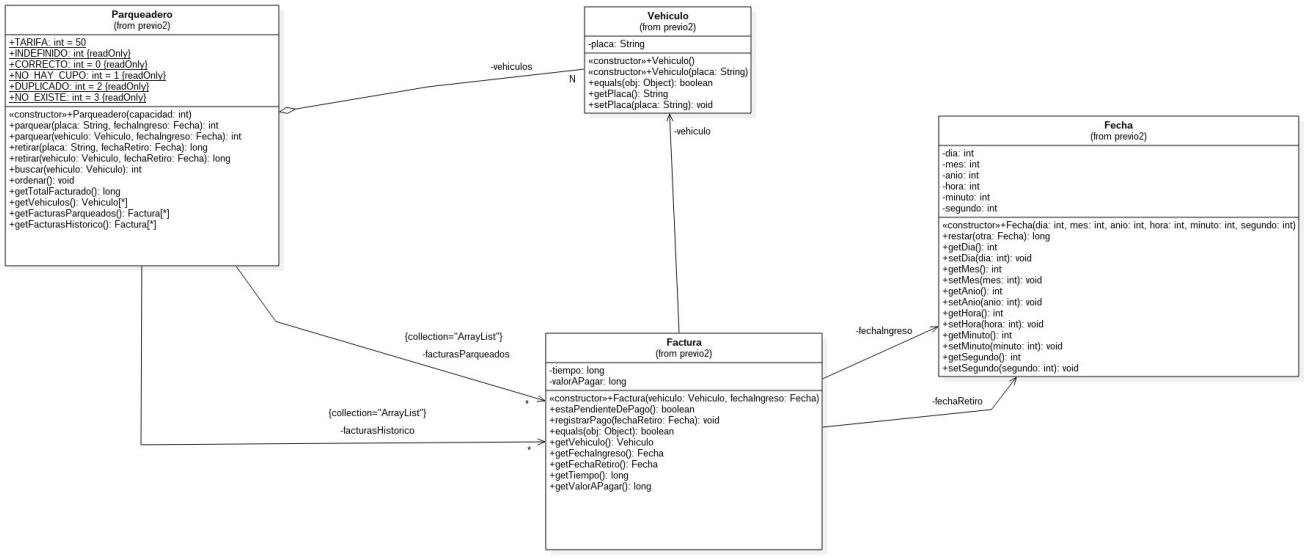
Segundo Previo Programación Orientada a Objetos I – POO I
Profesor Milton Jesús Vera Contreras miltonjesusvc@ufps.edu.co
2023-II

El previo está organizado en dos partes:

1. Problema en VPL, valor 70% del previo. El resultado en VPL corresponde al 60% y 40% la revisión del profesor. Esta parte debe resolverse durante las dos (2) horas del previo. El profesor podrá ofrecer la posibilidad de recuperar, dependiendo de los resultados del curso, pero lo ideal es que logre resolverse en las dos (2) horas.
2. Interfaz gráfica GUI en JavaFX, valor 30% del previo. Deberá entregarse al finalizar la semana, el viernes 3 de junio a media noche. El estudiante debe proponer la GUI como mejor considere y será muy bien valorada la apariencia gráfica y sencillez de la GUI, además de la estructura del código y el uso correcto de MVC.

Se quiere desarrollar una aplicación para apoyar el registro de parqueo, retiro y facturación de un Parqueadero. El siguiente diagrama de UML corresponde al diseño propuesto:



Se proporciona la plantilla de código fuente, a la cual no se pueden agregar ni clases, ni propiedades, ni métodos. Solo se requiere programar cada método.

Para todas las clases se deben implementar sus métodos constructores y los métodos GET/SET que correspondan, según el diagrama en UML.

La clase `Fecha` debe usar la clase `GregorianCalendar` de la API de Java
https://docs.oracle.com/en/java/javase/17/docs/api//java.base/java/util/GregorianCalendar.html
Esta clase tiene un `constructor` que recibe todos los datos y un método *`getTimeInMillis()`*, que retorna el tiempo en milisegundos, con lo cual se puede simplificar la resta de fechas.

La clase `Vehiculo` debe implementar el método equals, para determinar si dos vehículos son iguales, tienen la misma placa.

