Práctica 3

Introducción a la Programación Orientada a Objetos con Java

Inicio: A partir del 8 de marzo.
Duración: 3 semanas.
Entrega: En Moodle, una hora antes del comienzo de la siguiente práctica según grupos (semana del 5 de abril)
Peso de la práctica: 25%

Esta práctica da una introducción a la programación orientada a objetos con el lenguaje Java. Para ello, se desarrollarán de forma incremental varias clases en Java (incluyendo pruebas y documentación) que implementan varios componentes de una pequeña aplicación de gestión de tráfico.
En el desarrollo de esta práctica se utilizarán principalmente los siguientes conceptos de Java:

- *tipos de datos primitivos, String, Array y tipos referencia* (objetos) definidos por el programador,
- *clases* sencillas definidas por el programador para implementar tipos abstractos de datos mediante *variables de instancia, variables de clase, métodos de instancia, métodos de clase y constructores*.
- *entrada/salida elemental* para lectura de archivos en formato texto y visualización de texto en la consola
- *herencia, sobrescritura de métodos*
- *iniciación a las colecciones*
- *buen estilo de programación y comentarios para documentación mediante Javadoc.*

Apartado 0. Introducción

En esta práctica vamos a seguir desarrollando la aplicación de la dirección general de tráfico (DGT) cuyo diseño se realizó en el **apartado 1 de la práctica 2**. Inicialmente partiremos de la descripción dada en la práctica 2, y en sucesivos apartados de la práctica actual iremos progresivamente añadiendo nuevos requisitos de funcionalidad.

Apartado 1. Conductores y propietarios de vehículos (3 puntos)

El programa de más abajo se encarga de asociar los vehículos con los propietarios que los han adquirido. Estos propietarios pueden ser una persona física o una sociedad, en cuyo caso existe un responsable de la misma que es una persona física. Además, cada vehículo puede tener asociado un conductor, que debe ser una persona física (a efectos de la gestión de puntos por las multas). Si no se especifica, el conductor por defecto se establece a la persona propietaria del vehículo (si el propietario es una persona física) o al responsable de la sociedad.

Como ves, en el constructor de cada tipo de vehículo, opcionalmente se puede indicar el propietario (sociedad o persona). También puede establecerse el propietario o conductor de un vehículo mediante los métodos *setPropietario()* y *setConductor()*. Las personas pueden ser propietarias de más de un vehículo, y al imprimir cada persona, se imprimen los vehículos de su propiedad.

Nota:

- Para representar la colección de vehículos que posee una persona o sociedad, puedes usar una lista (por ejemplo, un *ArrayList*). Este es un tipo de colección en Java que se parametriza con el tipo de los elementos que contiene. Por ejemplo: *ArrayList<Vehiculo> vehiculos = new ArrayList<>()*; declara una variable *vehiculos* que es una lista de objetos de tipo *Vehiculo*. Puedes añadir un vehículo *v* a la lista usando el método *vehiculos.add(v)*, y obtener el tamaño de la lista con el método *vehiculos.size()*.
- Fíjate también en la declaración de paquetes de la primera línea y la importación de elementos de la segunda línea: deberás organizar tus clases en paquetes siguiendo ese esquema.

Se pide:

a) Modificar el diseño (el diagrama de clases) para acomodar estos nuevos requisitos. Ten en cuenta que un buen diseño orientado a objetos quizá deba contener clases adicionales, no mencionadas en el tester.

b) Codificar el diseño en Java. No te olvides de seguir la guía de estilo de programación Java disponible en Moodle, incluyendo comentarios, especialmente los usados para javadoc, en el código de todas las clases que escribas.

Nota: Es posible que al implementar los siguientes apartados (en particular la creación de carnets de conducir), el tester deje de dar la salida que se especifica aquí. En ese caso, modifica posteriormente este tester (añadiendo los carnets correspondientes a los conductores) para que produzca la salida esperada.

