

# Ejercicios: Clases internas, anónimas y reflexión

Semana del 13 de Abril

1) **Clases anónimas/internas:** Crea una clase `Store`, que puede almacenar de manera secuencial, y permitiendo repeticiones elementos de cualquier tipo. La clase `Store` debe ser compatible con `List`, y debe admitir un filtro (objetos compatibles con la interfaz `IFilter`) que deben pasar los elementos que se desan añadir.

**Se pide:** el código de la clase `Store`, y cambiar el siguiente programa para que, en vez de usar una clase interna estática, use una clase anónima.

```
interface IFilter<T> { boolean check(T elem); }

public class AnonymousClass {
    static class FilterEven implements IFilter<Integer> {
        @Override public boolean check(Integer elem) { return elem%2==0; }
    }

    public static void main(String[] args) {
        Store<Integer> store = new Store<>(new FilterEven()); // change to anonymous
        store.add(1);
        store.add(2);
        System.out.println(store);
    }
}
```

Salida esperada:

```
[2]
```

2) **Reflexión/instanceof:** Frecuentemente, el uso del operador `instanceof` indica un error en el diseño. ¿Sucede eso en el siguiente programa? Reorganiza su código para eliminar ese operador y mejorar la extensibilidad del código.

```
abstract class BankCard {
    protected String number;
    protected double amount;

    public BankCard(String number, double amount) {
        this.number = number; this.amount = amount;
    }
    public void withdraw(double qty) { this.amount -= qty; } // error control omitted
}

class CreditCard extends BankCard {
    public CreditCard(String number, long amount) { super(number, amount); }
}

class DebitCard extends BankCard {
    public DebitCard(String number, long amount) { super(number, amount); }
}

public class Banking {
    public void charge(BankCard[] cards, int qty) {
        for (BankCard bc : cards) {
            if (bc instanceof CreditCard) bc.withdraw(0.1*qty+qty); // get rid of this!
            else if (bc instanceof DebitCard) { // get rid of this!
                if (qty>100) bc.withdraw(qty);
                else bc.withdraw(qty+1);
            }
        }
    }
}

/*...*/}
```

3) **Clases internas:** Las clases internas se utilizan a veces para crear clases auxiliares, que no son de interés para otra clase más que la clase contenedora. Así, en este ejercicio queremos crear una clase **Book**, que internamente contenga objetos de tipo **Chapter**, pero no queremos exponer esa clase al resto del programa.

**Se pide:** usando clases internas, completar el siguiente programa para que dé la salida de más abajo.

```
public class Books {  
    public static void main(String... args) {  
        Book b = new Book("The Book of why", "Judea Pearl");  
        b.addChapter("Introduction", 22). // internally creates a Chapter object  
            addChapter("The ladder of causation", 31).  
            addChapter("Paradoxes Galore", 30);  
        System.out.println(b);  
    }  
}
```

Salida esperada:

```
'The Book of why' by Judea Pearl, with chapters:  
[Introduction (22 pp), The ladder of causation (31 pp), Paradoxes Galore (30 pp)]
```