La clase `Factura` debe implementar el método equals. Dos facturas se consideran iguales cuando sus vehículos son iguales. Además, se deben implementar los métodos `estaPendienteDePago` y `registrarPago`. El primer método debe informar si aún no se ha retirado el carro, es decir, no hay fecha de retiro. Puesto que no se permite salir sin pagar, el pago está pendiente. El segundo método debe registrar la fecha de retiro, calcular el tiempo y el valor a pagar.

La clase `Parqueadero` se inicializa con una capacidad máxima de vehículos. Si dicha capacidad se llena, no se permite el ingreso de vehículos. Esta clase debe implementar los siguientes métodos:

- Dos métodos `parquear`. Uno que recibe una cadena con la placa del vehículo y otro que recibe un objeto vehículo ya creado. Ambos métodos reciben como segundo parámetro la fecha de ingreso. Los vehículos se parquean en el primer espacio libre que encuentran. Al parquear se pueden presentar varias situaciones, para lo cual los métodos parquear debe retornar lo siguiente:
  - CORRECTO si pudo parquear. En este caso el vehículo se guarda en la lista de `vehículos`, que es la primera relación en el diagrama UML. Además, se crea la `Factura` y se guarda, para cobrar posteriormente, para lo cual se tiene la lista `facturasParqueados`, que es la segunda relación del diagrama UML.
  - NO_HAY_CUPO si no hay espacios vacíos para parquear, está lleno el parqueadero.
  - DUPLICADO si se intenta parquear un carro con la misma placa de otro carro ya parqueado.
  - Estos tres valores son, respectivamente, 0, 1 y 2. Pero se usan constantes por buena práctica de codificación.
- Dos métodos `retirar`. Uno que recibe una cadena con la placa del vehículo y otro que recibe un objeto vehículo ya creado. Ambos métodos reciben como segundo parámetro la fecha de retiro. Los vehículos dejan espacios vacíos al retirarse y, como en la vida real, nadie organiza el parqueadero. En esos espacios vacíos, posteriormente, parquearán otros vehículos. Al retirar un vehículos se puden presentar dos situaciones, para lo cual los métodos retirar deben retorna lo siguiente:
  - INDEFINIDO si el vehículo no se encuentra en el parqueadero.
  - El valor a pagar si el vehículo estaba parqueado. En este caso debe eliminarse el vehículo, dejando el espacio libre para otro cliente. Además, la factura que se creó al parquear debe eliminarse la lista `facturasParqueados` y almacenarse en la lista `facturasHistorico`, con lo cual se logra tener un control apropiado del negocio. El valor a pagar se calcula multiplicando el tiempo de parquéo en minutos por el valor de la `TARIFA`. El tiempo se calcula restanto las fechas de ingreso y retiro. La `TARIFA` puede cambiar en cualquier momento.
- Un método `buscar`, que permite determinar si un vehículo se encuentra parqueado. Si lo encuentra, regresa la posición en que se encuentra, de lo contrario regresa INDEFINIDO.
- Un método `getTotalFacturado`, que regresa el total de dinero que ha facturado el parqueadero.
- Un metodo `ordenar`. Puesto que, los vehículos dejan espacios vacíos al retirarse y, como en la vida real, nadie organiza el parqueadero, el dueño del parqueadero solicitó una funcionalidad especial, que orden los carros del parqueadero, dejando todos los espacios vacíos al final. En la vida real, esta tarea la hará una persona encargada de custodiar los vehículos. Dicha persona ubicará todos los carros de manera adyacente, manteniendo el orden que tenían y dejando los

espacios al final, para que sea más sencillo parquear. En el software, usted debe implementar esta funcionalidad.

Es obligatorio que se usen todos los métodos mencionados anteriormente, excepto los métodos get, los cuales son usados desde el Controller.

Para cada método se diseñaron pruebas que se detallan a continuación con el peso que tienen en la calificación:

Método	Peso en porcentaje %
testEqualsVehiculo	3
testConstructorFactura	3
testEqualsFactura	3
testRestarFechas	5
testPagarFactura	3
testConstructorParqueadero	5
testParquear1 (parámetro Vehiculo)	10
testRetirar1 (parámetro Vehiculo)	10
testParquear2 (parámetro String)	5
testRetirar2 (parámetro String)	5
testRetirarParquear (test integral de ambos)	15
testBuscar	5
testTotalFacturado	5
testTarifa (la tarifa puede cambiar)	3
testOrdenar	10
testFull (un test completo con datos aleatorios)	10
Total	100

Quien logre un buen resultado en este previo, se puede considerar un Aprendiz de Desarrollador Junior ;) ¡Muchos éxitos!