PROYECTO- Patrones de Diseño

25-26

Ingeniería del Software II

Gaizka Acedo

Agustín Beltrán de Heredia

Maddi López

0. Introducción

Para la realización de este proyecto hemos trabajado sobre el proyecto Rides, el utilizado en entregables previos (como las pruebas o refactorización).

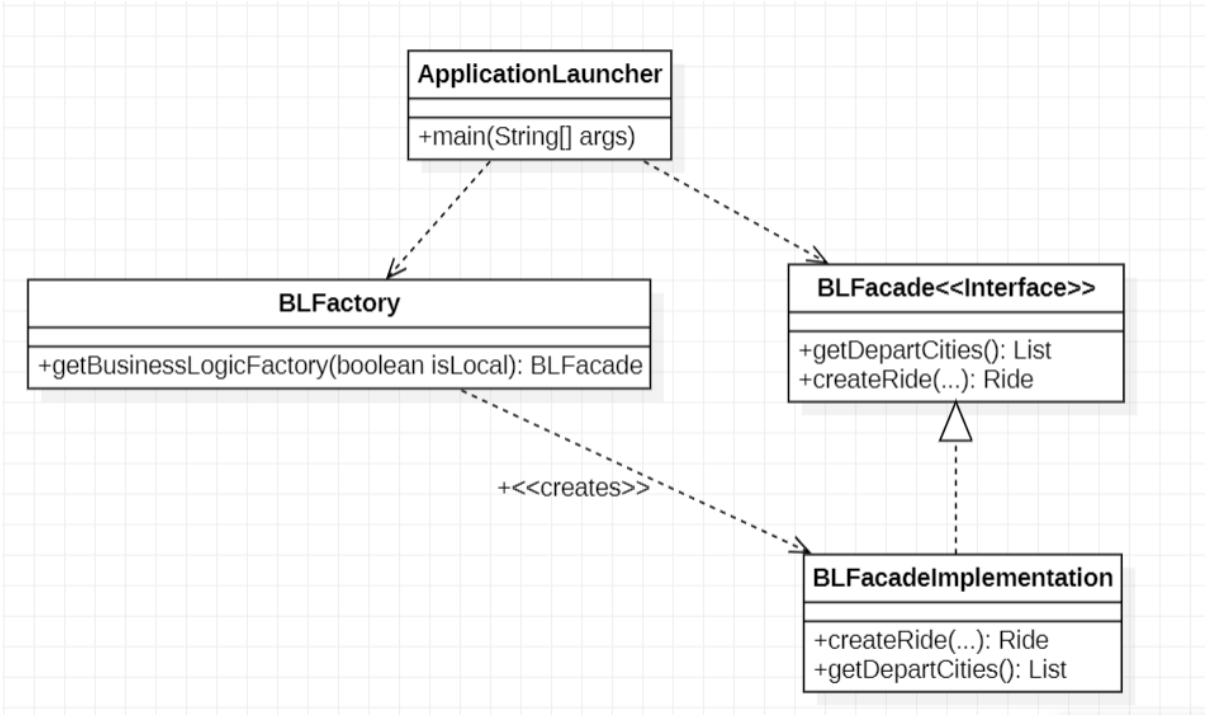
Importante mencionar que, para inicializar la BD, el valor de la variable *initialize* debe ser igual a *True*.

1. Patrón Factory Method

En este caso estas son las clases que juegan el papel de Creator, Product y ConcreteProduct:

- **Product:** `BLFacade`. Es la **interfaz** común que el `ApplicationLauncher` usará.
- **ConcreteProduct:** Son las implementaciones reales que la fábrica construirá.
    - `BLFacadeImplementation` (para la lógica local).
    - El objeto de servicio web remoto (que también implementa `BLFacade`).
- **Creator**:

1.1. UML indicando cambios realizados



1.2. Código modificado

Para realizar el patrón Factory Method estas 2 clases han sido modificadas de la siguiente manera:

