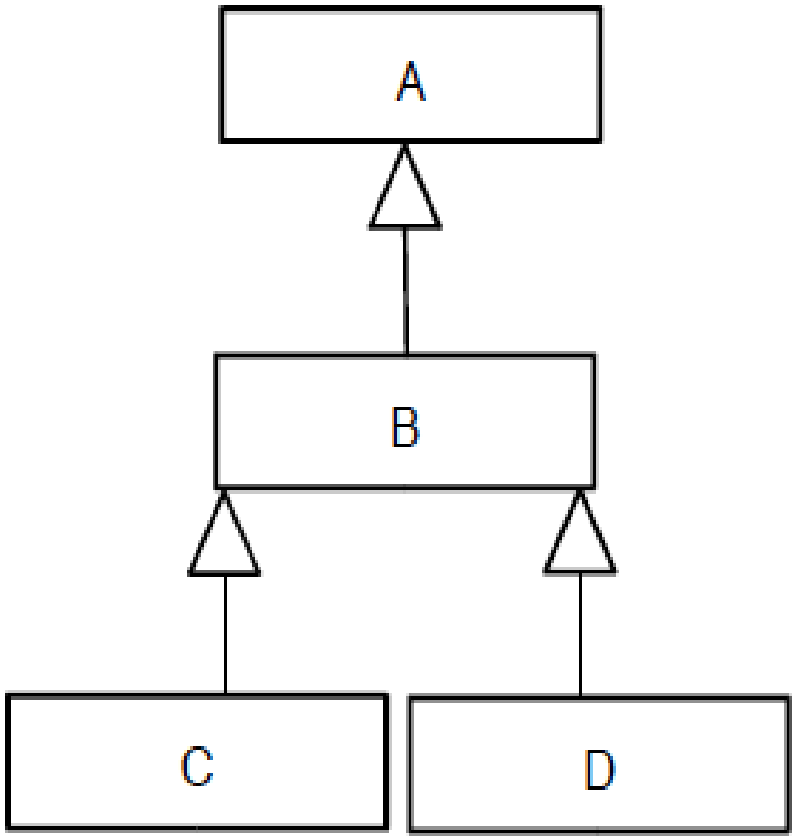
-Definir la clase Figura, Cuadrado, Circulo y Triángulo en un esquema de herencia.

-La clase Figura será de tipo abstract y definirá los métodos abstractos calcularArea y calcularPerimetro

Herencia

- La herencia es un mecanismo de la POO que permite construir una clase incorporando de manera implícita todas las características de una clase previamente existente.
- Sea una clase A. Si una segunda clase B hereda de A entonces decimos:
  - A es un ascendiente o superclase de B. Si la herencia entre A y B es directa decimos además que A es la clase padre de B
  - B es un descendiente o subclase de A. Si la herencia entre A y B es directa decimos además que B es una clase hija de A

