1 º Daw. Programación.

Colecciones, serialización y ficheros. Examen 2

Fecha: 23-mayo-2023

Nos piden, desde una empresa de alquiler de vehículos, que realicemos un prototipo de gestión de su empresa, centrándonos inicialmente en los siguientes procesos a programar:

Ejercicio 1. Realiza una clase 'Vehiculo', que tenga como atributos marca, modelo, kilometraje (número inicializado a cero que sólo puede incrementarse en positivo), matrícula, precio (euros por km recorrido, es un float), tipo (será un valor entre los siguientes: moto, coche, camioneta y autocaravana) y disponible (será un boolean que se inicializa a true). Tendrá los siguientes métodos:

- Constructor basado en todos los parámetros, pero teniendo en cuenta que si kilometraje no es 0 o mayor, se asignará a 0. Disponible se deberá inicializar basándose en el parámetro que nos den.
- Los getters y los setters. Tener en cuenta que en void setKilometraje(int km) lo que haremos es siempre comprobar que km es positivo y sumarlo a lo que llevamos.
- Tendremos un método abstracto: abstract public float facturaAlquiler(int kms);
- El método toString (override de Object) para mostrar el siguiente formato CSV:
  VEHICULO:marca;modelo;kilometraje;matricula;precio;disponible;tipo
- Hay que tener en cuenta que VEHICULO es una cadena de texto no variable, siempre se pone así literal y que el separador entre VEHICULO y los datos es el caracter ":", mientras que el separador de los datos es ";".

Ejercicio 2. Realiza las clases Moto, Coche, Limusina, Camioneta y Autocaravana, que hereden de Vehiculo, con las siguientes condiciones por cada clase:

- Una moto tendrá un atributo cilindrada, número, que indicará los cc del motor de la moto.
- Un coche tendrá un atributo pasajeros, número, irá entre 2 y 7, ni más, ni menos. En caso de recibir un valor erróneo en el constructor, se asignará 5, que es el valor por defecto. Si en el setter nos dan un valor erróneo, no se hace nada.
- Una camioneta tendrá un atributo peso máximo, que será un valor entre 3500 y 7500 (miden los kgs máximo transportados incluyendo el peso del vehículo).
- Una autocaravana tendrá un atributo camas disponibles, que determina el número de pasajeros máximo a viajar, y que irá desde 1 hasta 6, incluidos.
- Los getters y los setters: deben de controlar los valores límite en caso de existir.
- Todas las clases hijas deben de implementar el método public abstract float facturaAlquiler(int kms) multiplicando el precio por el parámetro kms, pero teniendo en cuenta las siguientes condiciones:
  - Las motos de 500cc o más llevan un recargo del 20%
  - Los coches de 6 y 7 plazas llevan un recago del 10%
  - Las camionetas de más de 5000kg de peso máximo llevan un recargo del 30%

    - Las autocaravanas de 5 o más plazas llevan un recargo del 15%.
- El método toString (override de Vehiculo) mostrará el siguiente formato CSV:
  VEHICULO:marca;modelo;kilometraje;matricula;precio;disponible;tipo;atributoEspecifico
- Hay que tener en cuenta que tipo nos valdrá para determinar qué tipo de vehículo está siendo codificado en CSV. El atributoEspecifico será el determinado por la clase hija concreta que se está programando.

Ejercicio 3. Realiza una clase Flota, con el atributo String nombreGerente, String dniGerente y String emailGerente, que será una colección de vehículos almacenada en un mapa de tipo HashMap<String,Vehiculo> y que tendrá los siguientes métodos:

- Constructor que inicializa el mapa con cero vehículos, y recoge el nombre, dni y email del gerente. Getters y setters de los atributos String pero NO DEL MAPA (IMPORTANTE).
- Método toString: Escribirá los datos en formato CSV de nombre, dni y email del gerente. NO DEL MAPA. Ejemplo:
  Juan Fernández Montes;11223344T;jfm@gmail.com
- public Moto[] motosDisponibles(): Devolverá un array con las motos disponibles en nuestro sistema. No estáran las que estén alquiladas, sólo las disponibles. Ordenadas por cilindrada.
- public Coche[] cochesDisponibles(): Lo mismo que el anterior para motos, pero con coches, ordenados por precio.
- public Camioneta[] camionetasDisponibles(): Lo mismo que las anteriores pero para camionetas, ordenadas por peso máximo.
- public Autocaravana[] autocaravanasDisponibles(): Lo mismo que las anteriores pero para autocaravanas, ordenadas por camas disponibles.
- public Vehiculo crearVehiculo(String[] datosAtributos, String tipo): Método que creará una Moto, Coche, Camioneta o Autocaravana dependiendo de lo especificado en el parámetro tipo. Si nos pasan algo erróneo, devolvemos null. Los atributos del objeto a crear vendrán en un String[] donde el orden de los atributos se corresponde EXACTAMENTE con el orden CSV producido en el método toString de la clase correspondiente (siempre deben de ser 8 valores, y el atributo en el penúltimo String del array datosAtributos debe de ser igual al parámetro tipo).
- public void putVehiculo(Vehiculo v): Añadirá el vehículo al mapa. Si ya existía nos quedamos en el mapa con el objeto v, el que estuviera previamente desaparece.
- public float facturarVehiculo(String matricula, int kms): Debe de obtener el vehículo identificado por la matrícula determinada, actualizar en este vehículo los kms de kilometraje, poner el vehiculo como disponible a true y devolver el precio a cobrar por estos kms a facturar.
- public boolean alquilarVehiculo(String matricula): Debe de obtener el vehículo determinado y ponerlo como no disponible. Devolverá false si el vehículo no existe y true si todo va bien.