- En la clase **ApplicacionLauncher**, el try-catch del método main se ha quedado de la siguiente manera:

```
try {
    BLFacade appFacadeInterface;
    UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
    boolean isLocal = c.isBusinessLogicLocal();

    appFacadeInterface=new BLFactory().getBusinessLogicFactory(isLocal);
    MainGUI.setBussinessLogic(appFacadeInterface);

}catch (Exception e) {
    a.jLabelWelcome.setText("Error: "+e.toString());
    a.jLabelWelcome.setForeground(Color.RED);

    System.out.println("Error in ApplicationLauncher: "+e.toString());
}
```

El cambio más significativo se encuentra dentro del bloque try, ya que, en este antes se realizaba toda la lógica para decidir cuál de los objetos se iba a crear. Ahora este método simplemente guarda el valor del BLFacade llamando al nuevo método creado en la clase BLFactory (que es el encargado de dicha acción), para después establecer la BusinessLogic del MainGUI mediante el setBussinessLogic.

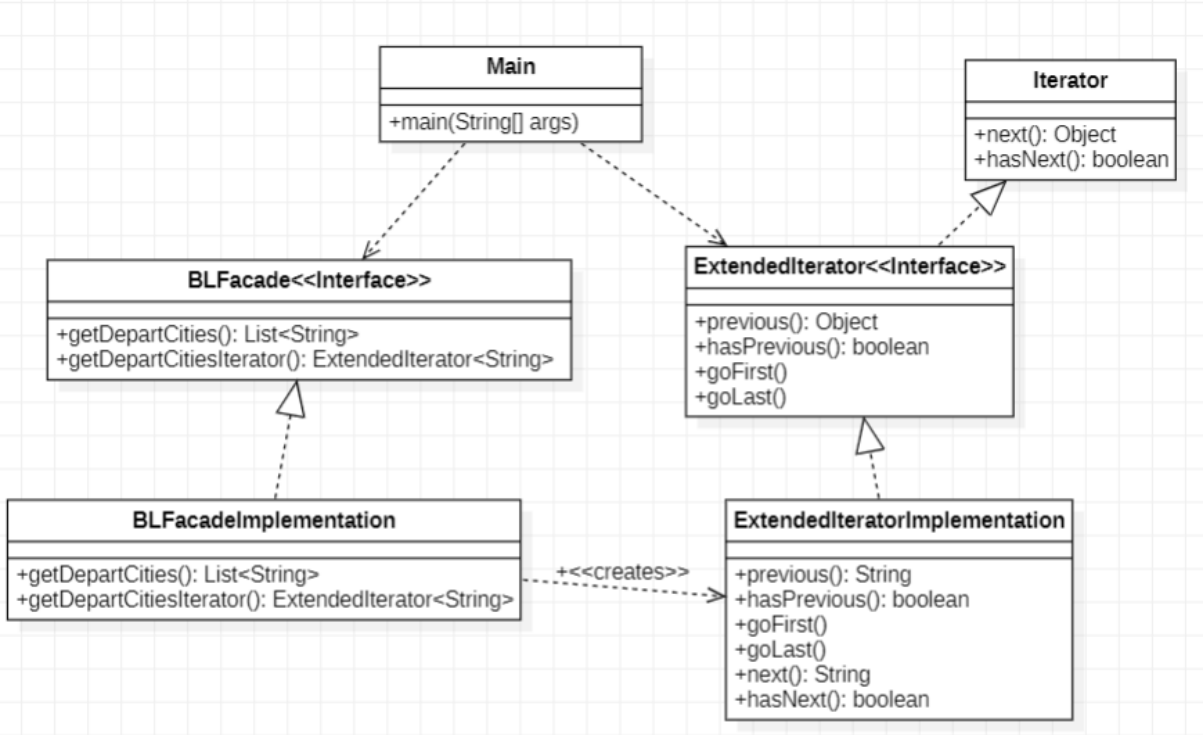
- En la clase **BLFactory** he creado el método getBusinessLogicFactory, donde se le pasa por parámetro si el Business Logic es local o no, y dependiendo de eso decide qué objeto se ha de crear:

```
public BLFacade getBusinessLogicFactory(boolean isLocal) {

    try {
        ConfigXML c=ConfigXML.getInstance();
        BLFacade appFacadeInterface;
        UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

        if (isLocal) {

            DataAccess da= new DataAccess();
            appFacadeInterface = new BLFacadeImplementation(da);
        }
        else { //If remote
            String serviceName= "http://"+c.getBusinessLogicNode() +":"+ c.getBusinessLogicPort()+"/ws/"+c.getBus

            URL url = new URL(serviceName);

            //1st argument refers to wsdl document above
            //2nd argument is service name, refer to wsdl document above
            QName qname = new QName("http://businessLogic/", "BLFacadeImplementationService");

            Service service = Service.create(url, qname);

            appFacadeInterface = service.getPort(BLFacade.class);
        }
        return appFacadeInterface;

    }catch (Exception e) {
        System.out.println("Error in BLFactory: "+e.toString());
        return null;
    }
}
```

