**Proyecto del Segundo Cuatrimestre Fundamentos de Programación (Curso 2021/2022)**

Autor: Mohanad JD Abulatifa

uvus: mohabu2

## Estructura de las carpetas del proyecto

* **/src**: Contiene los diferentes archivos que forman parte del proyecto.

está estructurado en los siguientes paquetes .

* + **fp.Coches** : Paquete que contiene los tipos del proyecto.
  + **fp.Coches.test**: Paquete que contiene las clases de test del proyecto.
  + **fp.common**: Paquete que contiene los tipos auxiliares del proyecto
  + **fp.utiles**: Paquete que contiene las clases de utilidad.
* **/data**: Contiene el dataset del proyecto
  + **<carsclean.csv>**: carsclean es un dataset contiene a informaciones sobre coches en venta en ciudad AL RIYADH (la marca , modelo , precio , etc) .

## Estructura del dataset

Coches(String company, String model, LocalDate productionDate , Condition condition, Integer kilometers , String color , PaymentMethod paymentMethod , Integer price)

* **company**: de tipo String, representa el nombre de la compañía del coche
* **model**: de tipo String, representa el modelo del coche en el dataset
* **productionDate**: de tipo LocalDate, representa la fecha de fabricación del coche
* **condition**: de tipo Enumerate , representa la condición actual del coche y son dos condiciones **(1 NEW 2( USED**
* **color**: de tipo String, representa el color del coche
* **PaymentMethod**: de tipo Enumerate, representa la manera de pago permitida y hay tres maneras **1) CashOrInstallments 2) Cash 3) Installments .**
* **price**: de tipo Integer, representa el precio del coche.

## Tipos implementados

### ***Tipo Base Coches***

Descripción breve del tipo base Coches.java es información sobre coches en venta en Al- Riyadh describiendo sus propiedades y haciendo unas operaciones usando la información.

**Propiedades**:

* **company**: de tipo String ,consultable.
* **model**: de tipo String ,consultable .
* **productionDate**: de tipo LocalDate, consultable.
* **condition**: de tipo Enumerate , consultable .
* **color**: de tipo String, consultable .
* **PaymentMethod**: de tipo Enumerate, consultable .
* **price**: de tipo Integer, consultable .

**) Todos de tipo consultable y no modificable porqué es un record ( .**

**Constructores**:

* C1: Tiene un parámetro por cada propiedad básica del tipo.
* C2: Crea un objeto de tipo Coche a partir de los siguientes parámetros: String company, String model, LocalDate productionDate , Condition condition, Integer kilometers , String color , PaymentMethod paymentMethod , Integer price

**Restricciones**:

* R1: Comprueba que el precio del Coches no es un número negativo.
* R2: Comprueba que los kilómetros circulados del Coches no es un número negativo ....
* R3: Comprueba que la fecha de fabricación del Coches no es una fecha a partir del año 2022 .

**Criterio de ordenación**: El método responsable del mecanismo de la ordenación, ordena primero dependiendo a sus precios y si el precio de dos coches es igual lo comprueba por los kilometros circulados .

**Representación como cadena :** El método hashCode sirve para obtener un código hash que sería como un identificador del objeto.Este hash se utiliza en algunas colecciones como (HashSet, HashMap, LinkedHashSet, LinkedHashMap, ConcurrentHashMap, etc ).

El método equals sirve comprobar si los objetos son iguales o distintos.

y aquí dos objetos son iguales si tienen el mismo precio y el mismo numero de kilómetros circulados.

**Otras operaciones**:

* método 1: averageKilometersPerYear() , calcula la media de los kilómetros circulados por año
* método 2: getPriceRating() Segundo métedo calcula la clasificación del precio del coche si Se clasifica como un coche **cheap , expensive**

#### Tipos auxiliares

* **condition**: enumerado. Puede tomar los valores **1) New 2( Used**
* **PaymentMethod**, enumerado. Puede tomar los valores **1) CashOrInstallments 2) Cash 3) Installments.**
* **PriceRating** , enumerado. Puede tomar los valores 1) cheap 2) expensive

**Factoría – Factoria**

Clase de factoría para construir objetos de tipo Car.

Car parse(String s) : recibe como parámetro una cadena con el formato de las líneas del fichero carsclean.csv ,Con este método, convertimos una cadena de caracteres, un String, a cada tipo de datos que tenemos (String , LocalDate , enum , Integer , String, enum , Integer ) .