Tester apartado 1: (disponible en Moodle)

```
package pr3.trafico.vehiculos;
import pr3.trafico.conductores.*;

public class TesterPr31 {
  public static void main(String[] args) {
        Persona ana    = new Persona("Ana Soler", 30);
        Persona luisa  = new Persona("Luisa Puerto", 17);
        Persona andres = new Persona("Andres Molina", 27);
        Sociedad edsa  = new Sociedad("Entregas a Domicilio S.A.", ana);  // Ana es responsable de EDSA

        Vehiculo parque[] = {
                new Coche("Fiat 500x", 2019, "1245 HYN", true, ana),         // Coche de Ana, que lo conduce
                new Camion("IvecoDaily", 2010, "5643 KOI", 2, edsa),         // Camion de EDSA, que conduce Ana
                new Motocicleta("Harley Davidson", 2003, "0987 ETG", false)}; // Sin propietario

        parque[2].setPropietario(ana);  // Ana es la propietaria, y conductora por defecto
        // El método setPropietario debe permitir pasar una Sociedad: parque[2].setPropietario(edsa);
        System.out.println("Luisa puede conducir una Harley? "+parque[2].setConductor(luisa));
        // Luisa, al ser menor de 18 no puede ser conductora

        for (Vehiculo v : parque ) {
                System.out.println(v);
                System.out.println("------------------------");
        }

        parque[2].setConductor(andres);
        System.out.println(parque[2]);
        System.out.println("Personas:");
        System.out.println(andres+"\n---");
        System.out.println(ana+"\n---");
        System.out.println("Sociedad:");
        System.out.println(edsa);
  }
}
```

Salida esperada apartado 1:

```
Luisa puede conducir una Harley? false
Coche diesel, modelo Fiat 500x, matrícula: 1245 HYN, fecha compra 2019, con 4 ruedas, índice:C propietario: Ana Soler conductor: Ana Soler
------------------------
Camión de 2 ejes, modelo IvecoDaily, matrícula: 5643 KOI, fecha compra 2010, con 4 ruedas, índice:B propietario: Entregas a Domicilio S.A. conductor: Ana Soler
------------------------
Motocicleta, modelo Harley Davidson, matrícula: 0987 ETG, fecha compra 2003, con 2 ruedas, índice:C propietario: Ana Soler conductor: Ana Soler
------------------------
Motocicleta, modelo Harley Davidson, matrícula: 0987 ETG, fecha compra 2003, con 2 ruedas, índice:C propietario: Ana Soler conductor: Andres Molina
Personas:
Andres Molina
---
Ana Soler propietario de:
Coche diesel, modelo Fiat 500x, matrícula: 1245 HYN, fecha compra 2019, con 4 ruedas, índice:C propietario: Ana Soler conductor: Ana Soler
Motocicleta, modelo Harley Davidson, matrícula: 0987 ETG, fecha compra 2003, con 2 ruedas, índice:C propietario: Ana Soler conductor: Andres Molina
---
Sociedad:
Entregas a Domicilio S.A. (responsable: Ana Soler) propietario de:
Camión de 2 ejes, modelo IvecoDaily, matrícula: 5643 KOI, fecha compra 2010, con 4 ruedas, índice:B propietario: Entregas a Domicilio S.A. conductor: Ana Soler
```

Apartado 2. Lectura de multas desde archivos de texto (1.5 puntos)

En este apartado debes desarrollar una clase de utilidad *LecturaRadares* con un método estático *leer* que lea línea a línea un archivo de texto que contiene siempre todos los datos de las multas impuestas por los distintos radares y los devuelva en una lista. Como ves más abajo, el retorno de *leer()* debe ser compatible con *List<Multa>*, por ejemplo, *ArrayList<Multa>*. Recuerda que para usar estas colecciones tu programa deberá incluir una instrucción *import* similar a la que hay al principio del tester. El método *leer()* recibe un parámetro que indica el nombre del archivo de texto a leer, cuya estructura se muestra a continuación.