Aquí se ve como dependiendo de si el valor de isLocal es True o False realiza una cosa u otra. Como tal, no he cambiado nada que no estuviese ya hecho en el try de la clase ApplicationLauncher antes de haberlo movido a este nuevo método. Lo único, en el catch, ya que este método devuelve un BLFacade he tenido que añadir “return null” en caso de error.

2. Patrón Iterator

2.1. UML indicando cambios realizados



2.2. Código modificado

Para realizar este patrón he creado y modificado varias clases:

- ExtendedIterator: Es la interfaz que define cómo recorrer la lista

```java
package iterator;

import java.util.Iterator;

public interface ExtendedIterator<Object> extends Iterator<Object> {
    //return    the actual  element and go  to  the previous
    public Object   previous();
    //true  if ther is  a   previous    element
    public boolean hasPrevious();
    //It    is  placed  in  the first   element
    public void goFirst();
    // It   is  placed  in  the last    element
    public void goLast();
}
```

- ExtendedIteratorImplementation: Es la clase que implementa la interfaz y maneja la lógica del recorrido

```java
package iterator;

import java.util.List;

public class ExtendedIteratorImplementation implements ExtendedIterator<String>{
    private List<String> ciudades;
    private int posicion;

    public ExtendedIteratorImplementation(List<String> ciudades) {
        this.ciudades = ciudades;
        this.posicion = 0;
    }

    //comprueba si quedan elementos por recorrer
    @Override
    public boolean hasNext() {
        return (posicion < ciudades.size());
    }

     //devuelve el elemento en la posicion actual y avanza su posicion
     @Override
     public String next() {
         String ciudad = ciudades.get(posicion);
         posicion +=1;
         return ciudad;
     }

     //Devuelve el elemento en la posición actual y retrocede la posición
     @Override
     public String previous() {
         String ciudad = ciudades.get(posicion);
         posicion -=1;
         return ciudad;
     }

    //Comprueba si es la primera posición o no. Si fuera position > 0, el bucle se detendría cuando el valor de
    //posicion fuera 0, y nunca mostraría el primer elemento de la lista.
    @Override
    public boolean hasPrevious() {
        return posicion >= 0;
    }

    //inicializa el valor de posicion a 0
    @Override
    public void goFirst() {
        posicion = 0;

    }

    //inicializa el valor posicion al ultimo elemento de la lista ciudades, es decir, a la ultima posicion
    @Override
    public void goLast() {
        posicion = ciudades.size()-1;
    }

}
```

