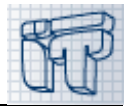


NOMBRE:
APELLIDOS:
NIA:
GRUPO:

Primer parcial

2ª Parte: Problemas (7 puntos sobre 10)

Duración: 80 minutos
Puntuación máxima: 7 puntos
Fecha: 15 de marzo de 2019

Instrucciones para el examen:

- No se permite el uso de libros o apuntes, ni tener teléfonos móviles u otros dispositivos electrónicos encendidos. Incumplir cualquiera de estas normas puede ser motivo de expulsión inmediata del examen.
- Rellena tus datos personales antes de comenzar a realizar el examen.

Ejercicio 1 (5 / 7 puntos)

La empresa WeMove ofrece un servicio de taxis autónomos en la ciudad de Madrid.

Apartado 1 (1,5 puntos)

WeMove requiere para la operación del servicio la implementación de la clase `AutonomousCar` que manejará los datos básicos del taxi, esto es, la matrícula, el número de plazas, el estado de ocupación (libre u ocupado) y los instantes de comienzo y finalización de cada viaje. Las clases hijas deben tener visibilidad de estas características del taxi y se deben implementar únicamente los métodos de acceso necesarios.

Asimismo, contiene los siguientes métodos:

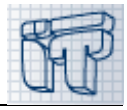
- `void bookCar()`: este método reserva el taxi (pasa a estado ocupado) e inicia el viaje de cara a la duración del mismo.
- `void releaseCar()`: este método fija el instante de finalización del viaje y libera el taxi (pasa a estado libre) para poder realizar otro viaje.
- `double calculateJourneyCost()`: este método calcula y devuelve el coste del viaje que dependerá del modelo de taxi elegido.

Apartado 2 (1 punto)

El servicio se va a lanzar inicialmente con un único modelo de vehículo eléctrico. Para ello, WeMove requiere que se implemente la clase `ElectricCar` teniendo en cuenta que es uno de los modelos de `AutonomousCar`. La clase `ElectricCar` ya puede calcular el coste de cada viaje teniendo en cuenta que tiene un coste fijo inicial de 5€ al que se suma un coste por minuto de 3€/minuto.

*Coste del viaje = Coste fijo inicial + Duración del viaje en minutos * coste/minuto*

Nota: el coste del viaje se debe implementar en el método `calculateJourneyCost`.



Apartado 3 (2 puntos)

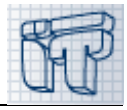
Con el objetivo de gestionar la flota de taxis se requiere implementar la clase Scheduler. Esta clase recibe en el constructor un array de taxis ya inicializados e inicialmente disponibles (libres) para usar en el servicio.

Dispone del método `public AutonomousCar bookCar(int minimumNumberOfSeats)` que recorre el array de taxis, busca el primero libre que disponga de las plazas mínimas requeridas, lo reserva y devuelve. Si no existe un taxi libre con el número de plazas necesarias o más, se debe devolver `null`.

Asimismo dispone del método `public void releaseCar(String registrationNumber)` que busca el taxi cuya matrícula recibe por parámetro y lo libera para que pueda ser reservado por otro cliente.

Apartado 4 (0,5 puntos)

Por último, se requiere la capacidad de generar la factura en modo texto para cualquier modelo de taxi con el formato "The amount of the bill is [cost]€" donde cost es el coste del viaje. Para ello se debe implementar la interfaz Billable que contiene el método `String generateBill()` y añadirla a la jerarquía de clases.



Ejercicio 2 (2 / 7 puntos)

El Ayuntamiento de la ciudad ha implantado un nuevo sistema de multas para los coches. Éstos se verán sujetos a restricciones en la circulación y el criterio que sigue el Ayuntamiento para multar a quienes no lo cumplan es el siguiente:

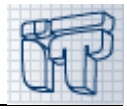
- Los días en los que la restricción al tráfico no esté activa, no se tramitará multa alguna y el importe de esta será de 0 para todos los coches, independientemente de los ocupantes en su interior.
- Los días de restricciones se diferenciará entre:

  a) Coches eléctricos, a los que no se les impondrá multa (multa = 0) y

  b) Coches no eléctricos, a los que sí se les aplicará una multa, dependiendo de los ocupantes del vehículo. Si el coche no eléctrico en días de restricción lleva 1 ocupante, la multa será de 40 euros. En cambio, si viajan en él más de un ocupante, con un máximo contemplado de 5 ocupantes, la multa será de 15 euros por cada uno.

