Memoria de la práctica 1 de Desarollo de Software

Ana Isabel Mena Meseguer
Jose Luis Parra Azor
Elena Vallejo Ruiz
Alejandro Ruiz Salazar

1 Ejercicio 1

Para la práctica, partimos de este diagrama UML, que a su vez hemos ido completando conforme íbamos creando las clases y las íbamos uniendo.

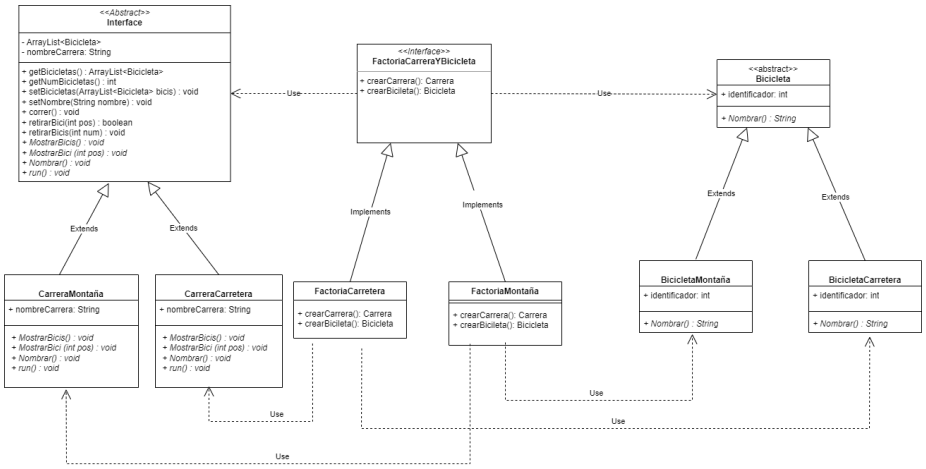

Figure 1: UML usado para los ejercicio 1 y 2

FactoriaCarreraYBicicleta es una interfaz, de las que heredan FactoriaCarrera y FactoriaMontaña, y Carrera y Bicicleta son clases abstractas de las que heredan CarreraMontaña y CarreraCarretera, y BicicletaMontaña y BicicletaCarretera respectivamente. Nuestro main pide un número por pantalla para crear los ArrayList de bicicletas tanto de montaña como de carretera y crea las carreras correspondientes con los ArrayList. Luego llamamos a los métodos start de thread que invoca a los métodos run que hemos creado para CarreraMontaña y CarreraCarretera.