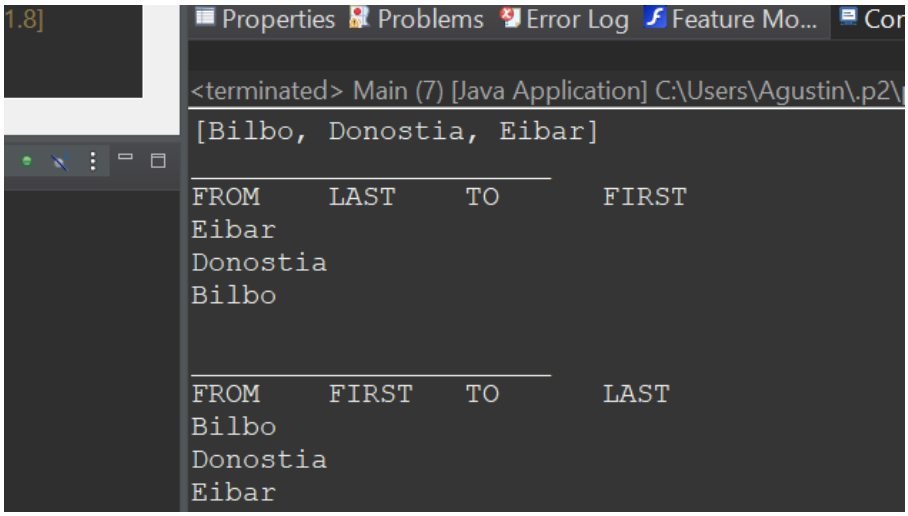
- Main: es igual que el método Main que se nos proporciona en el documento pdf del enunciado.
- BLFacade: he añadido el nuevo método que se pedía a la interfaz

```java
@WebMethod public List<String> getDepartCities();
@WebMethod public ExtendedIterator<String> getDepartCitiesIterator();
```

- BLFacadeImplementation: el único cambio ha sido la implementación del nuevo método añadido en la interfaz BLFacade. Como se pedía que los 2 métodos debían coexistir y que se evitase la reutilización de código he obtenido la lista llamando al método anterior para despúes poder crear una instancia del Iterator con dicha lista.

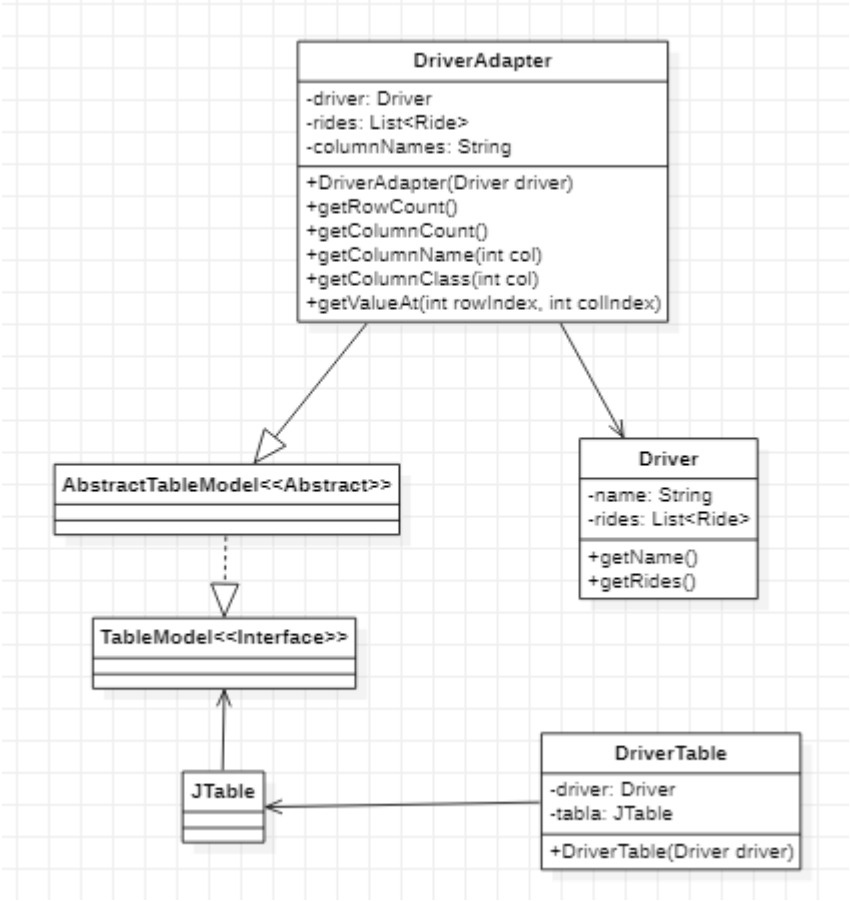
```
55      }
56      @WebMethod
57      public ExtendedIterator<String> getDepartCitiesIterator() {
58          List<String> ciudades = this.getDepartCities();
59          return new ExtendedIteratorImplementation(ciudades);
60      }
```

