ADSOF: Segundo examen parcial - 06/05/2020

Ejercicio 1 (3 puntos)

Te han encargado el diseño de un sistema de información hospitalario. Para ello, necesitas crear una clase `MedicalRecord` que pueda almacenar los problemas de salud (llamados registros) de los pacientes. `MedicalRecord` debe ser altamente reutilizable, con cualquier clase que implemente información sobre pacientes, y con cualquier tipo de registro que permita realizar las siguientes acciones:

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

Donde `info()` devuelve la descripción del registro, y `set()` permite indicar si el problema de salud es actual (true) o pasado (false).

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

1. `MedicalRecord` mantiene sus entradas ordenadas (alfabéticamente por el nombre del paciente en este caso)
2. Los registros asociados a un paciente se mantienen en orden de inserción y no admiten repeticiones, donde un registro se considera repetido si tiene igual `info()`.
3. Si se tratar de añadir una entrada repetida a `MedicalRecord`, éste lanza una excepción (en este caso, no tenemos en cuenta el teléfono del paciente, sino sólo su nombre).

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

```
class HealthIssue /*…completar si es necesario */{
        private String issue;
        private boolean active;
        public HealthIssue(String issue, boolean active) {
                this.issue = issue;
                this.active = active;
        }
        /*…completar si es necesario */
}

class Patient /*…completar si es necesario */{
        private String name;
        private String phone;
        public Patient (String n, String tel) {
                this.name = n;
                this.phone = tel;
        }
        public String toString() {
                return this.name;
        }
        /*…completar si es necesario */
}
```

```
public class HospitalInformationSystem {
  public static void main(String[] args) {
    MedicalRecord<Patient,HealthIssue> s = new MedicalRecord<>();
    Patient JohnDoe = new Patient("John Doe", "555-123-456");

    s.addRecord(JohnDoe).
      addAll(Arrays.asList(new HealthIssue("Pollen allergy", true),    // nombre del problema, y está activo (true)
                           new HealthIssue("Broken left leg", false),  // pierna rota, problema ya curado (false)
                           new HealthIssue("Broken left leg", true) )); // Registro duplicado y no añadido

    System.out.println(s);               // imprime el MedicalRecord
    if (s.fix(JohnDoe, "Pollen allergy")) // fix devuelve true si el HealthIssue del Patient existe (y false en otro caso)
        System.out.println(JohnDoe+" cured his allergy!");
    s.addRecord(new Patient("Hellen Amber", "555-543-234"));
    System.out.println(s);               // imprime el MedicalRecord

    System.out.println(s.getRecord(JohnDoe));            // Imprime el registro de salud de John Doe's
    s.addRecord(new Patient("John Doe", "555-999-666")); // Lanza una exception, ya que John Doe ya está en s
  }
}
```

Salida esperada:

```
{John Doe=[Pollen allergy:  ongoing, Broken left leg:  fixed]}
John Doe cured his allergy!
{Hellen Amber=[], John Doe=[Pollen allergy:  fixed, Broken left leg:  fixed]}
[Pollen allergy:  fixed, Broken left leg:  fixed]
Exception in thread "main" medicalrecords.RepeatedRecord: Repeated record: John Doe
        at medicalrecords.MedicalRecord.addRecord(HospitalInformationSystem.java:20)
        at medicalrecords.HospitalInformationSystem.main(HospitalInformationSystem.java:114)
```

Solución:

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

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

class MedicalRecord<K extends Comparable<K>, V extends IRecordInfo> {
        private TreeMap<K, Set<V>> pacients = new TreeMap<>();

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

class HealthIssue implements IRecordInfo{
        private String issue;
        private boolean active;
        public HealthIssue(String issue, boolean fixed) {
                this.issue = issue;
                this.active = fixed;
        }

        public boolean equals(Object o) {
                if (o==this) return true;
                if (!(o instanceof HealthIssue)) return false;
                HealthIssue hi = (HealthIssue) 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 Patient implements Comparable<Patient>{
        private String name;
        private String phone;
        public Patient (String n, String tel) {
                this.name = n;
                this.phone = tel;
        }
        public String toString() {
                return this.name;
        }
        @Override
        public int compareTo(Patient o) {
                return this.name.compareTo(o.name);
        }
}
```