Instituto Superior Santo Domingo	**UNIDAD Nº 1:** **Encapsulamiento y herencia.**	**TEMAS:** **Colaboración de clases y herencia.**	**Clase** **2**

Objetivos:

- Definición de varias clases.
- Relación de colaboración
- Relación de herencia

Introducción

A medida que los problemas son mayores es necesario asignar responsabilidades a distintas clases y definir relaciones de colaboración y herencia.

La programación orientada a objetos presenta como beneficios la posibilidad de encapsular atributos y métodos en una clase, esto hace posible que varias personas desarrollen un conjunto de clases y no haya problema con los nombres que definen para sus métodos y atributos. Esto debido que todo está contenido en la clase.

Pero las clases deben comunicarse una con otras. En el tiempo que uno diseña el programa tiene que especificar que clase colabora con otra.

Colaboración de clases

Normalmente un problema resuelto con la metodología de programación orientada a objetos no interviene una sola clase, sino que hay muchas clases que interactúan y se comunican.
Plantearemos un problema separando las actividades en dos clases.

Problema 1

Un banco tiene 3 clientes que pueden hacer depósitos y extracciones. También el banco requiere que al final del día calcule la cantidad de dinero que hay depositado.
Lo primero que hacemos es identificar las clases:
Podemos identificar la clase Cliente y la clase Banco.

Luego debemos definir los atributos y los métodos de cada clase:

Cliente
  atributos
    nombre
    monto

métodos
constructor
depositar
extraer
retornarMonto

Banco
atributos
3 Cliente (3 objetos de la clase Cliente)
1 Scanner (Para poder hacer la entrada de datos por teclado)
métodos
constructor
operar
depositosTotales

Creamos un proyecto en Eclipse llamado: ProyectoBanco y dentro del proyecto creamos dos clases llamadas: Cliente y Banco.

Cada clase debe ser creada con los pasos aprendidos en el apunte 1.

```
public class Cliente {
    private String nombre;
    private int monto;

    public Cliente(String nom) {
        nombre=nom;
        monto=0;
    }

    public void depositar(int m) {
        monto=monto+m;
    }

    public void extraer(int m) {
        monto=monto-m;
    }

    public int retornarMonto() {
        return monto;
    }

    public void imprimir() {
        System.out.println(nombre+" tiene depositado la
suma de "+monto);
    }
}
```

La segunda clase es:

```
public class Banco {
    private Cliente cliente1,cliente2,cliente3;

    public Banco() {
        cliente1=new Cliente("Juan");
        cliente2=new Cliente("Ana");
        cliente3=new Cliente("Pedro");
    }

    public void operar() {
        cliente1.depositar (100);
        cliente2.depositar (150);
        cliente3.depositar (200);
        cliente3.extraer (150);
    }

    public void depositosTotales ()
    {
        int t = cliente1.retornarMonto () +
cliente2.retornarMonto () + cliente3.retornarMonto ();
        System.out.println ("El total de dinero en el
banco es:" + t);
        cliente1.imprimir();
        cliente2.imprimir();
        cliente3.imprimir();
    }

    public static void main(String[] ar) {
        Banco banco1=new Banco();
        banco1.operar();
        banco1.depositosTotales();
    }
}
```

Analicemos la implementación del problema.
Los atributos de una clase normalmente son privados para que no se tenga acceso directamente desde otra clase, los atributos son modificados por los métodos de la misma clase:

```
    private String nombre;
    private int monto;
```

El constructor recibe como parámetro el nombre del cliente y lo almacena en el atributo respectivo e inicializa el atributo monto en cero:

```
    public Cliente(String nom) {
        nombre=nom;
        monto=0;
    }
```

Los métodos depositar y extraer actualizan el atributo monto con el dinero que llega como parámetro (para simplificar el problema no hemos validado que cuando se extrae dinero el atributo monto quede con un valor negativo):

```
    public void depositar(int m) {
        monto=monto+m;
    }

    public void extraer(int m) {
        monto=monto-m;
    }
```