2.3. Captura de imagen que muestra su ejecución



3. Patrón Adapter

3.1. UML indicando cambios realizados



3.2. Código modificado

Creación de la clase DriverTable

```java
package gui;

import javax.swing.*;
import java.awt.*;
import domain.Driver;
import exceptions.PasswordDoesNotMatchException;
import businesslogic.BLFacade;
import businesslogic.BLFactory;


public class DriverTable extends JFrame {

    private Driver driver;
    private JTable tabla;

    public DriverTable(Driver driver) {
        super(driver.getName() + "'s rides ");
        this.setBounds(100, 100, 700, 200);
        this.driver = driver;

        DriverAdapter adapt = new DriverAdapter(driver);

        tabla = new JTable(adapt);

        tabla.setPreferredScrollableViewportSize(new Dimension(500, 70));
        JScrollPane scrollPane = new JScrollPane(tabla);
        getContentPane().add(scrollPane, BorderLayout.CENTER);

        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

public static void main(String[] args) {

        boolean isLocal = true;
        BLFacade blFacade = new BLFactory().getBusinessLogicFactory(isLocal);

        String conductorEmail = "urtzi@email.com";
        String conductorPassword = "1234";

        System.out.println("Buscando al conductor por email: " + conductorEmail);

        Driver d = null;

        try {
            d = blFacade.getDriverByEmail(conductorEmail, conductorPassword);

        } catch (PasswordDoesNotMatchException e) {
            System.out.println("Error: La contraseña no coincide.");

        } catch (Exception e) {
            System.out.println("Error: No se encontró el conductor o la contraseña es incorrecta.");
        }

        if (d != null) {
            System.out.println("Conductor encontrado. Creando tabla...");
            DriverTable dt = new DriverTable(d);
            dt.setVisible(true);
        } else {
            System.out.println("Error: Conductor no encontrado (revisa email y contraseña).");
        }
    }
}
```

Creación de la clase DriverAdapter

```java
package gui;

import javax.swing.table.AbstractTableModel;


public class DriverAdapter extends AbstractTableModel {

    private Driver driver;
    private List<Ride> rides;
    private final String[] columnNames = {"from", "to", "date", "places", "price"};

    public DriverAdapter(Driver driver) {
        this.driver = driver;
        this.rides = driver.getRides();
    }

    @Override
    public int getRowCount() {
        return rides.size();
    }

    @Override
    public int getColumnCount() {
        return columnNames.length;
    }

    @Override
    public String getColumnName(int col) {
        return columnNames[col];
    }

    @Override
    public Class<?> getColumnClass(int col) {
        switch (col) {
            case 2: return Date.class;
            case 3: return Integer.class;
            case 4: return Double.class;
            default: return String.class;
        }
    }

    @Override
    public Object getValueAt(int rowIndex, int colIndex) {
        Ride ride = rides.get(rowIndex);
        switch (colIndex) {
            case 0: return ride.getFrom();
            case 1: return ride.getTo();
            case 2: return ride.getDate();
            case 3: return ride.getnPlaces();
            case 4: return ride.getPrice();
            default: return null;
        }
    }
}
```