Estructura del archivo con datos de entrada para tester 2 (disponible en Moodle, archivo `multas_radar1.txt`):
MATRICULA;TIPO_MULTA;PUNTOS

Cada línea del archivo consta de una serie de campos separados por un signo **`;`** (que podemos suponer que nunca aparecerá en el contenido de ningún campo). Los campos de cada línea describen todos los atributos de una multa: la matrícula del vehículo, una descripción del tipo de multa, y los puntos a descontar. Después de descomponer en campos cada línea leída, se debe crear un objeto *Multa* para añadirlo a la lista de retorno. Como la clase *LecturaRadares* es una clase de utilidad (para lectura de fichero), no debemos dejar crear objetos de ese tipo.

Tester apartado 2 (disponible en Moodle)**:**

```
package pr3.trafico.multas;
import java.util.List;

public class TesterRadares {
        public static void main(String[] args) {
                List<Multa> multas = LecturaRadares.leer("multas_radar1.txt");

                for (Multa m : multas)
                        System.out.println(m+"\n------------");
        }
}
```

Salida esperada apartado 2:

```
Multa [matricula=1245 HYN, tipoMulta=Exceso de velocidad, puntos=2]
------------
Multa [matricula=5643 KOI, tipoMulta=Aparcamiento indebido, puntos=1]
------------
Multa [matricula=0987 ETG, tipoMulta=Conduccion en direccion contraria, puntos=12]
------------
Multa [matricula=1245 HYN, tipoMulta=Adelantamiento en linea continua, puntos=2]
------------
Multa [matricula=0987 ETG, tipoMulta=Aparcamiento indebido, puntos=1]
------------
```

Apartado 3. Carnets de conducir (1.5 puntos)

En este apartado crearemos una clase que representa el carnet de conducir. Podremos asociar carnets a personas, y comprobaremos que estas reúnen los requisitos para ser conductor de las distintas clases de vehículos.

Un carnet tiene asociado un número de puntos (inicialmente 12), que pueden ir disminuyendo según se cometan infracciones. Los carnets deberán llevar asociados un identificador numérico único, creado automáticamente por el sistema. Un carnet permite conducir diversos tipos de vehículos (uno o más), dependiendo de la licencia. Por simplicidad, consideraremos 3 tipos de licencia: A (para conducir motocicletas), B (para conducir coches) y C1 (para conducir camiones). La edad mínima requerida para las licencias A y B son 18 años, mientras que para la C1 son 23. Debes asegurarte de que, cuando se asigna un conductor a una persona, tiene la edad mínima para todas las licencias del carnet. Además, cuando se asigna un conductor a un vehículo, el conductor debe tener carnet con la licencia apropiada (de otro modo el conductor queda sin asignar). Finalmente, el número de puntos de un carnet no puede ser menor de 0.

El tester de más abajo contiene tres pruebas que ejercitan algunas de las características de los carnets. Añade más pruebas que ejecuten en más profundidad las clases creadas. Este estilo se utiliza para crear *pruebas de unidad*, y frameworks como JUnit (https://junit.org/) facilitan su escritura. En sesiones posteriores (y en PADSOF) se explicará JUnit, pero puedes utilizarlo en esta práctica si quieres para crear los tests.

Nota: Recuerda hacer un diseño extensible y flexible, que permita añadir fácilmente nuevos tipos de licencia (con edades requeridas distintas) y facilite cambiar fácilmente las edades requeridas.