El método retornarMonto tiene por objetivo comunicar al Banco la cantidad de dinero que tiene el cliente (recordemos que como el atributo monto es privado de la clase, debemos tener un método que lo retorne):

```
    public int retornarMonto() {
        return monto;
    }
```

Por último el método imprimir muestra el nombre y el monto de dinero del cliente:

```
    public void imprimir() {
        System.out.println(nombre+" tiene depositado la
suma de "+monto);
    }
```

Como podemos observar la clase Cliente no tiene función main. Entonces donde definimos objetos de la clase Cliente?
La respuesta a esta pregunta es que en la clase Banco definimos tres objetos de la clase Cliente.
Veamos ahora la clase Banco que requiere la colaboración de la clase Cliente.
Primero definimos tres atributos de tipo Cliente.

```
public class Banco {
    private Cliente cliente1,cliente2,cliente3;
```

En el constructor creamos los tres objetos (cada vez que creamos un objeto de la clase Cliente debemos pasar a su constructor el nombre del cliente, recordemos que su monto de depósito se inicializa con cero):

```
    public Banco() {
        cliente1=new Cliente("Juan");
        cliente2=new Cliente("Ana");
        cliente3=new Cliente("Pedro");
    }
```

El método operar del banco (llamamos a los métodos depositar y extraer de los clientes):

```
    public void operar() {
        cliente1.depositar (100);
        cliente2.depositar (150);
        cliente3.depositar (200);
        cliente3.extraer (150);
    }
```

El método depositosTotales obtiene el monto depositado de cada uno de los tres clientes, procede a mostrarlos y llama al método imprimir de cada cliente para poder mostrar el nombre y depósito:

```
    public void depositosTotales ()
    {
        int t = cliente1.retornarMonto () +
cliente2.retornarMonto () + cliente3.retornarMonto ();
        System.out.println ("El total de dinero en el
banco es:" + t);
        cliente1.imprimir();
        cliente2.imprimir();
        cliente3.imprimir();
    }
```

Por último en la main definimos un objeto de la clase Banco (la clase Banco es la clase principal en nuestro problema):

```
    public static void main(String[] ar) {
        Banco banco1=new Banco();
        banco1.operar();
        banco1.depositosTotales();
    }
```

Problema 2

Plantear un programa que permita jugar a los dados. Las reglas de juego son: se tiran tres dados si los tres salen con el mismo valor mostrar un mensaje que "gano", sino "perdió".

Lo primero que hacemos es identificar las clases:
Podemos identificar la clase Dado y la clase JuegoDeDados.

Luego los atributos y los métodos de cada clase:
Dado
- atributos
  - valor
- métodos
  - tirar
  - imprimir
  - retornarValor

JuegoDeDados
- atributos
  - 3 Dado (3 objetos de la clase Dado)
- métodos
  - constructor
  - jugar

Creamos un proyecto en Eclipse llamado: proyecjuegodedados y dentro del proyecto creamos dos clases llamadas: Dado y JuegoDeDados.

El código fuente de la clase Dado es:

```
public class Dado {
    private int valor;

    public void tirar() {
        valor=1+(int)(Math.random()*6);
    }

    public void imprimir() {
        System.out.println("El valor del dado
es:"+valor);
    }

    public int retornarValor() {
        return valor;
    }
}
```

El código fuente de la clase JuegoDeDado es:

```
public class JuegoDeDados {
    private Dado dado1,dado2,dado3;

    public JuegoDeDados() {
        dado1=new Dado();
        dado2=new Dado();
        dado3=new Dado();
    }

    public void jugar() {
        dado1.tirar();
        dado1.imprimir();
        dado2.tirar();
        dado2.imprimir();
        dado3.tirar();
        dado3.imprimir();
        if (dado1.retornarValor()==dado2.retornarValor()
      && dado1.retornarValor()==dado3.retornarValor()) {
            System.out.println("Ganó");
        } else {
            System.out.println("Perdió");
```

```
        }
    }

    public static void main(String[] ar){
        JuegoDeDados j=new JuegoDeDados();
        j.jugar();
    }
}
```