El método `multarCoche`, de la clase `GestionMultas` (resumida a continuación) contiene el código necesario para la consulta del importe de una determinada multa, atendiendo a si existe restricción, a si el coche es eléctrico y al número de ocupantes del vehículo.

```
public class GestionMultas {
    /**
    * Método que devuelve el importe de la multa a un coche de
    * máx. 5 plazas
    * @param restr Existen (true) o no existen (false) restricciones
    * al tráfico
    * @param electr Coche eléctrico (true) o NO eléctrico (false)
    * @param ocup Ocupantes del coche
    * @exception IllegalArgumentException si algún parámetro no es
    * el esperado
    * @return Importe de la multa
    */
    public int multarCoche(boolean restr, boolean electr,
                int ocup) throws IllegalArgumentException{
        int multa = -1;
        if((!restr || electr) && (ocup >= 1) && (ocup <= 5)){
            multa = 0;
        }else if(ocup == 1){
            multa = 40;
        }else if((ocup >= 2) && (ocup <= 5)){
            multa = 15 * ocup;
        }else{
      throw new IllegalArgumentException("Operación fallida");
        }
        return multa;
    }
}
```



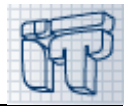
Apartado 1 (0.6 puntos)

Teniendo en cuenta el reglamento expuesto anteriormente, identifique las clases de equivalencia.

Apartado 2 (1.4 puntos)

Implemente la clase `GestionMultasTest` (Junit Test Case) para realizar una prueba de caja blanca que alcance un grado de cobertura del 100% de líneas y ramas del método `multarCoche` expuesto anteriormente.

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

public class GestionMultasTest {

	//Código a implementar en Apartado 2

}
```



Ejercicio 1. Rúbrica.

Criterios de corrección

Apartado 1: 1,5 puntos

- Clase abstracta con el método `calculateJourneyCost` abstracto: 0,3
- Atributos ok & protected y constructor correcto: 0,3
- Método bookCar: 0,3
- Método releaseCar: 0,3
- Métodos de acceso necesarios: 0,3. Si se implementan métodos de acceso no necesarios y son correctos no restaría nada porque no es importante y los alumnos se suelen liar. Además de que no es crítico a nivel de programación.

Apartado 2: 1 punto

- Extiende de AutonomousCar: 0,2
- Constructor correcto: 0,2
- Método calculateJourneyCost: 0,6

Apartado 3: 2 punto

- Constructor y atributos: 0,2
- Método releaseCar: 0,8
- Método bookCar: 1

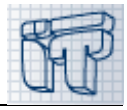
Apartado 4: 0,5 puntos

- Implementación de la interfaz: 0,2
- Implementación en AutonomousCar: 0,3

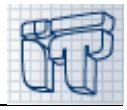
Ejercicio 1. Soluciones

```
package partial1_2019;

public interface Billable {

  String generateBill();

}
```

```
package partial1_2019;

public abstract class AutonomousCar implements Billable {

  protected String registrationNumber;
  protected int numberOfSeats;
  protected boolean isFree;
  protected long startJourneyTime;
  protected long endJourneyTime;

  public AutonomousCar(String registrationNumber, int numberOfSeats) {
    this.registrationNumber = registrationNumber;
    this.numberOfSeats = numberOfSeats;
```



```
    isFree = true;
  }

  public String getRegistrationNumber() {
    return registrationNumber;
  }

  public void bookCar() {
    isFree = false;
    startJourneyTime = System.currentTimeMillis();
  }

  public void releaseCar() {
    isFree = true;
    endJourneyTime = System.currentTimeMillis();
  }

  public boolean isFree() {
    return isFree;
  }

  public int getNumberOfSeats() {
    return numberOfSeats;
  }

  public abstract float calculateJourneyCost();


  public String generateBill() {
    float cost = calculateJourneyCost();
    return "The amount of the bill is " + cost + " + " + cost * 0.21f + " = "
+ cost * 1.21f + "€";
  }

}


package partial1_2019;

public class ElectricCar extends AutonomousCar {

  protected final float COST_PER_MINUTE = 2.5f;
  protected final float INITIAL_JOURNEY_COST = 5f;

  public ElectricCar(String registrationNumber, int numberOfSeats) {
    super(registrationNumber, numberOfSeats);
  }

  public float calculateJourneyCost() {
    return INITIAL_JOURNEY_COST +
        (endJourneyTime - startJourneyTime) * COST_PER_MINUTE / 1000 / 60;
  }

