TP 7: Template Method y Adapter

Programación Orientada a Objetos II
Comisión 2
2º Cuatrimestre de 2024

Nicolás Salvanescki

Ejercicio 1

2. (a) El esqueleto del algoritmo se define en el template method. Dicho método se encuentra en la clase abstracta y solo define el esqueleto del algoritmo, es decir, delega en las subclases los pasos que varían en comportamiento.

   (b) Como poder, se puede, pero no es lo recomendable, ya que define el esqueleto del algoritmo, el cual debería ser común a todas las subclases.

   (c) Se pueden redefinir los pasos del algoritmo que son delegados en las subclases y también los métodos **hook**, ya que estos se encuentran vacíos en la clase abstracta y son creados con el objetivo de agregar comportamiento mediante la redefinición en las subclases.

   (d) Eso mismo, un **hook** es un método vacío en la clase abstracta que puede redefinirse en las subclases para, así, agregar comportamiento al template method desde las mismas.

4. La clase abstracta es ***Cuenta*** y las clases concretas son **CajaAhorro**, **CuentaCorriente** y **CuentaEmpresarial**.

   En el diagrama a continuación, puede notarse el template method **extraer** en color verde, los métodos **calcularComision** y **getDescripcion** de la clase abstracta que se redefinen en las clases concretas en color magenta. Y por último, el hook method **informar** puede notarse en color azul.