La clase Dado define el atributo "valor" donde almacenamos un valor aleatorio que representa el número que sale al tirarlo.

```
public class Dado {
    private int valor;
```

El método tirar almacena el valor aleatorio (para generar un valor aleatorio utilizamos el método random de la clase Math, el mismo genera un valor real comprendido entre 0 y 1, pero nunca 0 o 1. Puede ser un valor tan pequeño como 0.0001 o tan grando como 0.9999. Luego este valor generado multiplicado por 6 y antecediendo (int) obtenemos la parte entera de dicho producto):

```
    public void tirar() {
        valor=1+(int)(Math.random()*6);
    }
```

Como vemos le sumamos uno ya que el producto del valor aleatorio con seis puede generar números enteros entre 0 y 5.
El método imprimir de la clase Dado muestra por pantalla el valor del dado:

```
    public void imprimir() {
        System.out.println("El valor del dado
es:"+valor);
    }
```

Por último el método que retorna el valor del dado (se utiliza en la otra clase para ver si los tres dados generaron el mismo valor):

```
    public int retornarValor() {
        return valor;
    }
```

La clase JuegoDeDatos define tres atributos de la clase Dado (con esto decimos que la clase Dado colabora con la clase JuegoDeDados):

```
public class JuegoDeDados {
    private Dado dado1,dado2,dado3;
```

En el constructor procedemos a crear los tres objetos de la clase Dado:

```
    public JuegoDeDados() {
```

```
        dado1=new Dado();
        dado2=new Dado();
        dado3=new Dado();
    }
```

En el método jugar llamamos al método tirar de cada dado, pedimos que se imprima el valor generado y finalmente procedemos a verificar si se ganó o no:

```
    public void jugar() {
        dado1.tirar();
        dado1.imprimir();
        dado2.tirar();
        dado2.imprimir();
        dado3.tirar();
        dado3.imprimir();
        if (dado1.retornarValor()==dado2.retornarValor()
&& dado1.retornarValor()==dado3.retornarValor()) {
            System.out.println("Ganó");
        } else {
            System.out.println("Perdió");
        }
    }
```

En la main creamos solo un objeto de la clase principal (en este caso la clase principal es el JuegoDeDados):

```
    public static void main(String[] ar){
        JuegoDeDados j=new JuegoDeDados();
        j.jugar();
    }
```

Problema propuesto

Plantear una clase Club y otra clase Socio.
La clase Socio debe tener los siguientes atributos privados: nombre y la antigüedad en el club (en años). En el constructor pedir la carga del nombre y su antigüedad. La clase Club debe tener como atributos 3 objetos de la clase Socio. Definir una responsabilidad para imprimir el nombre del socio con mayor antigüedad en el club.

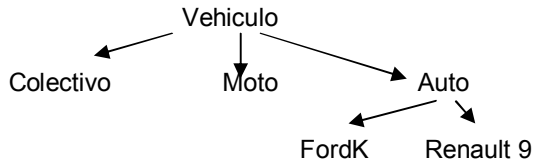
Herencia

Vimos en el concepto anterior que dos clases pueden estar relacionadas por la colaboración. Ahora veremos otro tipo de relaciones entre clases que es la Herencia.
La herencia significa que se pueden crear nuevas clases partiendo de clases existentes, que tendrá todas los atributos y los métodos de su 'superclase' o 'clase padre' y además se le podrán añadir otros atributos y métodos propios.

clase padre: Clase de la que desciende o deriva una clase. Las clases hijas (descendientes) heredan (incorporan) automáticamente los atributos y métodos de la la clase padre.