Diagramas de herencia



Implementación

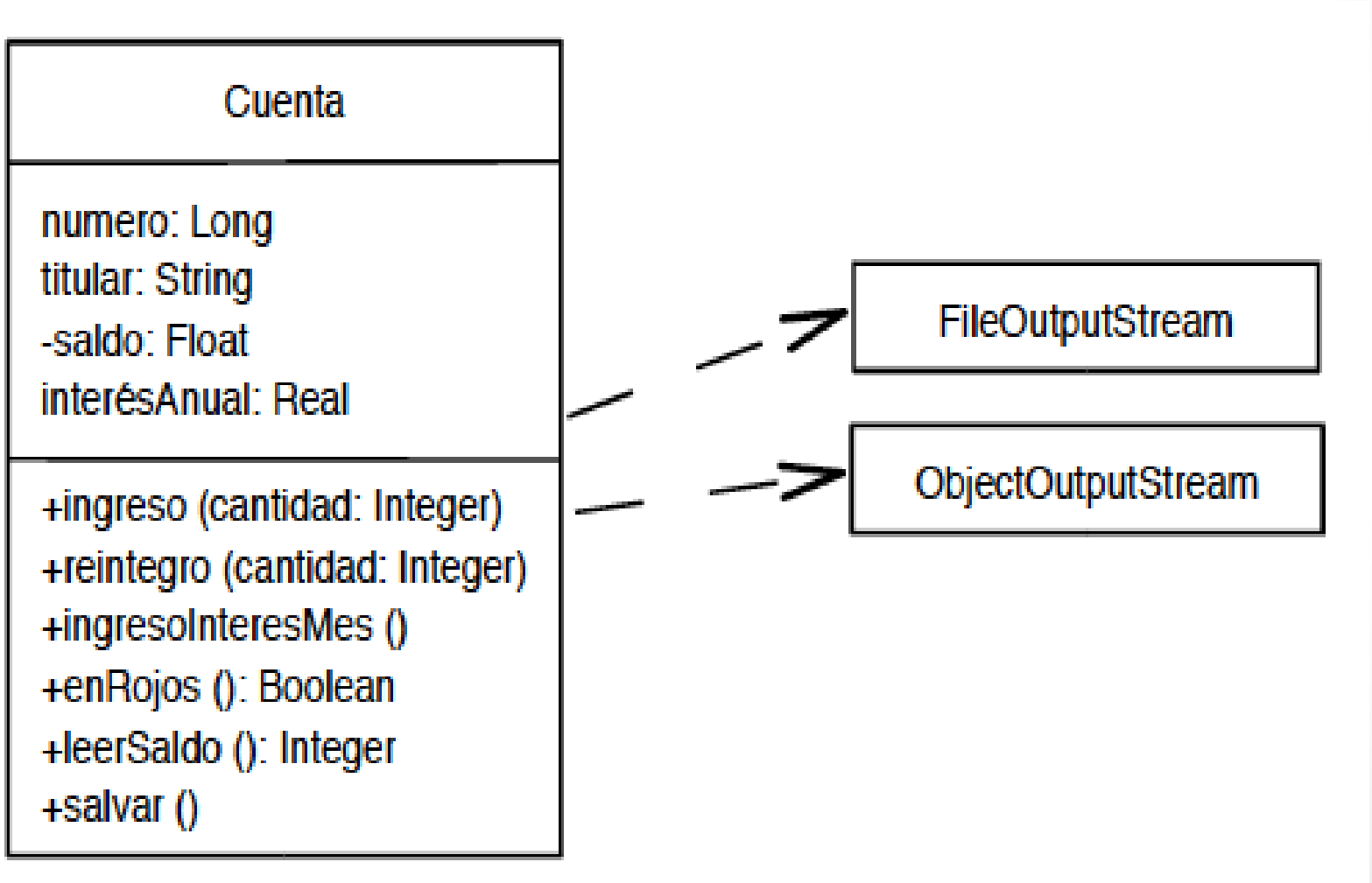
```
public class A
{
    public A() { }
}

public class B : A
{
    public B() { }
}
```

Dependencia

- La dependencia refleja que entre dos clases de objetos existe una posible colaboración temporal con algún propósito

- Una dependencia puede indicar la utilización de un objeto de una clase como argumento de una operación de otra o en su implementación