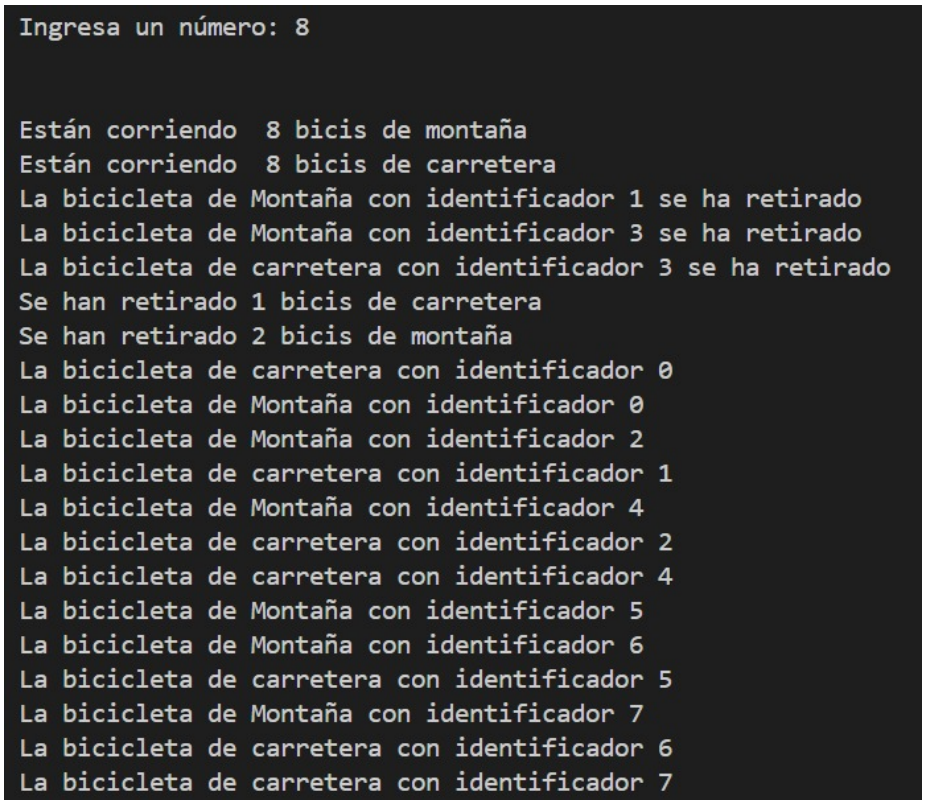

Figure 2: Salida por pantalla del ejercicio 1

Estos métodos lo que hacen es retirar el porcentaje de bicicletas que corresponda con el método retirarBicis y luego muestra por pantalla las bicicletas que quedan con el método correr. Usando start lo que conseguimos es que los dos métodos run de CarreraMontaña y CarreraCarretera se ejecuten de manera concurrente (mediante hebras).

Listing 1: Método run de CarreraMontana

```
1  @Override
2      public void run() {
3          System.out.println("Estan corriendo  "+
       getNumBicicletas() + " bicis de montana ");
4          int retirar = (int) Math.round(getNumBicicletas() *
       0.20);
5          retirarBicis(retirar);
6          System.out.println("Se han retirado "+ retirar + "
       bicis de montana ");
7          correr();
8      }
```

Listing 2: Método run de CarreraCarretera

```
@Override
    public void run() {
        System.out.println("Estan corriendo  "+
    getNumBicicletas() + " bicis de carretera ");
        int retirar = (int) Math.round(getNumBicicletas() *
    0.10);
        retirarBicis(retirar);
        System.out.println("Se han retirado "+ retirar + "
    bicis de carretera ");
        correr();
    }
```

2 Ejercicio 2

Para el ejercicio 2 lo que hemos tenido que hacer ha sido, en esencia, lo mismo que para el ejercicio 1. Al no poder usar interfaces en python, las hemos sustituido por clases abstractas. También tenemos que tener cuidado ya que no podemos usar como tipo la clase abstracta padre (a diferencia de java, donde se podía usar como tipo para instanciar clases hija), por lo que llamamos directamente a la clase de la cual queremos instanciar nuestros objetos. Finalmente, para implementar el patrón de diseño prototipo, hemos clonado las bicicletas cambiando luego su identificador.

Listing 3: main.py

```
import copy
from FactoriaMontana import FactoriaMontana
from FactoriaCarretera import FactoriaCarretera

if __name__ == "__main__":
    factoriaMontana = FactoriaMontana()
    bicicletasMontana = []
    factoriaCarretera = FactoriaCarretera()
    bicicletasCarretera = []

    # Creacion de la bicicleta de montana original
    bicicleta = factoriaMontana.crear_bicicleta(1)
    bicicletasMontana.append(bicicleta)

    # Inclusion del resto de bicicletas a partir de una copia
    de la original
    for i in range(9):
        bicicleta_aux = copy.deepcopy(bicicleta)
        bicicleta_aux.set_id(i + 2)
        bicicletasMontana.append(bicicleta_aux)
    carreraMontana = factoriaMontana.crear_carrera("Montana",
    bicicletasMontana)
```

```python

    # Creacion de la bicicleta de carretera original
    bicicleta = factoriaCarretera.crear_bicicleta(1)
    bicicletasCarretera.append(bicicleta)

    # Inclusion del resto de bicicletas a partir de una copia de la original
    for i in range(9):
        bicicleta_aux = copy.deepcopy(bicicleta)
        bicicleta_aux.set_id(i + 2)
        bicicletasCarretera.append(bicicleta_aux)
    carreraCarretera = factoriaCarretera.crear_carrera("Carretera", bicicletasCarretera)

    # Mostrar las bicicletas y las carreras
    print(carreraMontana.tipo())
    bicicletasMontana = carreraMontana.get_bicicletas()
    for bicicleta in bicicletasMontana:
        print(str(bicicleta.get_id()) + " " + bicicleta.tipo())

    print("\n" + carreraCarretera.tipo())
    bicicletasCarretera = carreraCarretera.get_bicicletas()
    for bicicleta in bicicletasCarretera:
        print(str(bicicleta.get_id()) + " " + bicicleta.tipo())
```

```
Carrera de montana
1 Bicicleta de montana
2 Bicicleta de montana
3 Bicicleta de montana
4 Bicicleta de montana
5 Bicicleta de montana
6 Bicicleta de montana
7 Bicicleta de montana
8 Bicicleta de montana
9 Bicicleta de montana
10 Bicicleta de montana

Carrera de carretera
1 Bicicleta de carretera
2 Bicicleta de carretera
3 Bicicleta de carretera
4 Bicicleta de carretera
5 Bicicleta de carretera
6 Bicicleta de carretera
7 Bicicleta de carretera
8 Bicicleta de carretera
9 Bicicleta de carretera
10 Bicicleta de carretera
```

Figure 3: Salida por pantalla del ejercicio 2.