}

package partial1_2019;

public class Scheduler {
```



```
  AutonomousCar[] cars;

  public Scheduler(AutonomousCar[] cars) {
    this.cars = cars;
  }

  public void releaseCar(String registrationNumber) {

    for (int i = 0; i < cars.length; i++) {
      if (cars[i].getRegistrationNumber().equals(registrationNumber)) {
        cars[i].releaseCar();
      }
    }

  }

  public AutonomousCar bookCar(int minimumNumberOfSeats) throws
NotFreeCarException {

    for (int i = 0; i < cars.length; i++) {
      AutonomousCar car = cars[i];
      if (car.isFree() && (car.getNumberOfSeats() >= minimumNumberOfSeats)) {
        car.bookCar();
        return car;
      }
    }

   return null;

  }

}
```

Ejercicio 2. Rúbrica

Criterios de corrección:

Apartado 1: 0.6 puntos

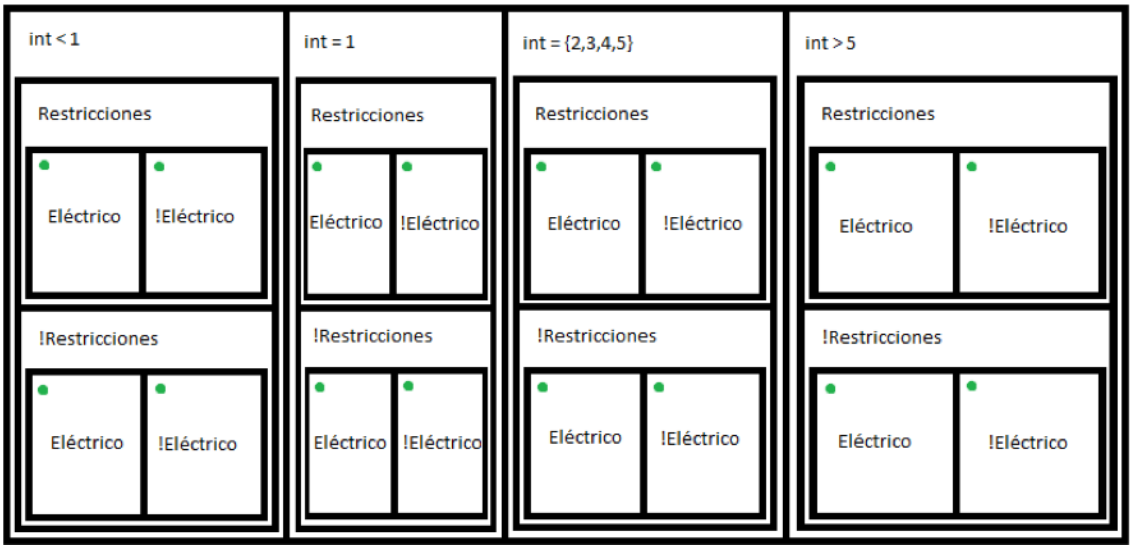
- Consideraciones restr (true, false) : 0.1 ptos.
- Consideraciones electr (true,false) : 0.1 ptos.
- Consideraciones int=1 : 0.1 ptos.
- Consideraciones int={2,3,4,5} : 0.1 ptos.
- Consideraciones int<1 : 0.1 ptos.
- Consideraciones int>5 : 0.1 ptos.

Apartado 2: 1.4 puntos

- 6 ramas lógicas en el código:
  - 5 ramas en condiciones !excepción : 0.2 ptos./rama x 5 rama = 1 ptos.
  - 1 rama en condición excepción: 0.4 ptos.

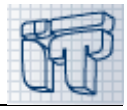
Ejercicio 2. Soluciones

Apartado 1



Apartado 2

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

public class GestionMultasTest {

    @Test
    public void testMultarCoche() {
        GestionMultas gestion = new GestionMultas();
        assertEquals(gestion.multarCoche(true, true, 1),0);
        assertEquals(gestion.multarCoche(false, false, 1),0);
        assertEquals(gestion.multarCoche(false, true, 1),0);
        assertEquals(gestion.multarCoche(true, false, 1),40);
        assertEquals(gestion.multarCoche(true, false, 3),45);
    }

    @Test
    public void testMultarCocheIllegalArgumentException() {
        GestionMultas gestion = new GestionMultas();
        assertThrows(IllegalArgumentException.class, ()-
>{gestion.multarCoche(false, false, 8);});
    }
```



```
}
```