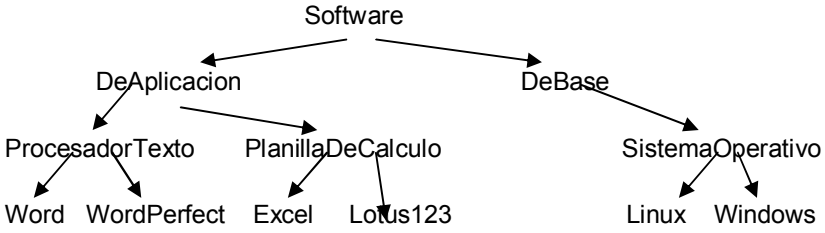
Subclase: Clase desciendiente de otra. Hereda automáticamente los atributos y métodos de su superclase. Es una especialización de otra clase. Admiten la definición de nuevos atributos y métodos para aumentar la especialización de la clase.

Veamos algunos ejemplos teóricos de herencia:
1) Imaginemos la clase Vehículo. Qué clases podrían derivar de ella?



Siempre hacia abajo en la jerarquía hay una especialización (las subclases añaden nuevos atributos y métodos)

2) Imaginemos la clase Software. Qué clases podrían derivar de ella?



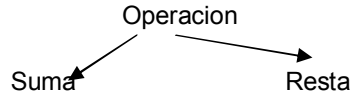
El primer tipo de relación que habíamos visto entre dos clases, es la de colaboración. Recordemos que es cuando una clase contiene un objeto de otra clase como atributo.
Cuando la relación entre dos clases es del tipo "...tiene un..." o "...es parte de...", no debemos implementar herencia. Estamos frente a una relación de colaboración de clases no de herencia.
Si tenemos una ClaseA y otra ClaseB y notamos que entre ellas existe una relacion de tipo "... tiene un...", no debe implementarse herencia sino declarar en la clase ClaseA un atributo de la clase ClaseB.
Por ejemplo: tenemos una clase Auto, una clase Rueda y una clase Volante. Vemos que la relación entre ellas es: Auto "...tiene 4..." Rueda, Volante "...es parte de..." Auto; pero la clase Auto no debe derivar de

Rueda ni Volante de Auto porque la relación no es de tipo-subtipo sino de colaboración. Debemos declarar en la clase Auto 4 atributos de tipo Rueda y 1 de tipo Volante.
Luego si vemos que dos clase responden a la pregunta ClaseA "..es un.." ClaseB es posible que haya una relación de herencia.
Por ejemplo:
Auto "es un" Vehiculo
Circulo "es una" Figura
Mouse "es un" DispositivoEntrada
Suma "es una" Operacion

Problema 1

Ahora plantearemos el primer problema utilizando herencia. Supongamos que necesitamos implementar dos clases que llamaremos Suma y Resta. Cada clase tiene como atributo valor1, valor2 y resultado.
Los métodos a definir son cargar1 (que inicializa el atributo valor1), carga2 (que inicializa el atributo valor2), operar (que en el caso de la clase "Suma" suma los dos atributos y en el caso de la clase "Resta" hace la diferencia entre valor1 y valor2, y otro método mostrarResultado.

Si analizamos ambas clases encontramos que muchos atributos y métodos son idénticos. En estos casos es bueno definir una clase padre que agrupe dichos atributos y responsabilidades comunes.
La relación de herencia que podemos disponer para este problema es:



Solamente el método operar es distinto para las clases Suma y Resta (esto hace que no lo podamos disponer en la clase Operacion), luego los métodos cargar1, cargar2 y mostrarResultado son idénticos a las dos clases, esto hace que podamos disponerlos en la clase Operacion. Lo mismo los atributos valor1, valor2 y resultado se definirán en la clase padre Operacion.
Crear un proyecto y luego crear cuatro clases llamadas: Operacion, Suma, Resta y Prueba

Archivo Operación.java:

```
import java.util.Scanner;
public class Operacion {
    protected Scanner teclado;
    protected int valor1;
    protected int valor2;
    protected int resultado;
    public Operacion() {
        teclado=new Scanner(System.in);
    }

    public void cargar1() {
```

```
        System.out.print("Ingrese el primer valor:");
        valor1=teclado.nextInt();
    }

    public void cargar2() {
        System.out.print("Ingrese el segundo valor:");
        valor2=teclado.nextInt();
    }

    public void mostrarResultado() {
        System.out.println(resultado);
    }
}
```

