Análisis y Diseño de Software

Curso 2020-2021

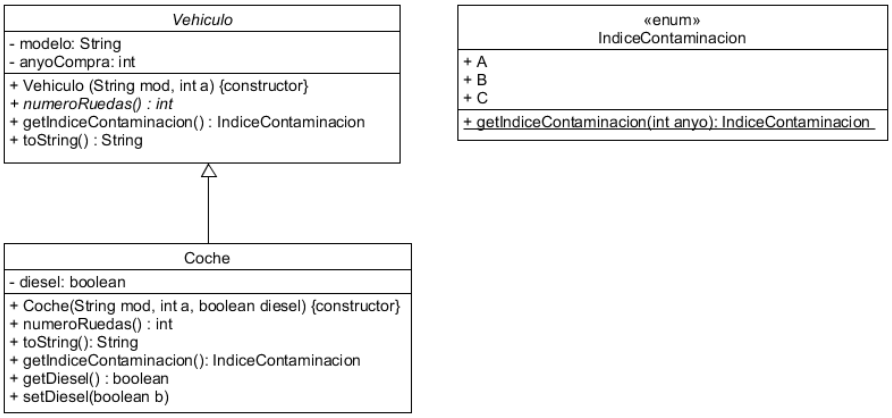
Práctica 2

Diseño Orientado a Objetos

Inicio: A partir del 22 de febrero.
Duración: 2 semanas.
Entrega: En Moodle, una hora antes del comienzo de la siguiente práctica según grupos (semana del 8 de marzo)
Peso de la práctica: 15%

El objetivo de esta práctica es introducir los conceptos básicos de orientación a objetos, desde el punto de vista del diseño (usando UML), así como de su correspondencia en Java.

Apartado 1 (3,5 puntos):



Te piden crear una aplicación para la dirección general de tráfico de una provincia, que permita almacenar datos de interés de distintos tipos de *vehículos*, así como el cálculo de su *índice de contaminación* (A, B o C, donde la A es el menos contaminante). Para ello, te proporcionan el diseño que se muestra en el diagrama de clases de derecha.

Como ves, una clase abstracta *Vehiculo* almacena el modelo y año de compra, y una subclase *Coche*, almacena si el coche es diésel o gasolina. Además del constructor, la clase *Vehiculo* contiene un método abstracto *numeroRuedas* (que debe ser implementado por todas las subclases para indicar el número de ruedas del vehículo), un método *getIndiceContaminacion*, que devuelve un enumerado de tipo *IndiceContaminacion*, y el método *toString*, que ya conoces de la práctica 1. Los enumerados son similares a clases. El enumerado *IndiceContaminacion* define 3 posibles valores (A, B, C, que en realidad son objetos de tipo *IndiceContaminacion*), y un método estático *getIndiceContaminacion*. Un método estático no necesita el contexto de un objeto para invocarse, sino que se define e invoca a nivel de clase (un enumerado en este caso).

La subclase *Coche* define su propio constructor, implementa el método *numeroRuedas (*que no tenía implementación en la clase `Vehiculo`), define un *getter* y un *setter* para el atributo *diesel*, y sobreescribe los métodos *toString* y *getIndiceContaminacion* de la clase base.

Los siguientes listados muestran la implementación completa de dichas clases en Java. Como puedes ver, las clases se han incluido en el paquete "*prac2.trafico*". Los paquetes sirven para organizar las clases que intervienen en los programas. De esta manera, las clases más relacionadas entre sí se declararán en el mismo paquete. La estructura de los paquetes es jerárquica y se corresponde con la estructura física de directorios (es decir, las clases han de almacenarse en un directorio "*prac2/trafico*").

Los *constructores* tienen el mismo nombre que la clase, y son invocados mediante **new** para construir un objeto (crear una instancia de la clase) e inicializar sus atributos. Así, el constructor de *Vehiculo* toma dos parámetros e inicializa los atributos *modelo* y *anyoCompra*, y el constructor de la clase *Coche* llama mediante **super(…)** al constructor de su superclase `Vehiculo` antes de inicializar el atributo *diesel*. La anotación (opcional) *@Override* indica que la intención del programador es sobreescribir un método (que ha de tener igual firma) de alguna de las superclases.

