

**TSU Software, Materia: Programación, Clave 50086, Semestre 17-P**  
**Practica 10. Herencia y Encapsulamiento.**

## 1. Objetivos

Los objetivos buscados en esta práctica son los siguientes:

- Identificar el concepto de herencia en un problema ordinario.
- Escribir superclases y subclases.
- Sobrescribir métodos de las superclases.

## 2. Ejemplo

Analizaremos un caso particular de operación en una institución bancaria, además de abrir cuentas en el banco se pueden contratar inversiones. Una **inversión** permite a los clientes obtener una rentabilidad por su dinero.

Una **inversión** se caracteriza por:

Estructura: titular, capital, plazo en días, tipo de interés

Comportamiento:

- *Liquidar el depósito*: una vez cumplido el plazo con lo que se devuelve al cliente el capital invertido más los intereses
- *Consultar los intereses*: producidos al final del periodo

```
public class Inversion {
    private Persona titular;
    private double capital;
    private int plazoDias;
    private double tipoInteres;

    public Inversion(Persona titular, double capital,
        int plazoDias, double tipoInteres) {
        this.titular = titular;
        this.capital = capital;
        this.plazoDias = plazoDias;
        this.tipoInteres = tipoInteres;
    }

    public double liquidar() {
        return getCapital() + getIntereses();
    }
    public double getIntereses() {
        return (plazoDias * tipoInteres * capital)/365;
    }
    public double getCapital() {...}
    public int getPlazoDias() {...}
    public double getTipoInteres() {...}
    public Persona getTitular() {...}
}
```

Ahora bien existe un tipo de inversión que se denomina depósito estructurado. Un depósito estructurado es un tipo de depósito que se caracteriza por tener una parte del capital invertido a interés fijo y otra parte a interés variable

- Comparte las características de depósito
- Añade características nuevas

A continuación crearemos una clase de inversión estructurada, aprovechando las similitudes y particularidades entre ambas clases.

La InversionEstructurada hereda de Inversion:

```
public class InversionEstructurada extends Inversion {
    private double tipoInteresVariable;
    private double capitalVariable;

    public InversionEstructurada( ... ) { ... }

    public double getInteresesVariable() {
        return (getPlazoDias() * tipoInteresVariable * capitalVariable)/365;
    }

    public double getTipoInteresVariable() { ... }

    public void setTipoInteresVariable(double tipoInteresVariable){ ... }
    public double getCapitalVariable() { ... }
}
```

Depósito estructurado hereda todas las características de depósito:

- Hereda todos los **atributos** aunque no los vea porque se han definido como privados (Principio Ocultamiento de la Información)
- Puede utilizar todos los métodos heredados como si fueran propios (por ejemplo, *getPlazoDias*)

Añade nuevas características:

- Atributos: *capitalVariable*, *tipoInteresVariable*
- Métodos: *getCapitalVariable*, *setCapitalVariable*, *getTiopInteresVariable*.

El constructor de la clase hija refina el comportamiento del padre. En Java los constructores no se heredan.

La primera sentencia del constructor de la clase hija **SIEMPRE** es una llamada al constructor de la clase padre. La llamada al constructor del padre puede ser:

- Implícita:
  - Si se omite, se llamará implícitamente al constructor por defecto
  - Equivale a poner como primera sentencia *super()*;
  - Si no existe el constructor por defecto en la clase padre dará un error en tiempo de compilación
- Explícita:
  - *super()*; o *super(a,b)*; o ...
  - Dependiendo de si el constructor al que invocamos tiene o no argumentos

```

public class InversionEstructurado extends Inversion {
    private double tipoInteresVariable;
    private double capitalVariable;

    public InversionEstructurado(Persona titular, double capital,
int plazoDias, double tipoInteres, double tipoInteresVariable,
double capitalVariable) {
        //Llamada explícita al constructor del padre
        super(titular, capital, plazoDias, tipoInteres);
        this.tipoInteresVariable = tipoInteresVariable;
        this.capitalVariable = capitalVariable;
    }
    ...
}

```

Ahora que ya hemos implementado una subclase de la clase inversión, debemos preguntarnos sobre los métodos heredados lo siguiente:

¿Son válidos todos los métodos heredados de la clase depósito para un depósito estructurado?

- *getCapital*: debe devolver la suma del capital fijo más el capital variable en el caso del depósito estructurado
- *getIntereses*: debe devolver la suma del interés fijo y el variable para un depósito estructurado

Al heredar es posible redefinir los métodos para adaptarlos a la semántica de la nueva clase. La redefinición reconcilia la reutilización con la extensibilidad. Es raro reutilizar una clase sin necesidad de hacer cambios.

Los **atributos** no se pueden redefinir, sólo se ocultan.:

- Si la clase hija define un atributo con el mismo nombre que un atributo de la clase padre, éste no está accesible.
- El campo de la superclase todavía existe pero no se puede acceder .
- 

Un **método** de la subclase con la misma signatura (nombre y parámetros) que un método de la superclase lo está redefiniendo.

- Si se cambia el tipo de los parámetros se está sobrecargando el método original

Una clase hija puede redefinir un método de la clase padre por dos motivos:

- *Reemplazo*: se sustituye completamente la implementación del método heredado manteniendo la semántica.
- *Refinamiento*: se añade nueva funcionalidad al comportamiento heredado.

En el refinamiento resulta útil invocar a la versión heredada del método.

La palabra reservada ***super*** se utiliza para invocar a un método de la clase padre.  
Se debe utilizar para el refinamiento de métodos

- No se tiene que utilizar para invocar a métodos heredados.

Se puede utilizar en el cuerpo de otros métodos:

- *Inversion*->*getCapital*: devuelve el capital fijo
- *InversionEstructurado*->*getCapital*: devuelve el capital fijo + capital variable
- En los métodos de *InversionEstructurado* habrá que determinar cuál de las dos versiones del método *getCapital* es la que necesitamos.

```
public class InversionEstructurado extends Inversion {  
  
    ...  
    @Override  
    public double getCapital() {  
        return super.getCapital() + getCapitalVariable();  
    }  
    @Override  
    public double getIntereses() {  
        return super.getIntereses()+getInteresesVariable();  
    }  
}
```

### 3. Ejercicios.

#### 3.1 Ejercicio 1

Mediante el análisis del problema presentado para su proyecto final:

- Identifiquen 2 superclases y al menos 2 subclases para cada superclase.
- Identifique los atributos y los métodos.
- Implemente las 6 clases, con los atributos y métodos identificados.
- Sobrecargue un método de la superclase para cada subclase.
- Realice una clase de prueba, donde haga uso de cada subclase, e identifique la diferencia de los comportamientos.