Archivo Suma.java:

```
public class Suma extends Operacion{
    void operar() {
        resultado=valor1+valor2;
    }
}
```

Archivo Resta.java:

```
public class Resta extends Operacion {
    public void operar(){
        resultado=valor1-valor2;
    }
}
```

Archivo Prueba.java:

```
public class Prueba {
    public static void main(String[] ar) {
        Suma suma1=new Suma();
        suma1.cargar1();
        suma1.cargar2();
        suma1.operar();
        System.out.print("El resultado de la suma es:");
        suma1.mostrarResultado();
        Resta resta1=new Resta();
        resta1.cargar1();
        resta1.cargar2();
        resta1.operar();
        System.out.print("El resultado de la resta
es:");
        resta1.mostrarResultado();
    }
}
```

La clase Operación define los cuatro atributos:

```
import java.util.Scanner;
public class Operacion {
    protected Scanner teclado;
    protected int valor1;
    protected int valor2;
    protected int resultado;
```

Ya veremos que definimos los atributos con este nuevo modificador de acceso (protected) para que la subclase tenga acceso a dichos atributos. Si los definimos private las subclases no pueden acceder a dichos atributos.
Los métodos de la clase Operacion son:

```
    public Operacion() {
        teclado=new Scanner(System.in);
    }

    public void cargar1() {
        System.out.print("Ingrese el primer valor:");
        valor1=teclado.nextInt();
    }

    public void cargar2() {
        System.out.print("Ingrese el segundo valor:");
        valor2=teclado.nextInt();
    }

    public void mostrarResultado() {
        System.out.println(resultado);
    }
```

Ahora veamos como es la sintaxis para indicar que una clase hereda de otra:

```
public class Suma extends Operacion{
```

Utilizamos la palabra clave extends y seguidamente el nombre de la clase padre (con esto estamos indicando que todos los métodos y atributos de la clase Operación son también métodos de la clase Suma.
Luego la característica que añade la clase Suma es el siguiente método:

```
    void operar() {
        resultado=valor1+valor2;
    }
```

El método operar puede acceder a los atributos heredados (siempre y cuando los mismos se declaren protected, en caso que sean private si bien lo hereda de la clase padre solo los pueden modificar métodos de dicha clase padre.
Ahora podemos decir que la clase Suma tiene cinco métodos (cuatro heredados y uno propio) y 3 atributos (todos heredados)
Luego en otra clase creamos un objeto de la clase Suma:

```
public class Prueba {
    public static void main(String[] ar) {
        Suma suma1=new Suma();
        suma1.cargar1();
        suma1.cargar2();
        suma1.operar();
        System.out.print("El resultado de la suma es:");
        suma1.mostrarResultado();
        Resta resta1=new Resta();
        resta1.cargar1();
        resta1.cargar2();
        resta1.operar();
        System.out.print("El resultado de la resta
es:");
        resta1.mostrarResultado();
    }
}
```

Podemos llamar tanto al método propio de la clase Suma "operar()" como a los métodos heredados. Quien utilice la clase Suma solo debe conocer que métodos públicos tiene (independientemente que pertenezcan a la clase Suma o a una clase superior)
La lógica es similar para declarar la clase Resta.
La clase Operación agrupa en este caso un conjunto de atributos y métodos comunes a un conjunto de subclases (Suma, Resta). No tiene sentido definir objetos de la clase Operacion.
El planteo de jerarquías de clases es una tarea compleja que requiere un perfecto entendimiento de todas las clases que intervienen en un problema, cuales son sus atributos y responsabilidades.

Problema propuesto

Confeccionar una clase Persona que tenga como atributos el nombre y la edad. Definir como responsabilidades un método que cargue los datos personales y otro que los imprima.
Plantear una segunda clase Empleado que herede de la clase Persona. Añadir un atributo sueldo y los métodos de cargar el sueldo e imprimir su sueldo.
Definir un objeto de la clase Persona y llamar a sus métodos. También crear un objeto de la clase Empleado y llamar a sus métodos.