ADSOF: Ordinaria - 28/05/2020 – CONTINUA

Ejercicio 3 (2.5 puntos)

Solución:

(1): nada, (2): try, (3): catch(ItemNotFoundException exc) ➔**0,25p**

Total:0,5p

```
interface IPriced<T extends Comparable<T>> extends Comparable<IPriced<T>>{        ➔ (requisitos sobre T e IPriced) 0,25p
        T getValue();
        default String getDesc() { return toString(); }
        @Override default int compareTo(IPriced<T> o) { return getValue().compareTo(o.getValue());}➔ métodos default 0,25p
}
```

```
class Item implements IPriced<Double>{
        private String name;
        private Double initPrice;
        public Item (String i, Double price) {
                this.name = i;
                this.initPrice = price;
        }
        @Override public String toString() { return this.name; }
        @Override public Double getValue() { return initPrice; }
}
```

Total:0,25p

(Errores comunes:

-- Item genérica cuando no lo es

-- Item no genérica, pero usa un parámetro generico

-- Añadir a Item las ofertas o las condiciones (es mejor añadir esto en SalesCrawler, ya que de otro modo habría que reimplementarlo en cada clase que implemente IPriced)

```
class ItemNotFoundException extends Exception {
        public ItemNotFoundException(String item) {
                super("Item "+item+" not found");
        }
}
```

Total:0,25p

(Errores comunes:

-- Almacenar aquí un Item, que haría esta excepción dependiente de Item. Esto es un error, ya que SalesCrawler es independiente de Item, que solo es un ejemplo de clase que implementa IPriced.)

Total: 1,25p

```
public class SalesCrawler<T extends Comparable<T>, I extends IPriced<T>> {        ➔ Requisitos sobre T e I: 0,2
        private Map<I, SortedSet<T>> offers = new TreeMap<>();                    ➔ 0,2
        private Map<I, Predicate<T>> acceptConditions = new HashMap<>();          ➔ 0,2

        public void addItem(I item, Predicate<T> pred) {                          ➔ 0,2
                this.offers.put(item, new TreeSet<>() );
                this.acceptConditions.put(item, pred);
        }

        public boolean offer(String item, T p) throws ItemNotFoundException{      ➔ 0,45
                I bidItem = this.itemDesc(item);
                SortedSet<T> itemBids = this.offersFor(item);
                if (itemBids==null) throw new ItemNotFoundException(item);
                if (!lowestOffer(itemBids, p, bidItem)) return false;
                itemBids.add(p);
                if (this.acceptConditions.get(bidItem).test(p))
                        return true;
                return false;
        }

        private boolean lowestOffer(SortedSet<T> itemBids, T bid, I it) {
                if (itemBids.size()==0) return it.getValue().compareTo(bid)>0;
                if (itemBids.headSet(bid).size()>0) return false;//no the highest bid so far
                return true;
        }

        @Override public String toString() {                                      ➔-0,1 si no está
                return this.offers.toString();
        }

        private I itemDesc(String desc) {
                for (I itm : this.offers.keySet())
                        if (itm.getDesc().equals(desc)) return itm;
                return null;
        }

        private SortedSet<T> offersFor(String itemDesc) {
                I item = this.itemDesc(itemDesc);
                if (item == null) return null;
                return this.offers.get(item);
        }
}
```

(continúa detrás)

Algunos comentarios sobre la solución:

- **Huecos:**
  - En (1) no hay que poner nada, ya que la excepción se captura
- **Interfaz IPriced:**
  - T e IPriced deben ser comparables
  - Añadimos métods default (compareTo, getDesc) para facilitar su implementación
- **Clase Item:**
  - Item no es genérica, como muestra el main
  - Item debe implementar IPriced<Double>
  - Si IPriced no fuese comparable, Item debería implementar Comparable e implementar los métodos. Por tanto, es mejor que IPriced nos proporcione ya ese soporte.
  - Declarar aquí atributos y métodos para las pujas o los predicados no es óptimo, ya que habría que reimplementarlos en clases que implementen IPriced. SalesCrawler no tiene dependencias con Item, sino con IPriced.
- **Clase excepción ItemNotFoundException**:
  - Debe extender de Exception
  - No puede extender de RuntimeException, porque no habría manera de rellenar razonablemente los huecos del main
  - Tiene que tener ese nombre por lo que se imprime en consola.
  - salesSimple es el nombre del paquete, no es el nombre de la clase
- **Clase SalesCrawler:**
  - Tiene dos parámetros genéricos: uno debe ser comparable y otro extender de IPriced.
  - Es un error muy grave de concepto poner en los parámetros genéricos cosas como Item, Double o similar, ya que son nombres de tipos concretos
  - Declarar que una clase lanza una excepción es un error grave de concepto. Las excepciones las lanzan los métodos.
  - Las pujas han de asociarse a los elementos, y esto deben ordenarse en orden de salida. Por esto, lo óptimo es usar un mapa, donde el orden natural sea el precio de salida. Las pujas debe ordenarse por orden ascendente, por lo que la estructura más apropiada sería un conjunto ordenado. Utilizar como clave del mapa un String no es válido, ya que el criterio de orden no se satisface.
  - Han de asociarse predicados a los elementos, para ello hace falta un mapa (aunque TreeSet es innecesario).
  - Errores comunes en método offer: no declarer que lanza la excepción, no lanzarla, no comprobar que es la oferta más baja, no guardar la oferta
- **Otros errores comunes**
  - Usar clases o interfaces genéricas sin parámetros
  - No poner control de acceso private a los atributos (error básico de concepto)
  - Poner un constructor con 0 parámetros sin código (por ejemplo en SalesCrawler) es innecesario.
  - Tratar de copiar de tus compañeros es siempre un error