Tester apartado 3 (disponible en Moodle, debes añadir imports y decidir en qué paquete colocar esta clase)

```
// pon el tester en el paquete más adecuado, e importa las clases necesarias
public class TesterAp3 {
  private void testMenorDe18NoPuedeTenerCarnetA() {
        Persona ana = new Persona("Ana Soler", 17);
        Carnet  c   = new Carnet(TipoLicencia.A);
        System.out.println("Test: MenorDe18NoPuedeTenerCarnetA");
        System.out.println(c);
        System.out.println(ana.setCarnet(c));    // debe devolver false, porque Ana no tiene 18
  }
  private void testMenorDe23NoPuedeTenerCarnetC1() {
        Persona ana = new Persona("Ana Soler", 19);
        Carnet  c   = new Carnet(TipoLicencia.A, TipoLicencia.C1);
        System.out.println("=================\nTest: MenorDe23NoPuedeTenerCarnetC1");
        System.out.println(c);
        System.out.println(ana.setCarnet(c));    // debe devolver false, porque Ana no tiene 23
  }
  private void testCarnetParaCategoria() {
        Persona ana = new Persona("Ana Soler", 24);
        ana.setCarnet(new Carnet(TipoLicencia.A, TipoLicencia.C1));
        Coche c     = new Coche("Fiat 500x", 2019, "1245 HYN", true, ana);
        System.out.println("=================\nTest: CarnetParaCategoria");
        System.out.println(c);           // ana no puede ser conductor, porque no tiene licencia para coches
        ana.getCarnet().addLicencia(TipoLicencia.B);
        c.setConductor(ana);
        System.out.println(c);           // ahora sí
        System.out.println(ana.getCarnet());             // carnet
  }
  public static void main(String[] args) {
        TesterAp3 tap3 = new TesterAp3();
        tap3.testMenorDe18NoPuedeTenerCarnetA();
        tap3.testMenorDe23NoPuedeTenerCarnetC1();
        tap3.testCarnetParaCategoria();
  }
}
```

Salida esperada:

```
Test: MenorDe18NoPuedeTenerCarnetA
Carnet [id=0, licencias=[A], puntos=12]
false
=================
Test: MenorDe18NoPuedeTenerCarnetC1
Carnet [id=1, licencias=[A, C1], puntos=12]
false
=================
Test: CarnetParaCategoria
Coche diesel, modelo Fiat 500x, matrícula: 1245 HYN, fecha compra 2019, con 4 ruedas, índice:C propietario: Ana Soler
conductor: no registrado
Coche diesel, modelo Fiat 500x, matrícula: 1245 HYN, fecha compra 2019, con 4 ruedas, índice:C propietario: Ana Soler
conductor: Ana Soler
Carnet [id=2, licencias=[A, C1, B], puntos=12]
```

Apartado 4. Procesado de multas (1.5 puntos)

A continuación, vamos a crear la funcionalidad para procesar las multas, restando los puntos correspondientes del carnet de los conductores de los vehículos multados. Para ello, debes crear una clase *ProcesadorMultas* que reciba en el constructor una lista de vehículos a considerar. El método *procesar()*, recibirá la lista de multas, iterará sobre ellas, y descontará los puntos correspondientes. El procesador deberá imprimir los siguientes mensajes:

- Un mensaje con los puntos que pierde cada conductor por cada multa.
- Un mensaje si el conductor llega a cero puntos.
- Un mensaje si el conductor tenía el carnet suspendido, o la multa hace que se le suspenda el carnet (ver más abajo).
- Un mensaje si el vehículo no tiene conductor asociado (y por tanto la DGT iniciará acciones legales contra el dueño).

Ten en cuenta lo siguiente a la hora de procesar la resta de puntos:

- Si un carnet tiene menos puntos que los que hay que restar por la infracción, se le asignan 0 puntos.
- Si un carnet tiene cero puntos y su conductor realiza una infracción, el carnet pasa a estar suspendido (y sigue con 0 puntos). Un conductor con carnet suspendido no podrá circular, y la DGT iniciará acciones legales contra él.

A modo de prueba, el siguiente programa debe producir la salida de más abajo.