También se ha incluido un programa de ejemplo y su salida. Observa que, como todos los coches son vehículos, podemos almacenarlos en un array de tipo *Vehiculo*. Esto nos será útil para poder añadir más subclases de *Vehiculo*, y poder manejar sus objetos de manera uniforme. El código Java de estas clases está disponible en Moodle.

Clase Vehiculo

```java
package prac2.trafico;

public abstract class Vehiculo {
  private String modelo;
  private int anyoCompra;

  public Vehiculo(String mod, int a) {
    this.modelo = mod;
    this.anyoCompra = a;
  }

  @Override public String toString() {
    return "modelo "+this.modelo+", fecha compra "+this.anyoCompra+", con "+
           this.numeroRuedas()+" ruedas, índice:"+this.getIndiceContaminacion();
  }

  public abstract int numeroRuedas();

  public IndiceContaminacion getIndiceContaminacion() {
    return IndiceContaminacion.getIndiceContaminacion(this.anyoCompra);
  }
}
```

Clase Coche

```java
package prac2.trafico;

public class Coche extends Vehiculo {
  private boolean diesel;

  public Coche(String mod, int a, boolean diesel) {
    super(mod, a);
    this.diesel = diesel;
  }

  @Override public int numeroRuedas() { return 4;     }

  @Override public String toString() {
    return "Coche "+(this.diesel ? "diesel" : "gasolina") + ", "+ super.toString();
  }

  @Override
  public IndiceContaminacion getIndiceContaminacion() {
    if (this.diesel) return IndiceContaminacion.C;
    return super.getIndiceContaminacion();
  }

  public boolean getDiesel() { return this.diesel; }
  public void setDiesel(boolean b) { this.diesel = b; }
}
```

Enumerado IndiceContaminacion

```java
package prac2.trafico;

public enum IndiceContaminacion {
  A, B, C;

  private static final int FECHAA = 2018;     // constante
  private static final int FECHAB = 2010;     // constante

  public static IndiceContaminacion getIndiceContaminacion(int anyo) {
    if (anyo >= FECHAA) return A;
    if (anyo >= FECHAB) return B;
    return C;
  }
}
```

Programa Ejemplo

```
package prac2.trafico;

public class Ejemplo1 {
  public static void main(String[] args) {
    Coche fiat500x = new Coche("Fiat 500x", 2019, true);    // Fiat 500x del 2019 diesel
    Coche minic    = new Coche("Mini Copper", 2015, false); // Mini del 2015 gasolina
    Coche xsara    = new Coche("Citroen XSara", 1997, true);

    Vehiculo [] vehiculos = { fiat500x, minic, xsara }; // almacenamos en un array

    for (Vehiculo v : vehiculos )  // iteramos e imprimimos
      System.out.println(v);
  }
}
```

Salida:

```
Coche diesel, modelo Fiat 500x, fecha compra 2019, con 4 ruedas, índice:C
Coche gasolina, modelo Mini Copper, fecha compra 2015, con 4 ruedas, índice:B
Coche diesel, modelo Citroen XSara, fecha compra 1997, con 4 ruedas, índice:C
```

Se pide: Extender el **diseño UML y el programa Java** con dos nuevos tipos de vehículo: motocicletas y camiones. Además del modelo y año de compra, una motocicleta almacena si es eléctrica o no. Las motocicletas tienen dos ruedas, y si son eléctricas tienen índice de contaminación "A", independientemente del año de compra. Si no lo son, aplican las normas generales del resto de vehículos. Los camiones tienen modelo, año de compra y número de ejes. El número de ruedas de un camión viene dado por el doble del número de ejes, y su índice de contaminación es C si tiene más de dos ejes. En otro caso, aplican las normas generales del resto de vehículos. Finalmente, queremos almacenar la matrícula de cualquier tipo de vehículo.