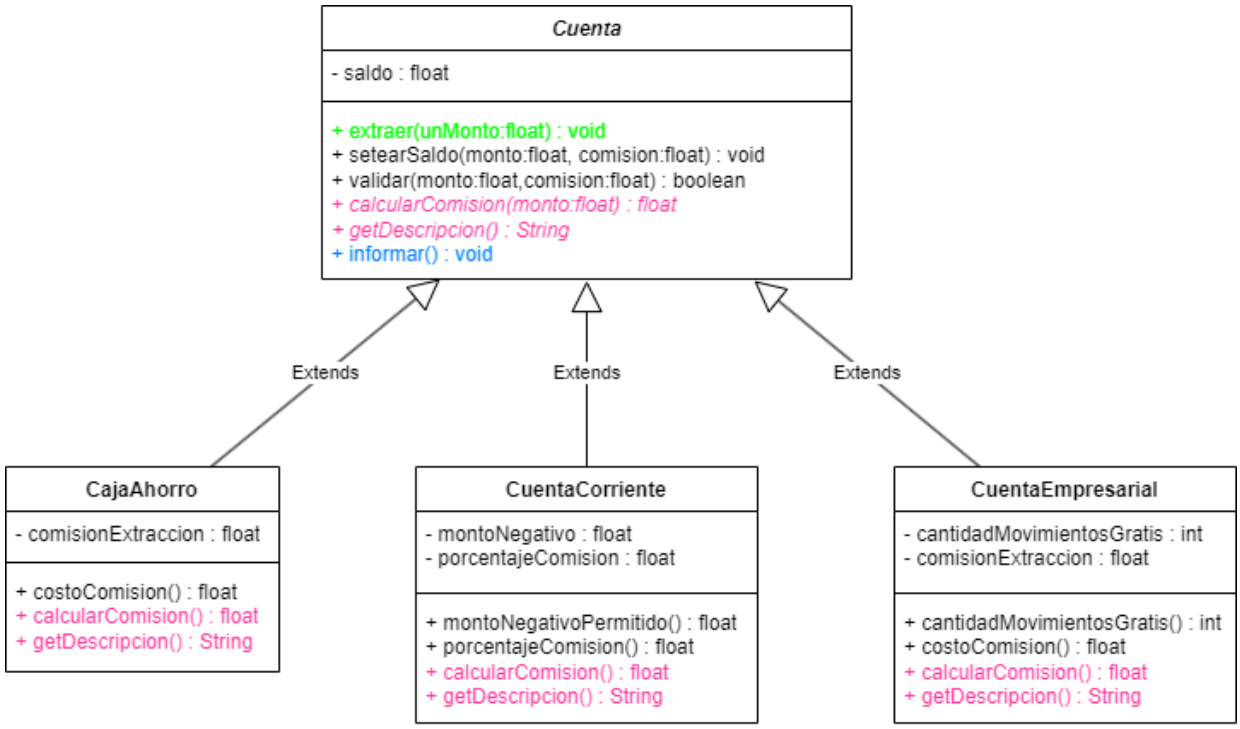

Figure 1: Diagrama de Clases UML Ejercicio 1

Ejercicio 2

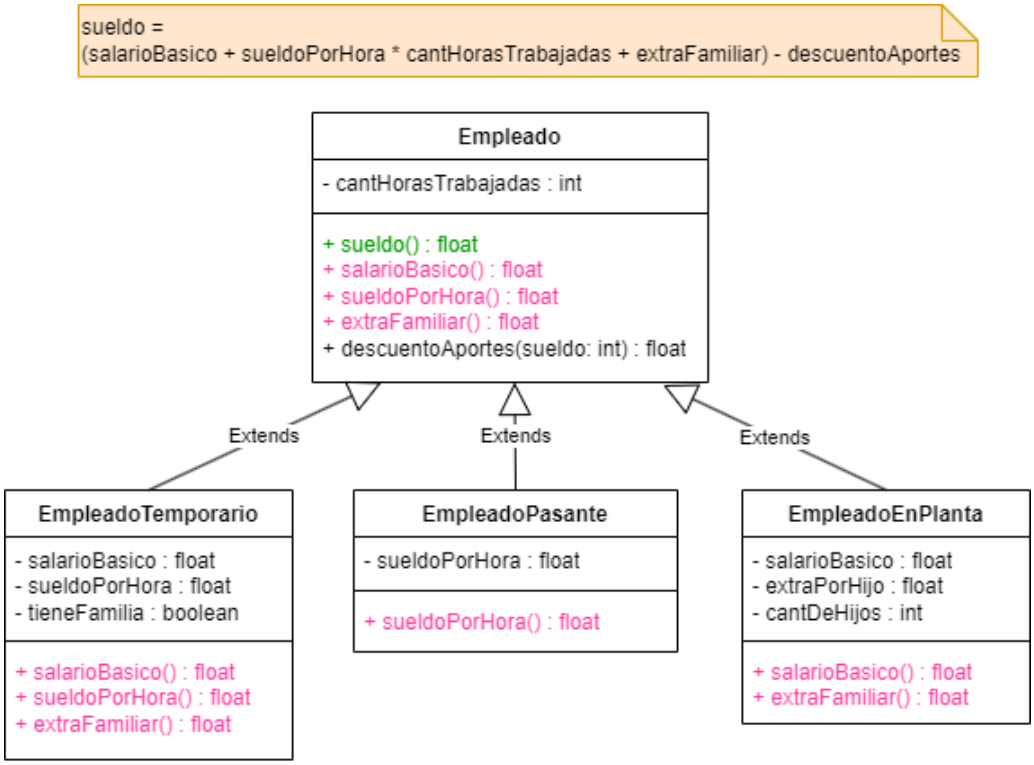

Figure 2: Diagrama de Clases UML Ejercicio 2

El template method es **sueldo** que por dentro tiene la siguiente lógica:

```
public float sueldo() {
    float sueldoBruto = this.salarioBasico()
                      + this.sueldoPorHora() * cantHorasTrabajadas
                      + this.extraFamiliar();

    return sueldoBruto - this.descuentoAportes(sueldoBruto);
}
```

Toda la lógica propia de cada tipo de empleado va a estar definiéndose en los métodos **salarioBasico**, **sueldoPorHora** y **extraFamiliar**. Si el tipo de empleado no necesita redefinir alguno de estos métodos, por default devuelven el número cero en la clase Empleado.