3 Ejercicio 3

Para este ejercicio hemos implementado el patrón composite dos veces. Nuestra idea trata sobre los elementos de una web, que se dividen en Página o Contenido, donde una pagina puede tener dentro mas páginas o contenido, y el contenido a su vez puede ser o texto o imágenes, donde un texto puede tener dentro mas textos o imágenes.

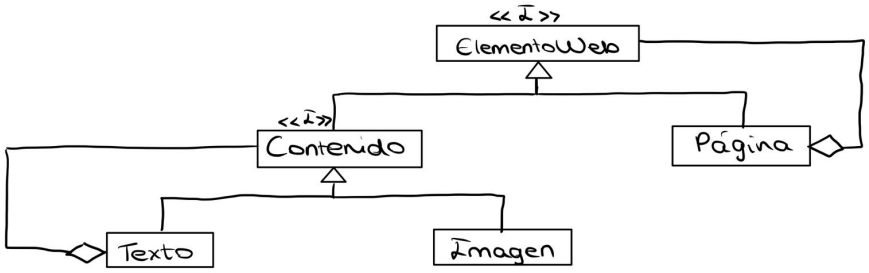

Figure 4: UML del ejercicio 3

Para probar que estuviese bien implementado, el main que hemos hecho crea varios objetos de cada clase. Primero hacemos algunas inserciones correctas como añadir una página dentro de otra página o un contenido dentro de una pagina, y luego hemos hecho unas cuantas inserciones erróneas para mostrar por pantalla mensajes de error al intentar realizarlas, como por ejemplo meter un texto dentro de una imagen, una imagen dentro de otra imagen o una pagina dentro de un contenido.

```
Se ha  insertado el elemento subpagina 1 en Portada
Se ha  insertado el elemento Parrafo 1 en subpagina 1
Se ha  insertado el elemento Parrafo 2 en subpagina 1
Se ha  insertado el contenido Imagen 1 en Parrafo 1
Se ha  insertado el contenido Imagen 2 en Parrafo 2
Se ha  insertado el contenido Parrafo 3 en Parrafo 2

No se ha podido insertar el contenido Imagen 3 en Imagen 2
No se ha podido insertar el elemento Portada en Parrafo 1
No se ha podido insertar el elemento Portada en Imagen 2
No se ha podido insertar el contenido Parrafo 1 en Imagen 3

Página Portada con identificador 1
  Elementos de la página Portada: subpagina 1
Página subpagina 1 con identificador 2
  Elementos de la página subpagina 1: Parrafo 1 Parrafo 2
Texto Parrafo 1 con identificador 11
  Elementos del texto Parrafo 1: Imagen 1
Texto Parrafo 2 con identificador 12
  Elementos del texto Parrafo 2: Imagen 2 Parrafo 3
```

Figure 5: Salida por pantalla del ejercicio 3

4 Ejercicio 4

Lo primero que hicimos para este ejercicio fue diseñar el diagrama UML con el patrón *filtros de intercepción*, cuyo resultado fue el siguiente:

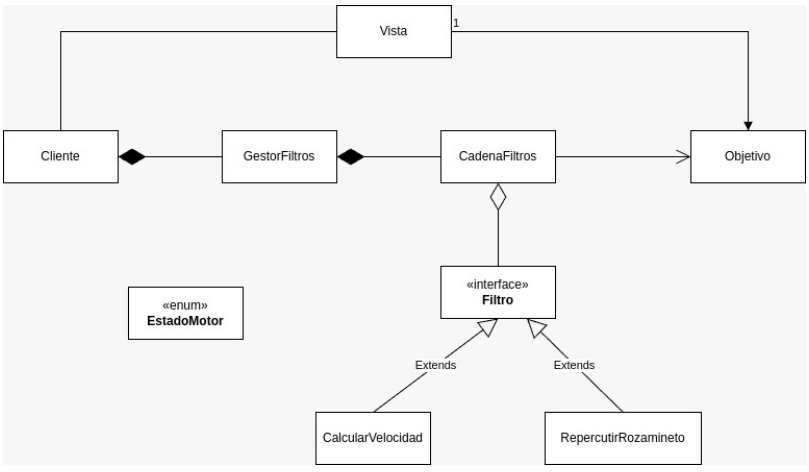

Figure 6: UML del ejercicio 4

Filtro como vemos es una interfaz de las que heredan las clases de *CalcularVelocidad* y *RepercutirRozamiento* como filtros.

