Test-Driven Development Desde Cero

Tema 2. Patrones de diseño y Refactorización

Guía pedagógica.

- Seleccione los ejercicios que consideres más interesantes o que más puedan aportarte, resuélvelos y envía tus soluciones a tu tutor para que los corrija.
- Puedes enviar las soluciones una a una o todas de golpe.
- Los ejercicios nos e evalúan, por lo que puedes enviar partes inconclusas o preguntar todas las dudas que tengas.
- Puedes resolver los ejercicios de cualquier manera que consideres adecuada, escribiendo código, explicando en un texto cómo lo resolverías, etc.
- Además de este documento, tienes otro con las soluciones de algunos ejercicios. Si te quedas atascado o no sabes cómo empezar consúltalo.

Ejercicio 01.

A la vista del siguiente código, identifique y aplique las refactorizaciones que considere más convenientes.

```
public class Persona {
    String numeroDeTelefono;

    public Persona(String numeroDeTelefono) {
        super();
        this.numeroDeTelefono = numeroDeTelefono;
    }

    public String getNumeroDeTelefono() {
        return numeroDeTelefono;
    }

    public void setNumeroDeTelefono(String numeroDeTelefono) {
        this.numeroDeTelefono = numeroDeTelefono;
    }

}
```

```java
public class Profesor extends Persona {

    String str;
    int edad;
    String numeroDeTelefono;
    List<Prestamo> prestamos;

    public Profesor(String numeroDeTelefono) {
        super(numeroDeTelefono);
    }

    public void printInformacionPersonal() {
        System.out.println("Nombre: "  + str);
        System.out.println("Edad: "  + edad);
        System.out.println("Teléfono: "  + numeroDeTelefono);

    }

    public void printTodaLaInformacion() {
        System.out.println("Nombre: "  + str);
        System.out.println("Edad: "  + edad);
        System.out.println("Teléfono: "  + this.numeroDeTelefono);
        for (Prestamo p: prestamos) {
            System.out.println(p);
        }
    }

}
```

Solución.

Vemos que *Profesor* hereda de *Persona* pero vuelve a definir el atributo *numeroDeTelefono*. Esto es una duplicación y, por tanto, refactorizamos quitando la declaración de este atributo de la clase *Profesor*.

Además, la clase *Profesor* posee un atributo llamado *str*. Este es un nombre poco descriptivo de la misión de este atributo, por ello refactorizamos y lo cambiamos por *nombre*.

Los métodos *print* de la clase *Profesor* contienen código duplicado. Podemos evitar esto haciendo que uno delos métodos llame al otro. En concreto el método *printTodaLaInformacion* (que, de paso, sería un nombre candidato a refactorizar ya que no dice mucho) llamará a *printInformacionPersonal*.

La clase *Profesor* no tiene métodos get/set, por lo que encapsulamos los atributos añadiendo dichos métodos.

Al final, la clase *Persona* no ha cambiado, pero la clase *Profesor* sí. El código resultante se muestra a continuación.

```java
public class Profesor extends Persona {
    String nombre;
    int edad;
    List<Prestamo> prestamos;

    public Profesor(String numeroDeTelefono) {
        super(numeroDeTelefono);
    }

    public void printInformacionPersonal() {
        System.out.println("Nombre: "  + nombre);
        System.out.println("Edad: "  + edad);
        System.out.println("Teléfono: "  + numeroDeTelefono);
    }

    public void printTodaLaInformacion() {
        this.printInformacionPersonal();
        for (Prestamo p: prestamos) {
            System.out.println(p);
        }
    }

}
```


```java
public class Profesor extends Persona {
    String nombre;
    int edad;
    List<Prestamo> prestamos;

    public Profesor(String numeroDeTelefono) {
        super(numeroDeTelefono);
    }

    public void printInformacionPersonal() {
        System.out.println("Nombre: "  + nombre);
        System.out.println("Edad: "  + edad);
        System.out.println("Teléfono: "  + numeroDeTelefono);
    }

    public void printTodaLaInformacion() {
        this.printInformacionPersonal();
        for (Prestamo p: prestamos) {
            System.out.println(p);
        }
    }

}
```

Ejercicio 02.

A continuación tiene un fragmento de un juego, en concreto la clase que se encarga de ver el movimiento que se desea hacer y mover las coordenadas del jugador en dicha dirección (considerando que el punto 0,0 está arriba a la izquierda).

```java
public class Game {

    Player p;

    //...

    public void movement(String m) {
        if (m.equalsIgnoreCase("Derecha")) {
            p.setX(p.getX()+1);
        }
        if (m.equalsIgnoreCase("Izquierda")) {
            p.setX(p.getX()-1);
        }
        if (m.equalsIgnoreCase("Arriba")) {
            p.setY(p.getY()-1);
        }
        if (m.equalsIgnoreCase("Abajo")) {
            p.setY(p.getY()+1);
        }

    }
}
```

```java
public class Player {

    int x, y;

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

}
```

Identifique qué refactorizaciones puede realizar (tanto las que vienen incluidas en la herramienta Eclipse como otras que no vinieran incluidas) en ambas clases.

Solución.

Podemos ver que la clase *Game* cambia los atributos de la clase *Player* a través de sus métodos get/set para mover al jugador. Esto no es una buna práctica ya que perdemos gran parte de los beneficios de utilizar clases y objetos. Por ello vamos a refactorizar para que la clase que sepa qué hay que cambiar al jugador cuando este se mueva se la propia clase *Player* y no *Game.*

Una posible solución e muestra a continuación.

```java
public class Game {

    Player p;

    //...

    public void movement(String m) {
        if (m.equalsIgnoreCase("Derecha")) {
            p.mueveDerecha();
        }
        if (m.equalsIgnoreCase("Izquierda")) {
            p.mueveIzquierda();
        }
        if (m.equalsIgnoreCase("Arriba")) {
            p.mueveArriba();
        }
        if (m.equalsIgnoreCase("Abajo")) {
            p.mueveAbajo();
        }

    }

}
```

```java
public class Player {
    int x, y;
    public int getX() {
        return x;
    }
    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

    public void mueveDerecha() {
        x+=1;
    }

    public void mueveIzquierda() {
        x-=1;
    }

    public void mueveArriba() {
        y -= 1;
    }

    public void mueveAbajo() {
        y += 1;
    }
}
```

Además, es aconsejable refactorizar para sustituir las cadenas de texto por constantes o, mejor aún , enumerados.