PROGRAMACIÓN WEB

Curso 2022/2023

PRÁCTICA 1: INICIACIÓN A JAVA

En esta práctica nos iniciaremos en el desarrollo con el lenguaje Java, incorporando además patrones de diseño para acostumbrarnos como programadores a seguir las mejores prácticas (*best practices*) en el desarrollo de software, distintivas de los buenos profesionales. En un equipo de cuatro personas, que se mantendrá invariable durante el curso académico, haremos uso de los elementos habituales del lenguaje, además de E/S por consola y ficheros, gestión de ficheros de propiedades y desarrollo de patrones de diseño. Tenga en cuenta los siguientes **aspectos normativos** referentes a esta práctica:

- Se entregará un informe en formato PDF (máximo 2 páginas) que deberá explicar y fundamentar las decisiones de diseño e implementación que ha adoptado el equipo, así como la relación de fuentes consultadas y un análisis de las dificultades encontradas.
- El código debe seguir las buenas prácticas de nombrado (variables, paquetes, etc.) y estar documentado haciendo uso de Javadoc a nivel de clase y de métodos.

  https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html
- La propuesta de ejercicios tiene margen de decisión por parte del equipo. Dado que se espera que estas decisiones sean distintas por cada equipo de prácticas, deberán estar justificadas pertinentemente en el informe.
- Solo hará entrega de la práctica un miembro del equipo a través de Moodle. La persona que entregue esta práctica lo hará igualmente con las demás prácticas. Se deberá subir un archivo ZIP a la tarea Moodle con el proyecto Eclipse (preferiblemente) o fuentes Java de la práctica, los ejecutables de cada ejercicio (JAR) y el informe en la raíz del archivo. El archivo ZIP que se entrega debe ser nombrado de la siguiente forma:

  GM<*gm*>_<*login*>.zip, donde <*gm*> es el número de grupo mediano y <*login*> es el *nick* de la UCO de la persona del equipo que hace la entrega.

El incumplimiento de las normas de entrega (contenido, nombrado, etc.) implicará la consideración de práctica no entregada. La fecha de entrega se dispondrá en la tarea Moodle correspondiente. Igualmente, el retraso en la entrega supondrá la no calificación en la práctica.

IMPORTANTE: No se admitirá entrega alguna fuera de plazo o a través de un medio distinto a la tarea de Moodle (p.ej. envío por email).

INTRODUCCIÓN AL PROYECTO DE PRÁCTICAS

Las tres prácticas de la asignatura estarán encaminadas a la realización de una aplicación web para la gestión de circuitos de *karts*. Dicha gestión se centrará en facilitar la reserva de pistas según distintos tipos de eventos, así como controlar el estado y mantenimiento de los materiales (karts, pistas, etc.). En cada práctica se propondrán una serie de ejercicios para que, de forma incremental, se trabaje en las clases y artefactos (ficheros, base de datos, etc.) necesarios para desarrollar el proyecto. La división en ejercicios facilitará la organización del trabajo durante las sesiones de prácticas, pero debe tenerse en cuenta en todo momento que forman parte de un único proyecto.

EJERCICIO 1

En este ejercicio se van a desarrollar las clases Java necesarias para representar los conceptos del dominio de la aplicación. En concreto, se deberán diseñar e implementar las siguientes clases:

Clase Kart: Clase que representa a un vehículo que utilizan los usuarios para recorrer las pistas. La clase debe contener los siguientes atributos:

- Identificador del *kart*, de tipo entero.
- Tipo de *kart*, de tipo booleano, para distinguir los destinados a niños y a adultos.
- Estado del *kart*, de tipo enumeración, para indicar si el kart está disponible, reservado o en mantenimiento.

En cuanto a métodos, la clase deberá proporcionar los siguientes:

- Constructor vacío (sin parámetros).
- Constructor parametrizado, cuyos parámetros sean todos los atributos de la clase.
- Métodos *get*/*set* para todos los atributos.
- Método *toString* que imprima la información del *kart*.