Ejercicio 3

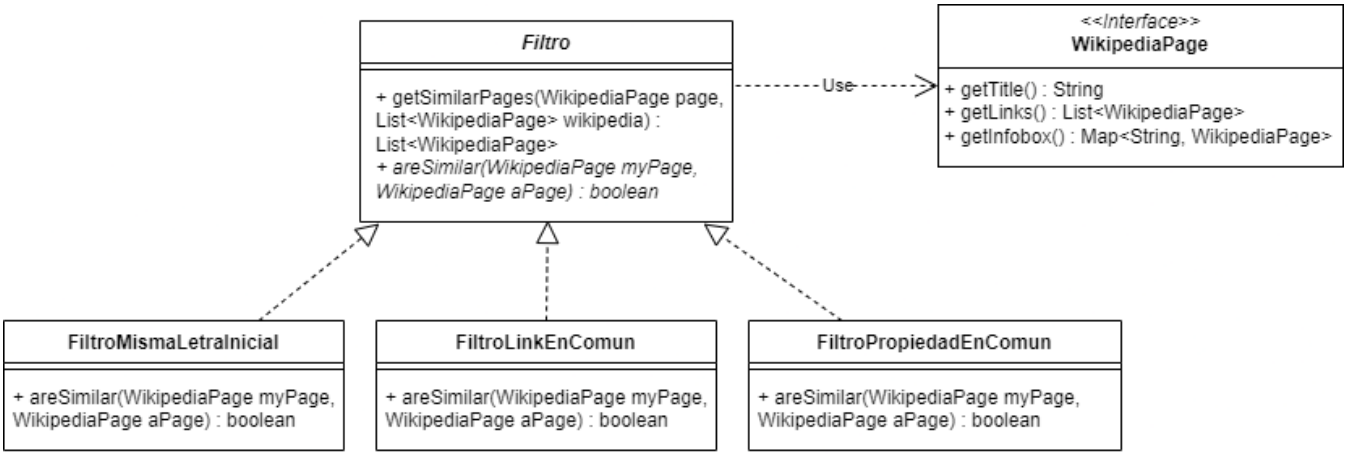

Figure 3: Diagrama de Clases UML Ejercicio 3

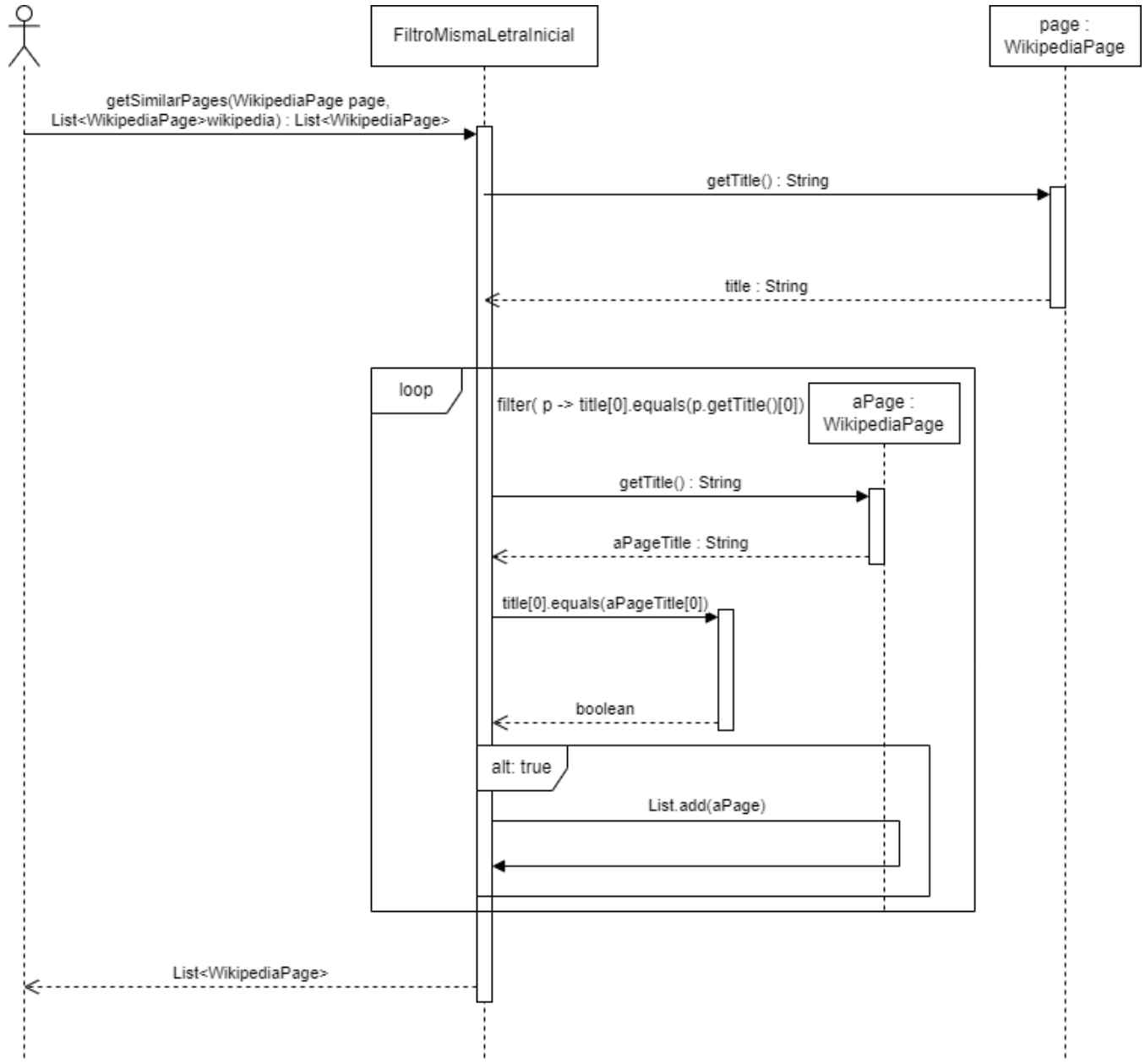

Figure 4: Diagrama de Secuencia MismaLetraInicial genérico

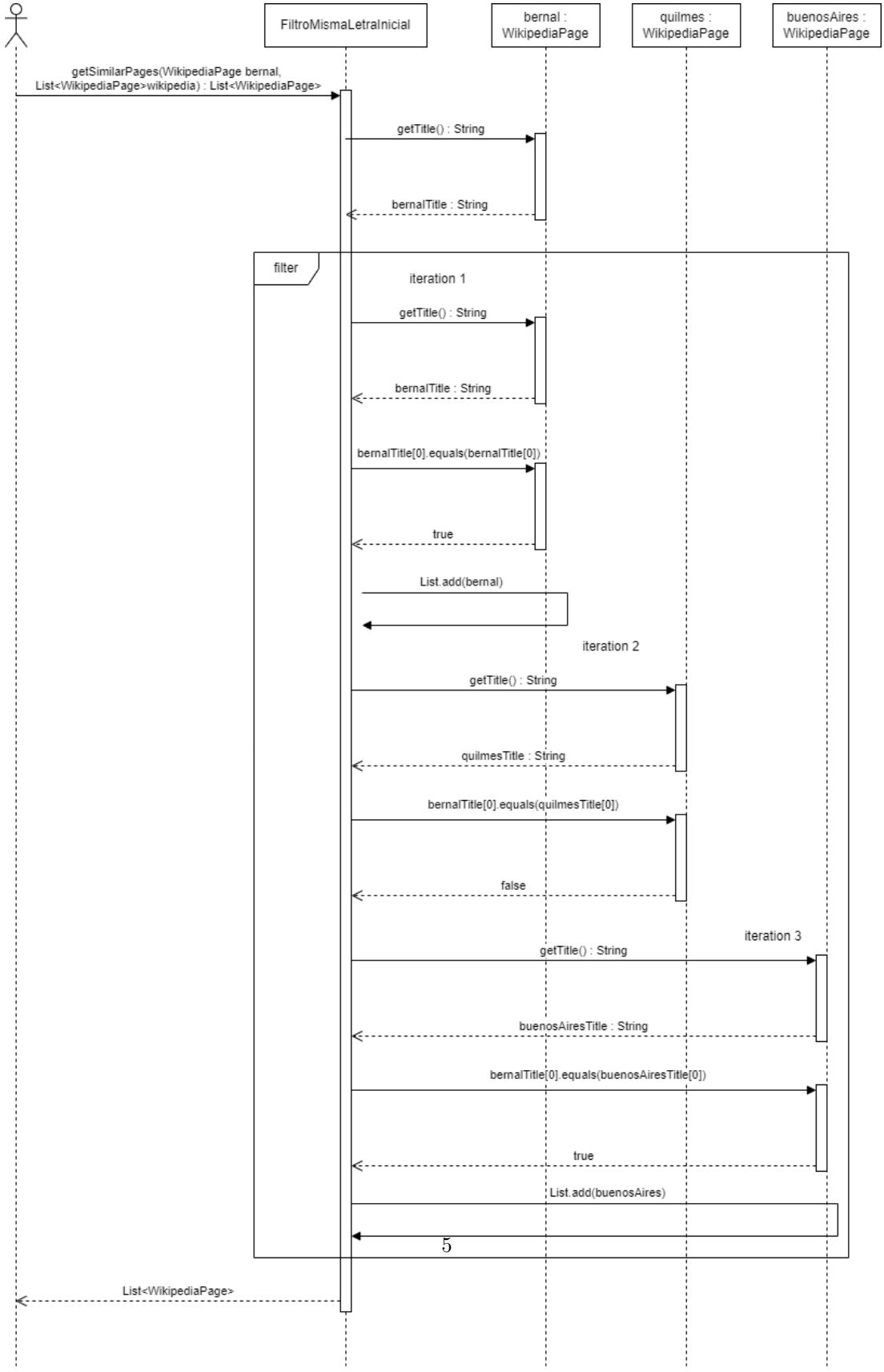

Figure 5: Diagrama de Secuencia MismaLetraInicial "Bernal"

Ejercicio 4

El template method es claramente **costo final**, mientras que las operaciones primitivas que pueden ser reemplazadas por las subclases son la operación abstracta ***esHoraPico*** y, aunque tenga implementación, **costoNeto**. De hecho, la clase LlamadaDescuento las sobreescribe ambas.

Operaciones concretas se pueden encontrar solo los getters y el constructor de la clase abstracta LlamadaTelefonica.