Ejercicio 4. Realiza la clase ControladorVehiculos, que tenga los métodos static:

- public static int leerDeFichero(String filename, Flota miFlota): Debe de abrir el fichero determinado y leer cada uno de los vehículos almacenados en el formato CSV que encuentre en ese fichero, instanciarlos y añadirlos al mapa del objeto “miFlota” utilizando el método crearVehiculo y putVehiculo. No tiene por qué controlar duplicados, sí que las líneas tienen el formato CSV adecuado para no “romper” la

ejecución si encontramos una línea con formato inadecuado (puedes usar excepciones o lógica de control). Devolverá el número de líneas que se han podido añadir al mapa con éxito.

- public static void grabarAFichero(String filename, Flota miFlota): Debe de “volcar” todo nuestro mapa al fichero determinado, grabando las motos, los coches, las camionetas y las autocaravanas (se pueden usar los métodos de Flota para obtener los distintos vehículos).

NOTA: Escribe el método Main con la secuencia, menús, llamadas, lo que consideres para probar Flota, Vehiculo, sus clases hijas y la lectura/escritura a fichero. No tienes por que hacer menús, puedes hacer una secuencia lineal con datos creados de manera literal y puedes crear con el bloc de notas (o el IDE) los ficheros de texto a “mano” para probar tu programa, al “estilo” de lo que hacemos cuando usamos JUNIT (no significa que se deba de usar JUNIT). La ejecución o prueba de vuestro código queda a vuestra elección, por lo tanto.

Referencias necesarias de API java a utilizar en el examen:

Map:

Value get(Key k): Devuelve el objeto value asociado a key. Null si no se encuentra.
Value put(Key k, Value v): Asigna a la clave k el valor v. Se devuelve el objeto que estuviera asignado a k previamente o null si no estaba.
Value remove(Key k): Borra el objeto asociado a k, y lo devuelve. Si no estaba, null.
Collection<V> values(): Devuelve la colección de valores almacenados en el mapa.

Collection:

Object[ ] toArray(): Devuelve un array con los objetos de la colección.
Collections.sort(List<E> l): ordena la lista basada en orden natural. Implementación en E de comparable.
Collections.sort(List<E> l, Comparator<E> c): ordena la lista basada en la comparación resultado de usar Comparator c.

ArrayList:

boolean addAll(Collection<E> c): Añade todos los objetos de la colección al ArrayList
void sort(Comparator<E> comp): Ordena los objetos del arraylist usando el comparador comp.

Comparable:

Interfaz con el método *int compareTo(Tipo t)*

Comparator:

Interfaz para crear objetos con el método *int compare(Tipo t1, Tipo t2)*

File:

FileReader: Lectura de ficheros de texto. Se pasa nombre del fichero en el constructor. Se proporciona después a un Scanner para facilitar la lectura.
FileWriter: Escritura de ficheros de texto. Se pasa nombre del fichero en el constructor y un boolean que implica si false se sobreescribe, true se añade al final.

PrintWriter: Se inicializa en el constructor con un FileWriter. Es un ayudante para mejorar la eficiencia y usar formato equivalente a si tuviéramos el System.out

Evaluación (Máx 100 puntos, aprobado con 50 puntos):

- Ejercicio 1: (6 puntos en total)
    - Atributos: 1p
    - Constructor: 1p
    - Los getters y los setters: 1p
    - abstract public float facturaAlquiler(int kms): 1p
    - El método toString: 2p
- Ejercicio 2. Por cada clase hija (4 x 7p = 28 puntos en total)
    - Atributos: 1p
    - Constructor: 1p
    - Los getters y los setters: 1p.
    - float facturaAlquiler(int kms): 2p
    - El método toString: 2p
- Ejercicio 3. (40 puntos en total)
    - Atributos: 1p
    - Constructor: 2p
    - Método toString: 1p
    - public Moto[] motosDisponibles(): 4p
    - public Coche[] cochesDisponibles(): 4p
    - public Camioneta[] camionetasDisponibles(): 4p
    - public Autocaravana[] autocaravanasDisponibles(): 4p
    - public Vehiculo crearVehiculo(String[] datosAtributos, String tipo): 10p
    - public void putVehiculo(Vehiculo v): 2p
    - public float facturarVehiculo(String matricula, int kms): 6p
    - public boolean alquilarVehiculo(String matricula): 2p
- Ejercicio 4. (26 puntos en total)
    - leer desde el fichero: 16p
    - escribir a fichero: 10p

NOTA: Se tendrá en cuenta para cada ejercicio:

- Los métodos pueden estar bien o mal. Si un método tiene errores, no se puntuará.
- Los atributos tienen que ser los definidos en el enunciado, con los tipos de datos del enunciado para ser valorados.