Clase Pista: Esta clase representa una pista construida en las instalaciones. La clase debe contener los siguientes atributos:

- Nombre de la pista, que será único, de tipo *String*.
- Estado de la pista, de tipo booleano, para controlar si la pista está disponible para reservas o no.
- Dificultad de la pista, de tipo enumeración, con tres posibles valores: infantil, familiar, adultos.
- Número máximo de *karts* autorizados a usar la pista a la vez, de tipo entero.
- Lista de *karts* asociados para su uso en la pista (ver clase *Kart*).

Además, la clase deberá proporcionar los siguientes métodos:

- Constructor vacío (sin parámetros).
- Constructor parametrizado, cuyos parámetros sean todos los atributos de la clase excepto la lista de *karts*, que habrá de inicializarse vacío.
- Métodos *get*/*set* para todos los atributos.

- Método *toString* que imprima la información de la pista.
- Método *consultarKartsDisponibles*, que devuelva el subconjunto de *karts* disponibles, esto es, aquellos que no están en mantenimiento ni reservados.
- Método *asociarKartAPista*, que añada un *kart* al conjunto asociado a esta pista controlando que si la pista es infantil solo pueden añadirse *karts* para niños. Si la pista es familiar, podrá haber *karts* para niños y adultos, mientras que, si es de tipo adulto, solo podrán añadirse *karts* para adultos.

Clase Usuario: Esta clase representa a una persona usuaria de las instalaciones de la empresa de *karts*. La clase debe contener los siguientes atributos:

- Nombre y apellidos de la persona, de tipo *String*.
- Fecha de nacimiento de la persona, de tipo *Date*.
- Fecha de inscripción (primera reserva), de tipo *Date*.
- Correo electrónico, que será único, de tipo *String*.

Además, la clase deberá proporcionar los siguientes métodos:

- Constructor vacío (sin parámetros).
- Constructor parametrizado, cuyos parámetros sean todos los atributos de la clase salvo la fecha de inscripción (coger hora actual del sistema).
- Métodos *get*/*set* para todos los atributos.
- Método *toString* que imprima la información del usuario.
- Método *calcularAntiguedad*, que indique cuántos años lleva registrado.

Ayuda: Para el manejo de fechas, se recomienda consultar la documentación oficial, en concreto, las clases Date y DateFormat.

EJERCICIO 2

En este ejercicio se aborda la definición e implementación de las reservas, utilizando el patrón Factoría para su creación. El primer paso será la creación de la clase reserva, siguiendo las siguientes directrices:

Clase Reserva: Esta clase representa las reservas que los usuarios hacen de las pistas durante un tiempo determinado. Para cada reserva debe almacenarse cierta información general:

- Identificador del usuario que realiza la reserva, que debe estar previamente registrado.
- Fecha y hora para la que realiza la reserva, de tipo *Date*.
- Duración de la reserva, expresada en minutos (representación como entero).
- Identificador de la pista que se utilizará, que debe existir.
- Precio de la reserva, expresado en euros, de tipo flotante.

- Descuento que se le aplica por antigüedad, si es necesario.

Por otro lado, debe considerarse la existencia de distintos tipos de reservas:

- Reserva infantil: Es una reserva que realiza un adulto, pero en la que solo participan niños en una pista infantil. Se deberá registrar el número de niños que participan.
- Reserva familiar: Es una reserva que realiza un adulto, en la que participan tanto adultos como niños en una pista de tipo de familiar. Se deberá registrar el número de participantes de cada tipo, debiendo contabilizarse al menos un usuario adulto (independientemente de que sea o no el usuario que realiza la reserva).
- Reserva adultos: Es una reserva que realiza un adulto, en la que solo participan adultos en una pista de ese tipo. Se deberá registrar el número de participantes, incluyendo al que realiza la reserva.

A nivel de reserva (y sus tipos), se deben contemplar los siguientes métodos:

- Constructor vacío (sin parámetros).
- Métodos *get*/*set* para todos los atributos.
- Método *toString* que imprima la información de la reserva, el cual podrá requerir especialización según el tipo de reserva.

Para crear las reservas se deberá utilizar un patrón factoría, de forma que se permita crear los tres tipos de reservas en dos modalidades diferentes:

- Reserva individual: se crea una única reserva para un usuario y fecha determinada.
- Reserva bono: se crea una reserva que forma parte de un bono adquirido por un usuario. El identificador de dicho bono deberá quedar registrado, así como el número de sesión dentro del bono.