En la siguiente captura de pantalla, se ve que se ha añadido un nuevo Driver con los datos que se pedía en el enunciado, lo mismo con el coche, y con las rutas que se han creado.

```java
private List<Driver> createDrivers() {
    List<Driver> drivers = new ArrayList<>();
    Driver driver1 = new Driver("driver1@gmail.com","Aitor Fernandez", "123");
    Driver driver2 = new Driver("driver2@gmail.com","Ane Gaztañaga", "456");
    Driver driver3 = new Driver("driver3@gmail.com","Test driver", "789");
    Driver driver4 = new Driver("urtzi@email.com", "Urtzi", "1234");
    drivers.add(driver1);
    drivers.add(driver2);
    drivers.add(driver3);
    drivers.add(driver4);
    return drivers;


}

private void assignCarsToDrivers(List<Driver> drivers) {
    Driver driver1 = drivers.get(0);
    Driver driver2 = drivers.get(1);
    Driver driver3 = drivers.get(2);
    Driver driver4 = drivers.get(3);

    Car car1 = new Car("1234 ABC", 4, driver1, false);
    Car car2 = new Car("2345 DFG", 4, driver2, false);
    Car car3 = new Car("3456 HIJ", 6, driver3, true);
    Car car4 = new Car("4567 KLM", 9, driver2, true);
    Car car5 = new Car("5678 NÑO", 1, driver1, false);
    Car car6 = new Car("1111URT", 5, driver4, false);
    Car car7 = new Car("2222ZIU", 0, driver4, false);

    driver1.addCar(car1);
    driver1.addCar(car5);
    driver2.addCar(car2);
    driver2.addCar(car4);
    driver3.addCar(car3);
    driver4.addCar(car6);
    driver4.addCar(car7);
}

private void createRides(List<Driver> drivers, int month, int year) {
   String Bilbo = "Bilbo";
   String Donostia = "Donostia";
   String Gasteiz = "Gasteiz";
   String Madrid = "Madrid";
   String Irun = "Irun";
   String Barcelona = "Barcelona";
    Driver driver1 = drivers.get(0);
    Driver driver2 = drivers.get(1);
    Driver driver3 = drivers.get(2);
    Driver driver4 = drivers.get(3);

    driver1.addRide(Donostia, Bilbo, UtilDate.newDate(year, month, 15), 7, driver1.getCars().get(0));
    driver1.addRide(Donostia, Gasteiz, UtilDate.newDate(year, month, 6), 8, driver1.getCars().get(0));
    driver1.addRide(Bilbo, Donostia, UtilDate.newDate(year, month, 25), 4, driver1.getCars().get(1));
    driver1.addRide(Donostia, "Iruña", UtilDate.newDate(year, month, 7), 8, driver1.getCars().get(1));

    driver2.addRide(Donostia, Bilbo, UtilDate.newDate(year, month, 15), 3, driver2.getCars().get(0));
    driver2.addRide(Bilbo, Donostia, UtilDate.newDate(year, month, 25), 5, driver2.getCars().get(1));
    driver2.addRide("Eibar", Gasteiz, UtilDate.newDate(year, month, 6), 5, driver2.getCars().get(0));

    driver3.addRide(Bilbo, Donostia , UtilDate.newDate(year, month, 14), 3, driver3.getCars().get(0));

    driver4.addRide(Donostia, Irun, UtilDate.newDate(2024, 4, 30), 20.0f, driver4.getCars().get(0));
    driver4.addRide(Donostia, Madrid, UtilDate.newDate(2024, 4, 30), 2.0f, driver4.getCars().get(0));
    driver4.addRide(Madrid, Donostia, UtilDate.newDate(2024, 4, 10), 5.0f, driver4.getCars().get(0));
    driver4.addRide(Barcelona, Madrid, UtilDate.newDate(2024, 3, 20), 10.0f, driver4.getCars().get(1));