Métodos hooks, estrictamente hablando, no hay ninguno.

Ejercicio 5

1. Los cambios quedarían así:

```java
/*---------------CuentaBancaria----------------*/

public void extraer(int monto){
    if(this.validar(monto)){
        this.setSaldo(this.getSaldo() - monto);
        this.agregarMovimientos("Extraccion");
    }
}

public abstract boolean validar(int monto);

/*--------------CuentaCorriente---------------*/

@Override
public boolean validar(int monto){
    return this.getSaldo() + this.getDescubierto() >= monto;
}

/*----------------CajaDeAhorro----------------*/

@Override
public boolean validar(int monto){
    return this.getSaldo() >= monto && this.getLimite() >= monto;
}
```

2. Los elementos que indica Gamma et. al en el código resultante son:

   - Clase Abstracta: CuentaBancaria
   - Clases Concretas: CuentaCorriente y CajaDeAhorro
   - Template Method: extraer
   - Operaciones Primitivas: validar
   - Operaciones Concretas: Constructores, getTitular, getSaldo, setSaldo, agregarMovimientos, getDescubierto y getLimite.
   - Hook methods: no están implementados.

Ejercicio 6

1. Están los adaptadores de clase y los de objeto, lo que difiere entre ambos es la estructura.

   Los class adapters tienen a la clase Adapter que hereda interfaces de las clases Target y Adaptee al mismo tiempo. En lenguajes como java, que no soportan la herencia múltiple, no podría implementarse.