Ejemplo de la relación Dependencia

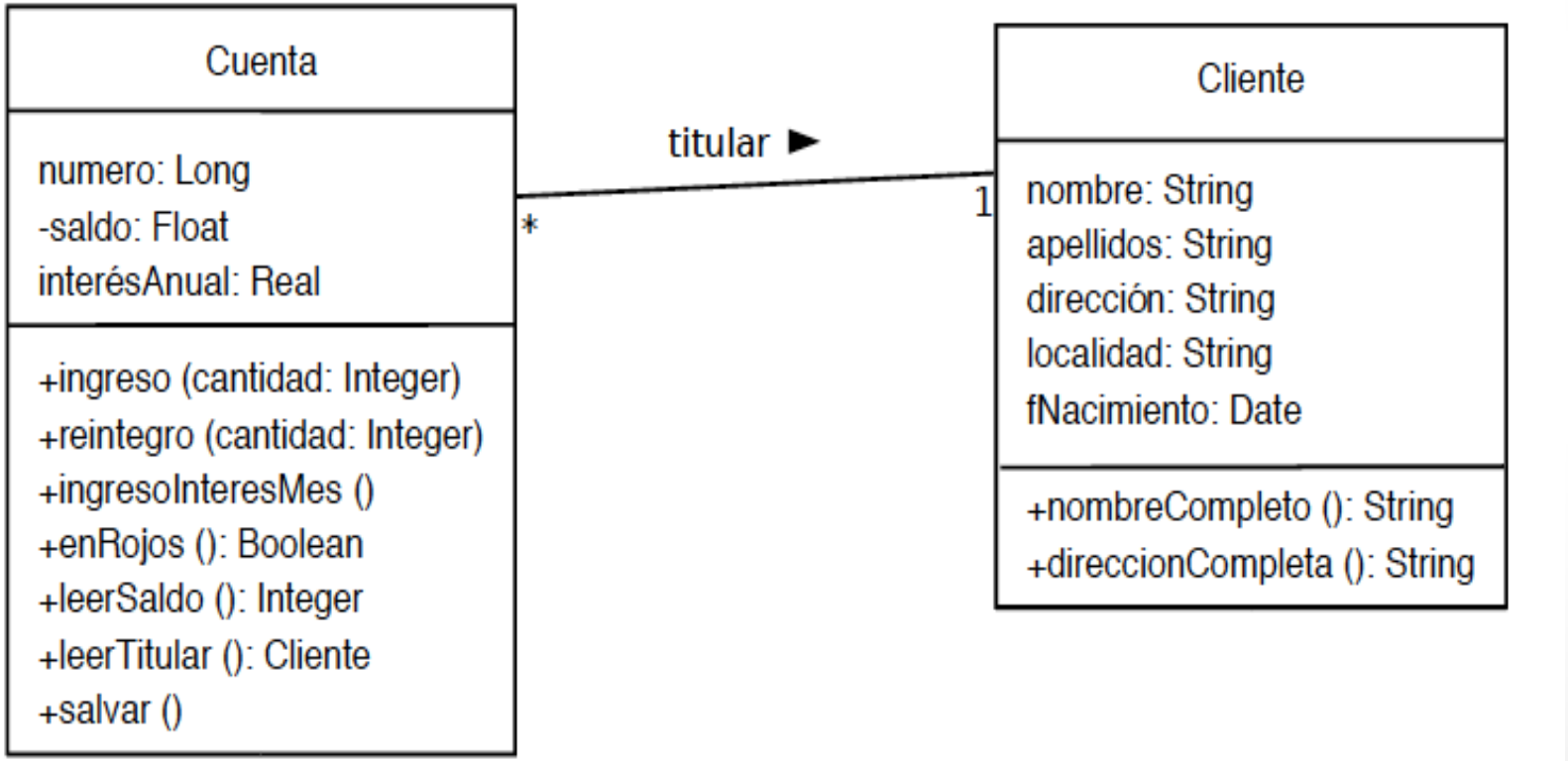


Asociación

- La asociación es la relación más importante y más común. Refleja una relación entre dos clases independientes que se mantiene durante creación de los objetos de dichas clases.

- En UML suele indicarse el nombre de la relación, el sentido de dicha relación y las cardinalidades en los dos extremos

Ejemplo Asociación



Asociación

- Una asociación se implementa en C# introduciendo referencias a objetos de la clase destino de la relación como atributos de la clase origen

- Si la relación tiene una cardinalidad superior a uno entonces será necesario utilizar un array o una estructura de datos dinámica (Array, ArrayList, List)

- Normalmente la conexión entre los objetos se realiza recibiendo la referencia de uno de ellos en el constructor u otra operación similar del otro.

Implementación

```
public class Cliente {
    String nombre, apellidos;
    String direccion, localidad;
    Date fNacimiento;

    Cliente (String aNombre, String aApellidos, String aDireccion,
        String aLocalidad, Date aFNacimiento) {
        nombre = aNombre;
        apellidos = aApellidos;
        direccion = aDireccion;
        localidad = aLocalidad;
        fNacimiento = aFNacimiento;
    }

    String nombreCompleto () { return nombre + " " + apellidos; }
    String direccionCompleta () { return direccion + ", " + localidad; }
}
```