Como guía el siguiente programa de ejemplo debe dar la salida de más abajo.

```
public class Ejemplo2 {
  public static void main(String[] args) {
        Coche fiat500x = new Coche("Fiat 500x", 2019, "1245 HYN", true);

        Motocicleta moto1 = new Motocicleta("Harley Davidson", 2003, "0987 ETG", false);
        Motocicleta moto2 = new Motocicleta("Torrot Muvi", 2015, "9023 MCV", true);

        Camion camion1 = new Camion("MAN TGA410", 2000, "M-3456-JZ", 3);
        Camion camion2 = new Camion("Iveco Daily", 2010, "5643 KOI", 2);

        Vehiculo [] vehiculos = { fiat500x, moto1, moto2, camion1, camion2 };

        for (Vehiculo v : vehiculos )
                System.out.println(v);
  }
}
```

Salida:

```
Coche diesel, modelo Fiat 500x, matrícula: 1245 HYN, fecha compra 2019, con 4 ruedas, índice:C
Motocicleta, modelo Harley Davidson, matrícula: 0987 ETG, fecha compra 2003, con 2 ruedas, índice:C
Motocicleta eléctrica, modelo Torrot Muvi, matrícula: 9023 MCV, fecha compra 2015, con 2 ruedas, índice:A
Camión de 3 ejes, modelo MAN TGA410, matrícula: M-3456-JZ, fecha compra 2000, con 6 ruedas, índice:C
Camión de 2 ejes, modelo IvecoDaily, matrícula: 5643 KOI, fecha compra 2010, con 4 ruedas, índice:B
```

Nota:

En realidad, un diagrama de clases es una abstracción del código final, así que normalmente y para facilitar su comprensión no se suelen incluir todos los detalles necesarios para la codificación. De esta manera se suelen omitir los métodos *setters* y *getters* y los constructores.

Por si quieres saber más...:

El método *toString()* de la clase *Vehiculo* es un ejemplo del patrón de diseño "Template Method", que consiste en escribir código en una clase padre que llama a métodos (quizá abstractos, como *numeroRuedas*()) que se implementan en las subclases. Tienes más información sobre patrones de diseño en:

Patrones de diseño elementos de software orientado a objetos reutilizable. Gamma, E., Helm, R., Johnson, R., Vlissides, J. INF/681.3.06/PAT. Addison-Wesley, 2003.

Apartado 2 (3 puntos):

Se quiere construir una aplicación para la elaboración de cuestionarios estadísticos, y la recogida de respuestas anónimas por usuarios. Un cuestionario viene dado por un texto introductorio, el número de minutos estimados para completarlo y la pregunta inicial. Cada pregunta tiene un texto, y se considerarán tres tipos: de respuesta abierta, de selección simple y de selección múltiple. Una pregunta de respuesta abierta se configura con un máximo de caracteres de respuesta, que por defecto serán 128. Las preguntas de selección presentarán 2 o más opciones, cada una con un texto. Se puede configurar si dichas opciones se presentan al usuario aleatoriamente o por el orden en el que se han introducido. Las preguntas de selección múltiple pueden configurarse con un flag, que indica si es posible no elegir ninguna opción como respuesta.

Además, al crear un cuestionario, hay que configurar cómo se navega de una pregunta a la siguiente. En el caso más simple, se selecciona una pregunta siguiente, pero el sistema debe soportar navegación condicional. En este caso, la siguiente pregunta depende de las opciones elegidas por el usuario en el caso de preguntas de selección. De este modo, desde una pregunta de selección, es posible establecer distintas navegaciones a distintas preguntas, en función de las distintas opciones seleccionadas como respuesta.

El sistema debe guardar las respuestas de los usuarios a los distintos cuestionarios de forma anónima. En particular, para un cuestionario se guarda la fecha de inicio y fin, así como la fecha de inicio y fin de las respuestas a cada pregunta. En el caso de preguntas de selección, se guardan las opciones seleccionadas, y en el caso de preguntas abiertas, el texto respondido.

Se pide:

a) Realiza un diagrama de clases UML que refleje el diseño de la aplicación. No es necesario que incluyas *constructores*, ni métodos *getters* o *setters* (**2 puntos**).

b) Incluye métodos en las clases de tu diagrama para (**0.6 puntos**):

1. Mostrar por pantalla los distintos tipos de preguntas.
2. Obtener cuál es la siguiente pregunta, dada una respuesta del usuario a una pregunta.
3. Obtener el tiempo que un usuario ha tardado en responder a una pregunta, y a un cuestionario.
4. Obtener la desviación (en segundos) de la media del tiempo de respuesta de un cuestionario, respecto del tiempo estimado para su compleción.

