Programación Orientada a Objetos
Curso 2023/2024
Sesión 7

El ejercicio consiste en la implementación de una aplicación para que los propietarios de un coche puedan compartir las plazas de su coche cuando vayan a realizar un viaje.

Previo

Para la realización del ejercicio es necesario utilizar fechas y gestionar el tiempo. A continuación, se dan algunas indicaciones:

- La clase `java.time.LocalDate` representa fechas como **objetos inmutables**. El método de clase `now()` devuelve la fecha actual.
- La clase `LocalDate` dispone del método de clase `of` que permite crear una fecha pasando como parámetro el año, el mes y el día, en este orden.
- La clase `LocalDate` dispone de los métodos `isAfer`, `isBefore` e `isEqual` para comparar fechas.

Ejercicio 1. Jerarquía de viajes

Este primer ejercicio consiste en la implementación de una jerarquía que representa distintos tipos de viajes en los que se pueden hacer reservas.

Antes de presentar el concepto viaje, se introduce el concepto que representa la reserva de un viaje. Una **reserva** corresponde con la reserva de un número de las plazas ofertadas en un viaje. Se caracteriza por las siguientes propiedades:

- Código de reserva: una cadena de texto generada automáticamente en el constructor de forma aleatoria utilizando `UUID.randomUUID().toString();`
- Usuario: cadena de texto que identifica al usuario que realiza la reserva.
- Número de plazas: entero que representa el número de plazas que se han reservado.
- Fecha en la que se realiza la reserva. Esta propiedad se inicializa en la construcción y corresponde a la fecha actual (`now`).

El constructor de la reserva recibe como parámetro el usuario y el número de plazas. Además, las propiedades no se pueden modificar tras la construcción. Por tanto, esta clase define objetos inmutables.

Un **viaje** representa una oferta que hace un propietario para compartir su coche. Las propiedades que caracterizan a un viaje son:

- Propietario del vehículo (cadena de texto).
- Coche: cadena de texto con el modelo del coche utilizado para el viaje.
- Ruta: cadena que describe el viaje (por ejemplo, “Murcia – Valencia – Barcelona”).
- Fecha de salida.
- Plazas ofrecidas.
- Reservas: lista con las reservas que se han realizado.
- Número de plazas reservadas. Corresponde con la suma del número de plazas de las reservas realizadas.
- Plazas disponibles. Corresponde con la diferencia entre las plazas ofrecidas y el número de plazas reservadas.

Las propiedades propietario, coche, ruta, fecha de salida y número de plazas ofrecidas no se pueden modificar una vez establecidas en la construcción. Las reservas serán añadidas a través de un método (se describe más adelante). Inicialmente la lista de reservas es vacía. En el constructor se establece el propietario, el coche, la ruta, la fecha de salida y las plazas ofrecidas. También se ofrece un segundo constructor en el que se omite el número de plazas ofrecidas tomando como valor por defecto 1.

La funcionalidad que ofrece la clase viaje es:

- *realizar reserva*: el método recibe como parámetro el usuario (cadena de texto) que realiza la reserva y el número de plazas que quiere reservar. Para poder formalizar la reserva se tiene que cumplir que: queden plazas disponibles y que el viaje no se haya realizado todavía (la fecha en la que se está realizando la reserva tiene que ser anterior a la fecha de salida). Si se cumplen estas restricciones, se generará un objeto reserva que se almacenará en la lista de reservas y que además se devuelve como resultado de la ejecución. En el caso de que no se cumpla alguna de las condiciones el método devolverá el valor nulo (`null`).
- Consultar la reserva asociada a un código: el método recibe como parámetro un código de reserva y devuelve el objeto reserva cuyo código es igual al recibido como parámetro, o `null` si no existe ninguna reserva con ese código.

Además, se definen dos **tipos de viaje especiales**:

- **Viaje premium**. Es un tipo de viaje que se caracteriza por permitir cancelar las reservas hasta el día antes de la fecha de salida. Para realizar la cancelación se proporciona el código de la reserva y devuelve un valor booleano indicando si se ha efectuado la cancelación.
- **Viaje selectivo**. Es un tipo de viaje que permite vetar a los usuarios. Por tanto, este tipo de viaje añade una nueva propiedad que contiene el conjunto de usuarios vetados. Inicialmente la colección estará vacía, pero se pueden añadir y eliminar usuarios vetados en cualquier momento. Por tanto, no se aceptará una reserva si el usuario que la realiza está vetado.

Implementa el siguiente **programa** para probar la funcionalidad:

- Crea los siguientes viajes cuyo propietario sea “José Antonio” y coche “Seat León”:
  - Viaje “Murcia-Cartagena” con fecha de salida 9/dic/2023 con el número de plazas por defecto.
  - Viaje selectivo “Murcia–Campus” con fecha de salida 11/dic/2023 y 4 plazas.
  - Vetar a “Beatriz” en el viaje selectivo.
  - Viaje premium “Murcia-Barcelona” con fecha de salida 15/dic/2023 y 6 plazas.
- Realiza las siguientes reservas y muestra el resultado de cada paso en la consola:
  - "Alberto" hace una reserva de dos plazas en el viaje “Murcia-Cartagena”. El resultado debe ser `null` porque el número de plazas ofertadas es 1.
  - "Enrique" hace una reserva de 3 plazas en el viaje “Murcia-Campus”.
  - "Beatriz" hace una reserva de 1 plaza en el viaje “Murcia-Campus”. El resultado debe ser `null`, quedan plazas disponibles, pero “Beatriz” está vetada.
  - "Ana" hace una reserva de dos plazas en el viaje “Murcia-Barcelona”.
- “Ana” cancela su reserva.

Ejercicio 2. Método `toString`

Implementa el método `toString` de la clase `Object` en las clases que implementan los viajes siguiendo las recomendaciones de la asignatura. Recuerda que:

- El método `toString` de la clase `Viaje` debe mostrar la información de todas las propiedades, incluidas las reservas. Por ello se debe implementar el método `toString` en la clase `Reserva`.
- `toString` sólo deben redefinirse en aquellas clases de la jerarquía que añadan nuevas propiedades, refinando en este caso la implementación de la clase padre.

Modifica el **programa** y añade la siguiente funcionalidad al final del método `main`:

- Crea una lista con los tres viajes anteriores.
- Recorre la lista de viajes:
    - Si el viaje es selectivo, quitar el veto a “Beatriz”.
    - Imprime la información (`toString`) de cada viaje.

Ejercicio 3. Método `clone`

Implementa el método `clone` de la clase `Object` en las clases que implementan los viajes siguiendo las recomendaciones de la asignatura y según la semántica de los tipos de datos. En la implementación del método `clone` se implementará una **copia profunda**.

Modifica el **programa** y añade la funcionalidad:

- Declara y construye una segunda lista de viajes (*copias*).
- Recorre la lista de *viajes* y para cada elemento:
    - Crea una copia del viaje y añádela a la lista *copias*.
    - Si el viaje es Premium, cancela todas las reservas en el objeto copia.
- Muestra el contenido de la lista *copias* por la consola (`toString`).