Una vez terminado el diagrama, hemos realizado el diseño de la interfaz gráfica en NetBeans con JFrame debido a que en este entorno de desarrollo fue donde realizamos las prácticas de PDOO y es con el que más familiarizados estamos. Así es como ha quedado nuestra interfaz:

Figure 7: Interfaz Gráfica Ejercico 4

Hemos realizado este ejercicio en Java, el código de la implementación del filtro *CalcularVelocidad* es el siguiente:

Listing 4: CalcularVelocidad.java

```
1   public class CalcularVelocidad implements Filtro {
2
3    int incrementoVelocidad;
4
5    public CalcularVelocidad(){
6        incrementoVelocidad = 0;
7    }
8
9    public double ejecutar(double revoluciones, EstadoMotor
     estadoMotor){
10       if( ( estadoMotor == EstadoMotor.APAGADO ) || (
     estadoMotor == EstadoMotor.ENCENDIDO)){
11           incrementoVelocidad = 0;
12       }else if ( estadoMotor == EstadoMotor.FRENANDO){
13           incrementoVelocidad = -100;
14       }else{
15           incrementoVelocidad = +100;
16       }
17
18       double rev = revoluciones + incrementoVelocidad;
19
20       if( rev > 5000)
21           rev = 5000;
22
23       return rev;
24   }
25
26 }
```

Y el filtro *RepercutirRozamiento* quedaría de la siguiente manera:

Listing 5: RepercutirRozamiento.java

```
1  public class RepercutirRozamiento implements Filtro{
2      final int decrementoVelocidad = 1;
3
4      public RepercutirRozamiento(){}
5
6      public double ejecutar(double revoluciones, EstadoMotor
       estadoMotor){
7          if (revoluciones > decrementoVelocidad)
8              return revoluciones - decrementoVelocidad;
9          else
10             return 0;
11     }
12 }
```

Finalmente, al ejecutar nuestro programa, el primer paso es encender el coche. Una vez hecho esto, podemos proceder a acelerar o frenar, aunque nunca simultáneamente. Al acelerar, nuestras revoluciones aumentarán de 100 en 100, si bien se reducirá en 1 debido al rozamiento, lo que incrementará nuestra velocidad y recorrido. Al soltar el acelerador y mantener el motor encendido, las revoluciones comenzarán a disminuir de 1 en 1 hasta alcanzar 0, momento en el que el contador de revoluciones volverá a 0, debido a que estamos detenidos. Por otro lado, al frenar, nuestras revoluciones disminuirán de 100 en 100, en contraposición al efecto del acelerador. Aquí dejamos un ejemplo de uso:

Figure 8: Ejemplo de Uso Ejericicio4

5 Ejercicio 5

Para el ejercicio 5 hemos tenido que implementar 2 estrategias: Selenium y BeautifulSoup De esta manera en el main hemos incluido 2 while: uno que nos permitirá elegir qué empresa queremos (entre TESLA, AMAZON y NETFLIX) y otro para elegir la estrategia (Selenium o BeautifulSoup).

Listing 6: main.py

```
if __name__ == "__main__":
url = 'https://finance.yahoo.com/quote/AMZN'
opcion = 0
while (opcion != 1 and opcion != 2 and opcion != 3):
print("1. TESLA")
print("2. AMAZON")
print("3. NETFLIX")
opcion = int(input("\nSeleccione una empresa: "))
if (opcion == 1):
url = 'https://finance.yahoo.com/quote/TSLA'
elif (opcion == 2):
url = 'https://finance.yahoo.com/quote/AMZN'
else:
url = 'https://finance.yahoo.com/quote/NFLX'
estrategia = 0
while (estrategia != 1 and estrategia != 2):
print("\n\n1. BeautifulSoup")
print("2. Selenium")
estrategia = int(input("\nSeleccione una estrategia: "))
if (estrategia == 1):
context = Context(BeautifulSoupStrategy())
else:
context = Context(SeleniumStrategy())
scrape = context.scrape(url)
fich = "informacion.json"
close_value = context._strategy.find('td', 'data-test', '
    PREV_CLOSE-value')
print('Close Value:', close_value)
open_value = context._strategy.find('td', 'data-test', 'OPEN-
    value')
print('Open Value:', open_value)
volume_value = context._strategy.find('td', 'data-test', '
    TD_VOLUME-value')
print('Volume Value:', volume_value)
market_cap_value = context._strategy.find('td', 'data-test', '
    MARKET_CAP-value')
print('Market Cap Value:', market_cap_value)
data = {
'Close Value': close_value,
'Open Value': open_value,
'Volume Value': volume_value,
```

```
'Market Cap Value': market_cap_value
}
with open(fich, "w") as f:
json.dump(data, f, indent=4)
if (estrategia == 2):
context._strategy.quit()
print("Informacion scrapeada de forma correcta y guardada en",
    fich)
```

De esta manera una vez obtenidos los valores, los almacenaremos en el archivo json con la orden dump.