Tester Apartado 4 (disponible en Moodle, debes añadir imports y decidir en qué paquete colocar esta clase)

```
public class TesterMultas {
  public static void main(String[] args) {
        Persona ana    = new Persona("Ana Soler", 30);            // Ana
        Persona andres = new Persona("Andres Molina", 27);
        Sociedad edsa  = new Sociedad("Entregas a Domicilio S.A.", ana);  // Ana es responsable de EDSA

        ana.setCarnet(new Carnet(TipoLicencia.B, TipoLicencia.C1));
        andres.setCarnet(new Carnet(TipoLicencia.A));

        Vehiculo parque[] = {
                new Coche("Fiat 500x", 2019, "1245 HYN", true, ana), // Coche de Ana, que lo conduce
                new Camion("IvecoDaily", 2010, "5643 KOI", 2, edsa), // Coche de EDSA
                new Motocicleta("Harley Davidson", 2003, "0987 ETG", false, andres)};

        ProcesadorMultas pm = new ProcesadorMultas(Arrays.asList(parque));
        pm.procesar(LecturaRadares.leer("multas_radar1.txt"));
  }
}
```

Salida esperada:

```
El conductor Ana Soler pierde 2 puntos
El conductor Ana Soler pierde 1 puntos
El conductor Andres Molina pierde 12 puntos
El conductor Andres Molina queda con 0 puntos
El conductor Ana Soler pierde 2 puntos
El conductor Andres Molina pierde 1 puntos
Queda suspendido el carnet del conductor Andres Molina
```

Nota: En este apartado, debes crear más casos de prueba que ejerciten el resto de condiciones (por ejemplo, vehículos sin conductor declarado).

Apartado 5. Ampliando el diseño original (2.5 puntos)

Ampliar el diseño original de la aplicación para la dirección general de tráfico, incluyendo el concepto de ITV (Inspección Técnica de Vehículos). Todos los vehículos deben pasar una ITV periódicamente a partir de una antigüedad, medida respecto al año de matriculación (que por simplicidad supondremos igual al año de compra). La periodicidad de la ITV puede ser distinta para cada tipo de vehículo:

- coches y motocicletas a partir de los 4 años de antigüedad, cada dos años, y a partir de 10 años de antigüedad, anualmente;
- camiones: hasta los 2 años exentos de pasar la ITV, de 2 a 6 años cada 2 años, con más de 6 hasta 10 años cada año y más de 10 años cada 6 meses.

Cada vehículo debe mantener una lista de las ITVs que ha pasado, y cada ITV debe contener información de la fecha, taller de realización, y un campo con comentarios. El taller de realización viene dado por un nombre, una dirección y una provincia. Se deben incluir métodos para gestionar y consultar el estado de la ITV de un vehículo y para consultar el tiempo que queda hasta la siguiente inspección. Además, queremos obtener todos los vehículos que han pasado una ITV en un determinado taller.

Esta nueva funcionalidad tiene implicaciones en otras partes de tu diseño: si el procesador de multas detecta una infracción de un vehículo con ITV caducada, debe imprimir un mensaje por pantalla, restar un punto adicional al conductor del vehículo, e imprimir la infracción en un fichero *ITV_caducada.txt* con igual formato que el fichero de multas.

Nota: En este apartado se te da más libertad para tu diseño (no te damos tester), pero tú mismo debes diseñar casos de prueba que prueben de manera exhaustiva esta nueva funcionalidad.

Normas de Entrega:

- Se debe entregar un único fichero ZIP con todo lo solicitado abajo, que deberá llamarse de la siguiente manera: **P3_**GR<numero_grupo>_<nombre_estudiantes>.zip. Por ejemplo Marisa y Pedro, del grupo 2213, entregarían el fichero: **P3_**GR2213_MarisaPedro.zip.
- El fichero debe contener:
    - un directorio ***src*** con todo el código Java en su **versión final del apartado 5**, incluidos los datos de prueba y testers adicionales que hayas desarrollado en los apartados que lo requieren,
    - un directorio ***doc*** con la documentación generada
    - un directorio ***txt*** con todos los archivos de datos de texto utilizados y generados en las pruebas
    - un archivo PDF con el **diagrama de clases,** una explicación y una breve justificación de las decisiones que se hayan tomado en el desarrollo de la práctica, los problemas principales que se han abordado y cómo se han resuelto, así como los problemas pendientes de resolver.