Ten en cuenta que en algún caso puede ser necesario añadir métodos auxiliares. No es necesario que incluyas el código del método.

c) Realiza un diagrama de objetos que represente un cuestionario con 3 preguntas (de respuesta abierta, simple y múltiple), dos navegaciones (condicional y simple), y una respuesta (**0.4 puntos**).

Apartado 3 (3,5 puntos):

Se quiere construir una aplicación para la gestión de reuniones virtuales, con soporte de vídeo y chat. Cuando un usuario quiere utilizar la aplicación, debe introducir un *nick* (que debe ser único en el sistema), mediante el que será identificado en las reuniones en las que participe. Una vez dentro de la aplicación, el usuario tiene la opción de registrarse, para lo que se le pedirá un nombre, email y contraseña.

Un usuario registrado puede crear una sala de reuniones, que se identifica por un nombre. Las salas de reuniones pueden ser públicas. Existen tres tipos de salas: de chat, de videoconferencia, y mixtas. Una sala de chat puede estar o no moderada, y se ha de establecer un idioma de conversación (inglés, francés, español, alemán u otro). Una sala de videoconferencia puede tener asociada una imagen de fondo virtual, que será usada automáticamente en todas las videoconferencias realizadas en la sala. Finalmente, una sala mixta puede contener salas de chat, videoconferencia u otras salas mixtas de manera jerárquica. El creador de una sala (de cualquier tipo), puede prohibir la entrada a ciertos usuarios, y el sistema impedirá la participación de estos usuarios en cualquier reunión que se celebre en dicha sala.

El creador de una sala, puede programar reuniones en ella. Una reunión viene dada por un nombre, una fecha y opcionalmente una duración máxima en minutos. Las reuniones serán de dos tipos: públicas y privadas. Las reuniones privadas pueden organizarse en salas públicas o privadas, pero en una sala privada sólo pueden organizarse reuniones privadas. En una reunión privada se ha de establecer la lista de usuarios (registrados o no) que pueden asistir. En una reunión pública se establece un límite máximo de asistentes. En cualquiera de los dos casos, no pueden participar usuarios que tengan la entrada prohibida a la sala en la que se organiza la reunión, o en alguna de las salas contenidas en ella (en caso de que la sala de la reunión sea una sala mixta).

El sistema debe registrar el contenido de las reuniones. En particular, en cualquier tipo de reunión, se grabarán los mensajes creados en las salas de chat. Un mensaje viene dado por el usuario que lo crea, un texto y una fecha. Se consideran dos tipos de mensajes: públicos (que llegan a todos los participantes que se encuentren en la sala) y privados (dirigidos a un usuario concreto). En cualquiera de los dos casos, un mensaje puede crearse como respuesta a otro. En las reuniones privadas realizadas en salas de videoconferencia, es posible grabar fragmentos de vídeo, que se identifican mediante las fechas inicial y final, así como el nombre del fichero de vídeo (asignado automáticamente por el sistema).

Se pide:

a) El diseño del sistema usando diagramas de clases. Añade los métodos necesarios en las clases para obtener la funcionalidad que menciona el enunciado. No es necesario incluir constructores, *getters* o *setters*. (**3 puntos**)

b) Especifica el pseudocódigo (o puedes usar Java) de los siguientes métodos: (**0.5 puntos**)

1. Comprobar que un usuario puede acceder a una reunión.
2. Obtener los usuarios con entrada prohibida en una sala, y todas las salas contenidas en ella.

Normas de Entrega:

- Se deberán entregar los apartados 1, 2 y 3. Crea un directorio por cada apartado.
- La entrega la realizará uno de los alumnos de la pareja, a través de Moodle.
- Si el ejercicio pide código Java, se ha de entregar además la documentación generada con *javadoc*. Si el ejercicio pide un diagrama de diseño, se deberá entregar en PDF junto con una breve explicación (dos o tres párrafos a lo sumo).
- Se debe entregar un único fichero ZIP / RAR con todo lo solicitado, que deberá llamarse de la siguiente manera: `GR<numero_grupo>_<nombre_estudiantes>.zip`. Por ejemplo Marisa y Pedro, del grupo 2261, entregarían el fichero: `GR2261_MarisaPedro.zip`.