La clase scrape_strategy es una clase abstracta para las estrategias de scraping. A la hora de realizar la clase beautifulSoup implementamos la clase ScrapeStrategy definiendo las funciones de scrape y find: El método scrape toma una url como entrada y realiza un get utilizando la biblioteca requests. Después se verifica si este get fue exitoso (codigo 200). En caso de que sea así, utiliza BeautifulSoup para analizar el contenido HTML.

Listing 7: método scrape

```
    def scrape(self, url):
        response = requests.get(url)
        if response.status_code == 200:
                        self.soup = BeautifulSoup(response.
    content, 'html.parser')
        else:
                        return f'No se pudo recuperar la pagina
     web, status code: {response.status_code}'
```

El método find nos devuelve el valor que se encuentre en la página html del elemento que mandemos por parámetros.

Listing 8: método find

```
def find(self, elemento, atributo, valor):
variable = self.soup.find(elemento, {atributo: valor})
if variable:
return variable.text.strip()
else:
return 'Valor no encontrado'
```

Para implementar Selenium hemos tenido que utilizar un driver, utiliza Selenium para abrir el navegador web Chrome y acceder a la url. Se realiza una espera implícita para darle tiempo a los elementos dinámicos a cargar.

El método find utiliza WebDriverWait para esperar a que el elemento sea visible y luego utiliza find_element_by_xpath para encontrar el elemento a partir

de su XPath.

El método quit permite cerrar el driver despues de hacer los finds necesarios.

Listing 9: clase SeleniumStrategy

```
class SeleniumStrategy(ScrapeStrategy):
def __init__(self):
super().__init__()
self.selenium = None
self.driver = None
def scrape(self, url):
self.driver = webdriver.Chrome()
self.driver.implicitly_wait(30)
self.driver.get(url)
pass
def find(self, elemento, atributo, valor):
WebDriverWait(self.driver, 10).until(
EC.visibility_of_element_located((By.XPATH,
"//td[@data-test='PREV_CLOSE-value']"))
)
variable = self.driver.find_element_by_xpath(f"//{elemento}[@{
    atributo}='{valor}']")
text = variable.text
return text
def quit(self):
self.driver.quit()
```

```
1. TESLA
2. AMAZON
3. NETFLIX

Seleccione una empresa: 2


1. BeautifulSoup
2. Selenium

Seleccione una estrategia: 1
Close Value: 171.96
Open Value: 173.50
Volume Value: 34,908,647
Market Cap Value: 1.822T
Informacion scrapeada de forma correcta y guardada en informacion.json
```

Figure 9: Salida por pantalla del ejercicio 5 usando BeautifulSoup

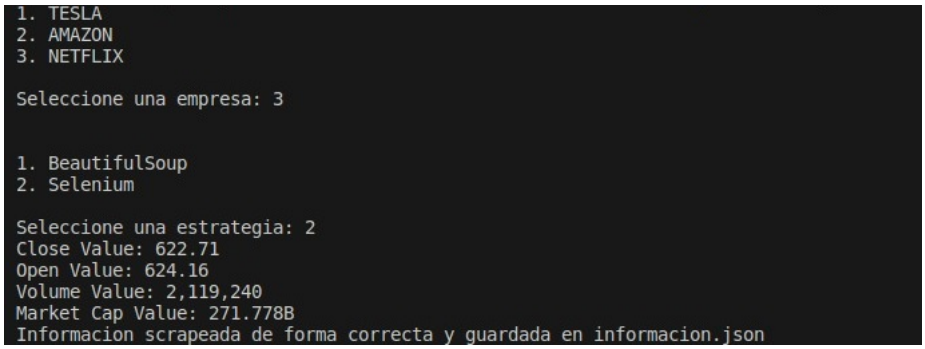

Figure 10: Salida por pantalla del ejercicio 5 usando Selenium

(Para Selenium quizá haya que ejecutar un par de veces)