Implementación

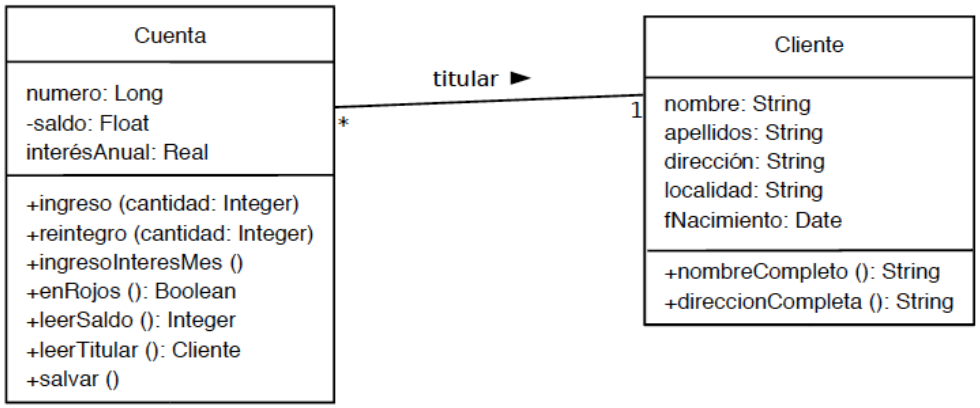
```
public class Cuenta {
    long numero;
    Cliente titular;
    private float saldo;
    float interesAnual;

  // Constructor general
    public Cuenta (long aNumero, Cliente aTitular, float aInteresAnual) {
        numero = aNumero;
        titular = aTitular;
        saldo = 0;
        interesAnual = aInteresAnual;
    }

    Cliente leerTitular () { return titular; }

  // Resto de operaciones de la clase Cuenta a partir de aquí
```

Práctica #5

Descripción / Propósito(s):

- Definición de clases
- Implementación de relaciones de clase

Implemente la clase Cliente y establezca la relación de asociación entre Cliente y Cuenta como lo muesta el diagrama

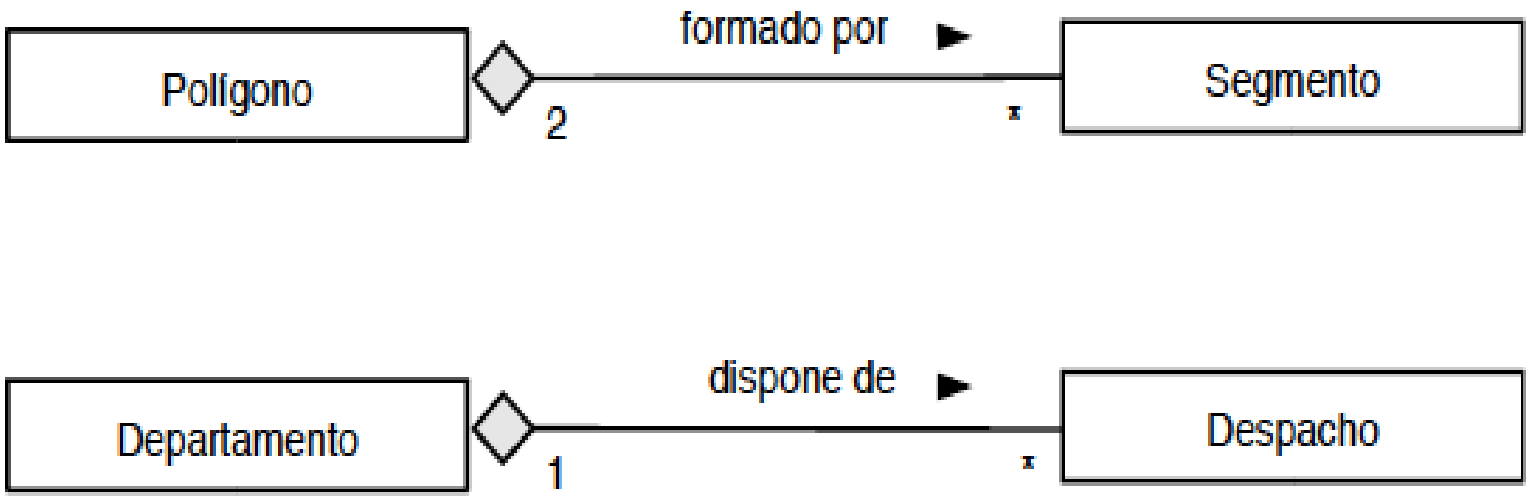


Agregación

- La agregación es un tipo especial de asociación donde se añade el matiz semántico de que la clase de donde parte la relación representa el “todo” y las clases relacionadas “las partes”.