EJERCICIO 3

Haciendo uso de las clases anteriores, se deberán crear tres clases gestoras que implementen las funcionalidades que estarán disponibles en el programa principal.

Gestor de usuarios: Esta clase será la encargada de gestionar la información de los usuarios que hacen reservas. Deberá proporcionar las siguientes funcionalidades:

- Dar de alta a un usuario, comprobando que no está registrado previamente.
- Modificar la información de un usuario.
- Listar a los usuarios actualmente registrados.

Gestor de pistas: Esta clase será la que gestione las instalaciones y material necesario para dar servicio. En concreto, deberá implementar las siguientes funcionalidades:

- Crear pistas y karts.
- Asociar *karts* a pistas disponibles, controlando que los *karts* no estén asignados a otras pistas o en mantenimiento. También deben cumplirse las restricciones entre tipos de pistas (es decir, su dificultad) y de *karts*, así como el número máximo de *karts*.
- Listar las pistas en mantenimiento.

- Dado un número de *karts* y tipo de pista, devolver el conjunto de pistas que estén libres (no reservadas ni en mantenimiento) y tengan al menos ese número de *karts* asociados.

Gestor de reservas: Esta clase será la encargada de la gestión de reservas de pistas por parte de los usuarios. A continuación, se detallan las funcionalidades mínimas esperadas y las restricciones que deberán tenerse en cuenta:

- Hacer reservas individuales: Las reservas deberán realizarse por un usuario registrado y en una pista que cumpla las condiciones de la reserva (número de participantes, tipo de pista, etc.). Si el usuario tiene más de 2 años de antigüedad, se le aplicará un 10% de rebaja en el precio de la reserva.
- Hacer reservas dentro de un bono: Todas las sesiones del bono serán del mismo tipo (infantil, familiar o adultos), aunque no necesariamente con el mismo número de participantes ni duración siempre. Todas las reservas del bono se asocian a un mismo usuario, y contarán con un precio reducido (5% menos que el precio original según la duración) respecto al de la reserva individual. No se considerará la aplicación de la rebaja por antigüedad en ninguna de las reservas asociadas al bono. Cada bono tendrá 5 sesiones y una fecha de caducidad de un año desde la primera reserva.
- El precio de una reserva se establece en función de la duración de esta: 60 minutos (20€), 90 minutos (30€), 120 minutos (40€).
- Las reservas pueden realizarse, modificarse y/o cancelarse hasta 24h antes de la hora de inicio.
- El gestor debe permitir consultar el número de reservas futuras, esto es, para cualquier fecha posterior a la actual.
- El gestor debe permitir consultar las reservas que existen para un día y una pista concretos.

Para probar la funcionalidad de los distintos gestores, se desarrollará un programa Java con interfaz texto (E/S por consola) que ofrezca menús para acceder a las distintas funcionalidades relacionadas con usuarios, pistas y reservas. Se recomienda la organización de diferentes menús de opciones según el tipo de gestor, así como de cargar previamente información desde uno o varios ficheros de texto. Estos ficheros deberán quedar actualizados al terminar el programa, pudiendo trabajar con listas durante la ejecución del programa.

Estructure adecuadamente el código para separar las clases relacionadas con los datos, las operaciones de gestión de esos datos, y la interfaz de usuario por consola. Las rutas a los ficheros de datos deberán estar indicadas en un fichero de propiedades dentro del proyecto Eclipse, evitando el uso de rutas absolutas. Al fichero de propiedades se accederá desde el programa Java haciendo uso de la clase *java.util.Properties* (ejemplo disponible en Moodle). Para más información:

- Clase: https://docs.oracle.com/javase/8/docs/api/java/util/Properties.html
- Tutorial: https://www.baeldung.com/java-properties

Para la implementación de la entrada de datos por consola se recomienda utilizar la clase *java.util.Scanner*. https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html

Recuerde que aparte del código fuente, debe entregarse el programa como ejecutables JAR. Incluya además un fichero texto README.txt en el que se explique los parámetros requeridos para la ejecución del programa. Tutorial: https://www.baeldung.com/java-create-jar