}
```

```java
private List<Driver> createDrivers() {
    List<Driver> drivers = new ArrayList<>();
    Driver driver1 = new Driver("driver1@gmail.com","Aitor Fernandez", "123");
    Driver driver2 = new Driver("driver2@gmail.com","Ane Gaztañaga", "456");
    Driver driver3 = new Driver("driver3@gmail.com","Test driver", "789");
    Driver driver4 = new Driver("urtzi@email.com", "Urtzi", "1234");
    drivers.add(driver1);
    drivers.add(driver2);
    drivers.add(driver3);
    drivers.add(driver4);
    return drivers;


}

private void assignCarsToDrivers(List<Driver> drivers) {
    Driver driver1 = drivers.get(0);
    Driver driver2 = drivers.get(1);
    Driver driver3 = drivers.get(2);
    Driver driver4 = drivers.get(3);

    Car car1 = new Car("1234 ABC", 4, driver1, false);
    Car car2 = new Car("2345 DFG", 4, driver2, false);
    Car car3 = new Car("3456 HIJ", 6, driver3, true);
    Car car4 = new Car("4567 KLM", 9, driver2, true);
    Car car5 = new Car("5678 NÑO", 1, driver1, false);
    Car car6 = new Car("1111URT", 5, driver4, false);
    Car car7 = new Car("2222ZIU", 0, driver4, false);

    driver1.addCar(car1);
    driver1.addCar(car5);
    driver2.addCar(car2);
    driver2.addCar(car4);
    driver3.addCar(car3);
    driver4.addCar(car6);
    driver4.addCar(car7);
}

private void createRides(List<Driver> drivers, int month, int year) {
   String Bilbo = "Bilbo";
   String Donostia = "Donostia";
   String Gasteiz = "Gasteiz";
   String Madrid = "Madrid";
   String Irun = "Irun";
   String Barcelona = "Barcelona";
   String Iruña = "Iruña";
   String Eibar = "Eibar";
    Driver driver1 = drivers.get(0);
    Driver driver2 = drivers.get(1);
    Driver driver3 = drivers.get(2);
    Driver driver4 = drivers.get(3);

    driver1.addRide(Donostia, Bilbo, UtilDate.newDate(year, month, 15), 7, driver1.getCars().get(0));
    driver1.addRide(Donostia, Gasteiz, UtilDate.newDate(year, month, 6), 8, driver1.getCars().get(0));
    driver1.addRide(Bilbo, Donostia, UtilDate.newDate(year, month, 25), 4, driver1.getCars().get(1));
    driver1.addRide(Donostia, Iruña, UtilDate.newDate(year, month, 7), 8, driver1.getCars().get(1));

    driver2.addRide(Donostia, Bilbo, UtilDate.newDate(year, month, 15), 3, driver2.getCars().get(0));
    driver2.addRide(Bilbo, Donostia, UtilDate.newDate(year, month, 25), 5, driver2.getCars().get(1));
    driver2.addRide(Eibar, Gasteiz, UtilDate.newDate(year, month, 6), 5, driver2.getCars().get(0));

    driver3.addRide(Bilbo, Donostia , UtilDate.newDate(year, month, 14), 3, driver3.getCars().get(0));

    driver4.addRide(Donostia, Irun, UtilDate.newDate(2024, 4, 30), 20.0f, driver4.getCars().get(0));
    driver4.addRide(Donostia, Madrid, UtilDate.newDate(2024, 4, 30), 2.0f, driver4.getCars().get(0));
    driver4.addRide(Madrid, Donostia, UtilDate.newDate(2024, 4, 10), 5.0f, driver4.getCars().get(0));
    driver4.addRide(Barcelona, Madrid, UtilDate.newDate(2024, 3, 20), 10.0f, driver4.getCars().get(1));
}
```

3.3. Captura de imagen que muestra su ejecución