ADSOF: Segundo examen parcial - 06/05/2020

Ejercicio 1 (3 puntos)

Te han encargado el diseño de un sistema de información para un taller de coches. Para ello, necesitas crear una clase `GarageHistory` que pueda almacenar las averías de los vehículos. `GarageHistory` debe ser altamente reutilizable, con cualquier clase que implemente información sobre vehículos, y con cualquier clase de avería (breakdown) que permita realizar las siguientes acciones:

```
interface IBreakdownInfo {
        String info();
        void set(boolean v);
}
```

Donde `info()` devuelve la descripción de la avería, y `set()` permite indicar si la avería es actual (true) o pasada (false).

Se pide completar el siguiente programa para que produzca la salida indicada más abajo. En particular, nota que:

1. `GarageHistory` mantiene sus entradas ordenadas (alfabéticamente por la matrícula del vehículo en este caso)
2. Las averías asociadas a un vehículo se mantienen en orden de inserción y no admiten repeticiones, donde una avería se considera repetida si tiene igual `info()`.
3. Si se tratar de añadir un vehículo repetido a `GarageHistory`, esta clase lanza una excepción (en este caso, no tenemos en cuenta el modelo del vehículo, sino sólo su número de matrícula).

Se valorará especialmente el uso de principios de orientación a objetos en el diseño, así como su generalidad, reusabilidad y extensibilidad.

```
class CarBreakdown /*…completar si es necesario */{      // CarBreakdown y Car se las daremos en la plantilla
  private String problem;
  private boolean active;
  public CarBreakdown (String issue, boolean active) {
        this.problem = issue;
        this.active = active;
  }
  /*…completar si es necesario */
}

class Car /*…completar si es necesario */{
  private String model;
  private String plate;  // matricula
  public Car (String m, String plate) {
        this.model = m;
        this.plate = plate;
  }
  public String toString() { return this.plate; }
  /*…completar si es necesario */
}

public class GarageInformationSystem {
  public static void main(String[] args) {
    GarageHistory<Car,CarBreakdown> s = new GarageHistory<>();
    Car ford = new Car("Ford fiesta", "5555ABC");

    s.addCar(ford).
      addAll(Arrays.asList(new CarBreakdown("Flat tire", true),          // problema (pinchazo) y está activo
                           new CarBreakdown("Broken left door", false),  // puerta rota, ya arreglada
                           new CarBreakdown("Broken left door", true))); // se considera avería repetida, y no se añade

    System.out.println(s);                       // Imprime el histórico del garaje
    if (s.fix(ford, "Flat tire"))                // fix devuelve true si la avería para el coche existe
        System.out.println(ford+" was fixed its flat tire");
    s.addCar(new Car("Citroen CX", "4444CBA"));
    System.out.println(s);

    System.out.println(s.getBreakdown(ford));    // Imprime el registro de averías del coche

    s.addCar(new Car("Renault Clio", "5555ABC"));// Lanza una excepcion, ya que la matrícula está en el sistema
  }
}
```

Salida esperada:

```
{5555ABC=[Flat tire:  ongoing, Broken left door:  fixed]}
5555ABC was fixed its flat tire
{4444CBA=[], 5555ABC=[Flat tire:  fixed, Broken left door:  fixed]}
[Flat tire:  fixed, Broken left door:  fixed]
Exception in thread "main" carGarage.RepeatedCar: Repeated car: 5555ABC
        at carGarage.GarageHistory.addCar(GarageInformationSystem.java:20)
        at carGarage.GarageInformationSystem.main(GarageInformationSystem.java:114)
```

Solución:

```
class RepeatedCar extends RuntimeException {
  public RepeatedCar(String recordName) {
        super ("Repeated car: "+recordName);
  }
}

interface IBreakdownInfo {
        String info();
        void set(boolean v);
}

class GarageHistory<K extends Comparable<K>, V extends IBreakdownInfo> {
        private TreeMap<K, Set<V>> cars = new TreeMap<>();

        public Set<V> addCar(K p) {
                if (this.cars.containsKey(p)) throw new RepeatedCar(p.toString());
                Set<V> records = new LinkedHashSet<>();
                this.cars.put(p, records);
                return records;
        }
        public Set<V> getBreakdown(K p) { return this.cars.get(p); }
        public boolean fix(K p, String issue) {
                if (!this.cars.containsKey(p)) return false;
                for (V record : this.cars.get(p)) {
                        if (record.info().equals(issue)) {
                                record.set(false);
                                return true;
                        }
                }
                return false;
        }
        @Override public String toString() { return this.cars.toString(); }
}

class CarBreakdown implements IBreakdownInfo{
        private String issue;
        private boolean active;
        public CarBreakdown(String issue, boolean fixed) {
                this.issue = issue;
                this.active = fixed;
        }
        public boolean equals(Object o) {
                if (o==this) return true;
                if (!(o instanceof CarBreakdown)) return false;
                CarBreakdown hi = (CarBreakdown) o;
                return this.issue.equals(hi.issue);
        }
        public int hashCode() { return this.issue.hashCode();}

        @Override public String toString() {
                return this.issue + ": "+ (active ? " ongoing" : " fixed");
        }
        @Override public String info() { return this.issue; }

        @Override public void set(boolean v) { this.active = v; }
}

class Car implements Comparable<Car>{
        private String model;
        private String plate;
        public Car (String m, String plate) {
                this.model = m;
                this.plate = plate;
        }
        public String toString() {
                return this.plate;
        }
        @Override public int compareTo(Car o) { return this.plate.compareTo(o.plate); }
}
```