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.

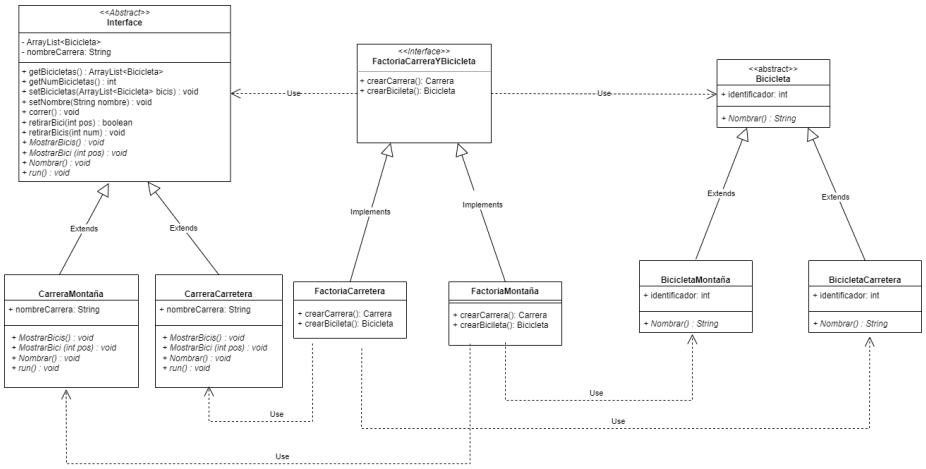

Figure 1: UML usado para los ejercicio 1 y 2

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.

```
Ingresa un número: 8


Están corriendo  8 bicis de montaña
Están corriendo  8 bicis de carretera
La bicicleta de Montaña con identificador 1 se ha retirado
La bicicleta de Montaña con identificador 3 se ha retirado
La bicicleta de carretera con identificador 3 se ha retirado
Se han retirado 1 bicis de carretera
Se han retirado 2 bicis de montaña
La bicicleta de carretera con identificador 0
La bicicleta de Montaña con identificador 0
La bicicleta de Montaña con identificador 2
La bicicleta de carretera con identificador 1
La bicicleta de Montaña con identificador 4
La bicicleta de carretera con identificador 2
La bicicleta de carretera con identificador 4
La bicicleta de Montaña con identificador 5
La bicicleta de Montaña con identificador 6
La bicicleta de carretera con identificador 5
La bicicleta de Montaña con identificador 7
La bicicleta de carretera con identificador 6
La bicicleta de carretera con identificador 7
```

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).

```java
@Override
public void run() {
    System.out.println("Están corriendo  "+ getNumBicicletas() + " bicis de montaña ");
    int retirar = (int) Math.round(getNumBicicletas() * 0.20);
    retirarBicis(retirar);
    System.out.println("Se han retirado "+ retirar + " bicis de montaña ");
    correr();
}
```

Figure 3: Método run de CarreraMontaña

```java
@Override
public void run() {
    System.out.println("Están 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();
}
```

Figure 4: Método run de CarreraCarretera

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 1: main.py

```python
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)

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

```
25
26      # Inclusion del resto de bicicletas a partir de una copia
        de la original
27      for i in range(9):
28          bicicleta_aux = copy.deepcopy(bicicleta)
29          bicicleta_aux.set_id(i + 2)
30          bicicletasCarretera.append(bicicleta_aux)
31      carreraCarretera = factoriaCarretera.crear_carrera("
        Carretera", bicicletasCarretera)
32
33      # Mostrar las bicicletas y las carreras
34      print(carreraMontana.tipo())
35      bicicletasMontana = carreraMontana.get_bicicletas()
36      for bicicleta in bicicletasMontana:
37          print(str(bicicleta.get_id()) + " " + bicicleta.tipo())
38
39      print("\n" + carreraCarretera.tipo())
40      bicicletasCarretera = carreraCarretera.get_bicicletas()
41      for bicicleta in bicicletasCarretera:
42          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 5: 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.

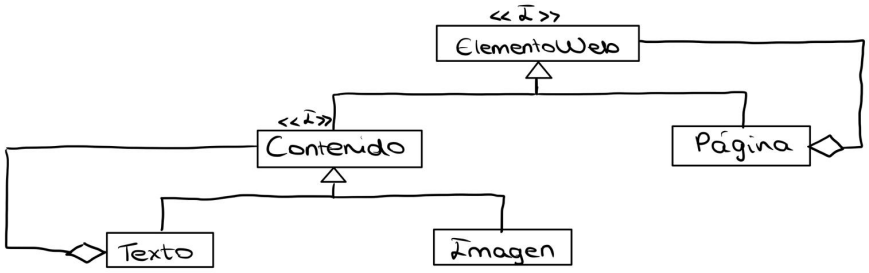

Figure 6: UML del ejercicio 3

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 7: Salida por pantalla del ejercicio 3