- La implementación se realiza igual que la asociación

Ejemplo Agregación

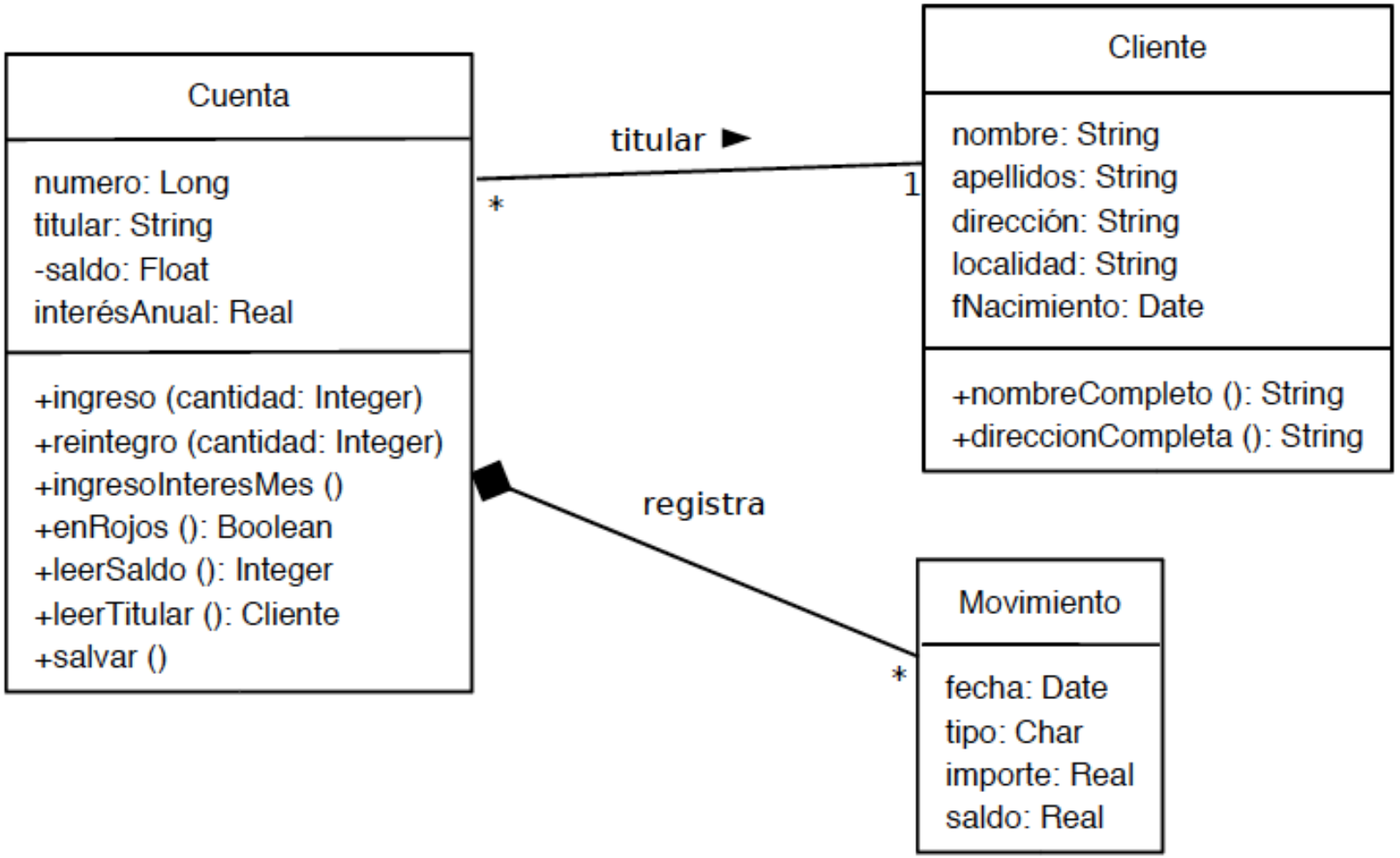


Composición

- La composición es un tipo de agregación que añade el matiz de que la clase “todo” controla la existencia de las clases “parte”. Es decir, normalmente la clase “todo” creará al principio las clases “parte” y al final se encargará de su destrucción

- Supongamos que añadimos un registro de movimientos a la clase *Cuenta, de* forma que quede constancia tras cada ingreso o reintegro

Ejemplo Composición



Práctica #6

Descripción / Propósito(s):

- Definición de clases
- Implementación de relaciones de clase

Implemente la clase Movimiento y establezca la relación de composición entre Cuenta y Movimiento como lo muesta el diagrama