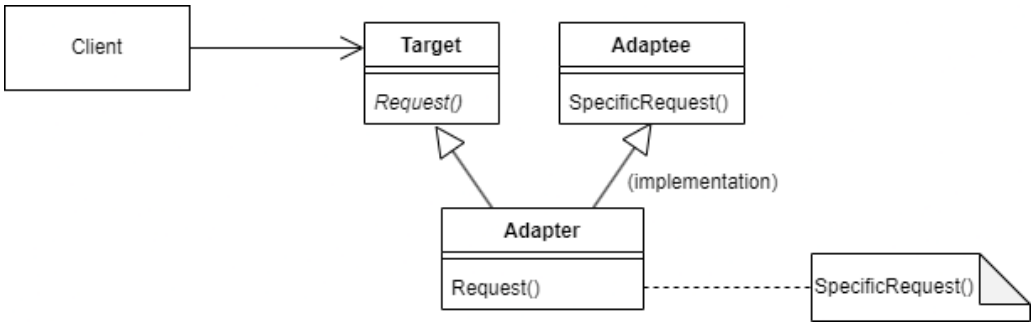

Figure 6: Class Adapter

   Los object adapters, por otro lado, utilizan el principio de composición de objetos. El Adapter implementa la interfaz de target y, a su vez, tienen al Adaptee como variable de instancia. Gracias a esto, es posible llamar a los métodos del adaptee. Se puede implementar en cualquier lenguaje que soporte programación orientada a objetos e interfaces.

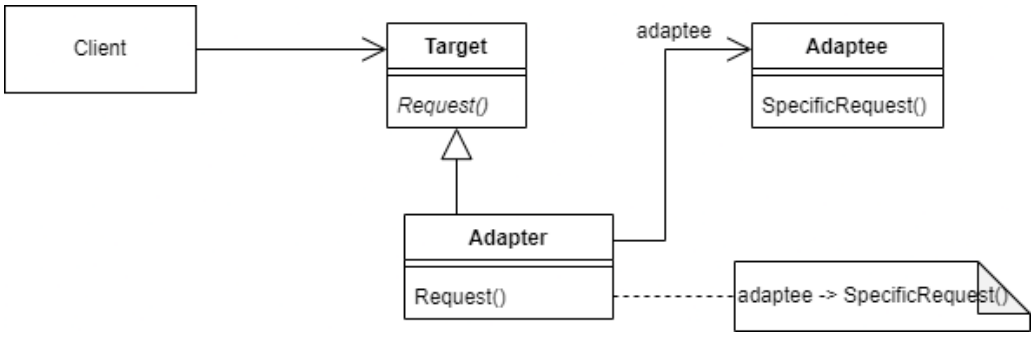

Figure 7: Object Adapter

2. La diferencia ya fue explicada, voy a centrarme en explicar que hace el adapter. La clase Adapter entiende el mensaje que el usuario le manda a Target, es decir, que implementa su interfaz y es capaz de utilizar la clase Adaptee para retornar lo que se desea de ella.

3. En java solo se puede implementar el Adapter de Objeto. Sin embargo, en algunos casos muy específicos donde se necesita solo heredar de una clase, se puede implementar también Adapter de Clase.

4. La interfaz Enumeration entiende los mensajes hasMoreElements, que indica si siguen habiendo elementos por delante, y nextElement, que retorna el próximo elemento. Esta interfaz permite al usuario recorrer los elementos

de la clase Vector. En este caso, el Target es Enumeration y el Adaptee es Vector. Cualquier clase que nosotros creemos y que implemente Enumeration sería un Adapter de Objeto.

5. ArrayList puede ser recorrida por la interfaz Iterator, al igual que Vector puede serlo con Enumeration. La diferencia radica en que Iterator también entiende el mensaje remove, el cual borra el último elemento devuelto por el iterador en la ArrayList. Los roles son los mismos, Iterator es el Target y ArrayList el Adaptee.

6. El código de la implementación puede encontrarse, con sus respectivos tests, en el paquete ar.edu.unq.po2.tp7.ej6

Ejercicio 7

El código está en ar.edu.unq.po2.tp7.ej7

Ejercicio 8

Todos los puntos fueron hechos en el ejercicio 6 de este Trabajo Práctico.