String[] splits = s.split(";") se usa para dividir un string en substrings mediante una expresión y así nuestro String se divide cada vez que que sale (;) y se crean 8 Strings que son nuestras propiedades y usamos trim() para la manipulación de cadenas (String), el cual sirve para quitar los espacios a la cadena, Y devolvemos un objeto del tipo Car .

leeDeFichero(String ruta) : Toma como entrada el parámetros de tipo String (ruta) que referencia a la ruta del csv de nuestro proyecto y creamos una lista vacia para almacenar los datos dentro de ella luego hacemos uso del fichero (Ficheros) ,quitamos la cabecera , leemos el csv corriendo todas las líneas haciendo uso al métedo parse y al final devolvemos la lista de objetos para hacer sobre ella nuestros tratamientos.

### Tipo Contenedor – CarsRiyadh

Clase contenedora de los objetos de tipo Cars.

**Propiedades**:

* *Cars*, de tipo *List<Car>*, consultable. Lista de Coches.

**Constructors** :

C1: Constructor por defecto. Crea un objeto de tipo Cars sin ningún coche almacenada.

C2: Constructor con un parámetro de tipo List<Cars>. Crea un objeto de tipo Cars con los coches incluidas en la lista dada como parámetro.

**Criterio de igualdad**: Dos coches son iguales si lo son sus listas de coches.

**Propiedades derivadas:**

* **getNumeroCars() :** obtener el número de coches en nuestra lista
* **addElement() :** añadir un elemento a la lista
* **addListOfElements(List<Car> c) :** añadir una lista de elementos a la lista
* **deleteElement(Car c)** : eliminar un elemento de la lista

**Tratamientos secuenciales:**

* **ExisteCocheCompaniaYmodelo(String f , String m):** Toma como entrada dos parámetros de tipo String ( f , m ) que referencian al nombre de la marca del coche y su modelo y devuelve como salida un Booleano de (true ó false) si existe un coche con las características indicadas con la misma marca y el mismo modelo dados como parámetros .
* **ElCocheMasBarato():** No toma como entrada algún parámetro y devuelve como salida una Lista del tipo Car para el coche que tiene el precio mínimo , en otras palabras , devuelve lista con las propiedades del coche que tiene el precio más bajo .
* **GetCocheCondicion():**Toma como entrada un parámetro de tipo Booleano ( c ) que referencia al enumerado de la condición del coche que tienen como valores ( Used , New ) y devuelve como salida una lista de los coches que cumplen la condición indicada y dada como parámetro.
* **AgrupacionDeCochesPorFecha():**No toma como entrada algún parámetro devuelva un Map en el que las claves sean el año en el que se ha fabricado el coche y los valores una lista de coches que corresponden a la clave .
* **AcumulacionDeCochesPorCompania():**No toma como entrada algún parámetro devuelva un Map en el que las claves sean las marcas de todos los coches los valores un Integer que hace referencia al número total de coches que hay de cada marca.

**Test Tipos :**

**Clase main para acceder a los datos y comprobar las funcionalidades sobre ellos.**

1. **TestCar :** 
   1. **testea las restricciones que hemos puesto para nuestro tipo.**
   2. **testea el equal para comprobar si los tipos creados son iguales o no.**
   3. **testea la operación derivada AverageKilometersPerYear().**
   4. **testea la operación derivada priceRating().**
   5. **crea varios objetos del tipo Car y los agrega a una lista luego les ordena para testear el compareTo.**
2. **TestCarsRiyadh :** 
   1. **Testea la lectura de datos para hacer todas las operaciones sobre ella.**
   2. **Muestra el número de datos que están en la lista .**
   3. **Testea los siguientes tratamientos secuenciales :**
      1. **ExisteCocheCompaniaYmodelo()**
      2. **ElCocheMasBarato()**
      3. **GetCocheCondicion()**
      4. **agrupacionDeCochesPorFecha()**
      5. **acumulacionDeCochesPorCompania()**
   4. **dos funciones auxiliares para mejor lectura de los maps**
      1. **mostrarDiccionarioAgrupaciones(Map<Integer> , List<Car> map)**
      2. **mostrarDiccionarioTotal(Map<String , Integer>map)**