ADSOF: Segundo examen parcial - 06/05/2020

Ejercicio 2 (3.5 puntos)

Una empresa ofrece *servicios* de asistencia técnica *a través de diversos medios* y necesita una aplicación para gestionar su facturación.
El *precio final* de cada servicio resulta de aplicar un porcentaje de *incremento por prioridad* a la suma de su *coste base* más su *coste extra*. Los porcentajes de incremento por prioridades de servicio son 0%, 12.5%, 12.5% y 25% para las prioridades baja, normal, media y alta, respectivamente. El coste extra será 0 por defecto, a menos que se haya especificado otra cantidad para ese servicio. El coste base se debe calcular de forma distinta según el tipo de servicio. Por ahora consideraremos solo los servicios ofrecidos por *visita* a domicilio o por *llamada* telefónica. El coste base de las visitas se indica individualmente junto a la descripción del servicio y la prioridad asignadas para cada visita. El coste base de las llamadas depende de la duración indicada junto a la descripción y a la prioridad (asumiendo prioridad baja si no se especifica ninguna): serán de 2.00€ por minuto, con un mínimo de 30.00€ y un máximo de 120.00€ por llamada.

Además, queremos poder acceder fácilmente a todos los servicios de prioridad igual o superior a una prioridad indicada, o todos los servicios si no se especifica ninguna. También deseamos contabilizar el número total de llamadas, haciendo que la aplicación añada automáticamente dicha numeración a la descripción de cada llamada.

Nota: por simplicidad, puedes ignorar el control de errores derivados de cantidades negativas y strings vacíos o nulos.

Se pide diseñar y codificar en Java, siguiendo buenos principios de orientación a objetos, las clases necesarias para resolver los requisitos anteriores haciendo que el programa dado más abajo **produzca la siguiente salida:**

```
Num. de llamadas: 3
Media o superior: 4
          Call X#1, BAJA, Precio final:  30,00
        Call Z#2, NORMAL, Precio final: 112,50
          Call X#3, ALTA, Precio final: 150,00
       Visit 28701, BAJA, Precio final: 100,00
         Visit ABC, ALTA, Precio final: 250,00
       VisitExtra, MEDIA, Precio final: 225,00
```

Tester en forma de imagen para evitar cortar y pegar de texto del enunciado a Eclipse. Más abajo va en forma de texto.

```
public class Ej2 {
   public static void main(String[] args) {
      Service[] servicios = {
         new Call("Call X", 12), // min 30€, default BAJA -> incremento 0%
         new Call("Call Z", 50, Priority.NORMAL), // 100€, MEDIA -> incr 12,5% = 112,50
         new Call("Call X", 80, Priority.ALTA), // max 120€  -> incr 25% = 150
         new Visit("Visit 28701", 100.0, Priority.BAJA), // 100€, incr 0%, sin extra
         new Visit("Visit ABC", 200.0, Priority.ALTA), // 200 -> 25% = 250
         new Visit("VisitExtra", 150.0, Priority.MEDIA).extra(50.0) // extra coste
      };
      System.out.println("Num. de llamadas: "+ Call.count() ); // 4 llamadas
      System.out.println("Media o superior: "
                         + Service.select(Priority.NORMAL).size() ); // NORMAL o superior
      for (Service s : Service.select()) {
         System.out.printf("%24s, Precio final: %6.2f\n",s ,s.finalPrice());
      }
   }
}
```

Solución: Indica asignación orientativa de n décimas de puntos en forma de [n] sobre total de 35. Otros fallos penalizados aparte.

```
public enum Priority {
	//  LOW(0.0), MED(0.125), HIGH(0.125), VERY_HIGH(0.25);
	BAJA(0.0), NORMAL(0.125), MEDIA(0.125), ALTA(0.25);     // [2]
	private double value;
	private Priority(double value) { this.value = value; }  // [2]
	public double getValue() { return this.value; }
}

import java.util.*;
public abstract class Service { //  abstract [1]
	private static List<Service> services = new ArrayList<>();
	public static List<Service> select(Priority en) {        // [3]
		List<Service> result = new ArrayList<>();
		for (Service a : Service.services)
			if (a.priority.compareTo(en)>=0) result.add(a);
		return result;
	}
	public static List<Service> select() {                   // [2]
		return select( Priority.values()[0]);
		// return Collections.unmodifiableList(Service.memory);
	}
	private Priority priority;
	private String description;
	public Service(String d, Priority e) {   //  [3]
		description = d; priority = e; services.add(this); }

	public double finalPrice()  { return (this.basicCost() // [4]
					+ this.extra() )
					* (1.0 + this.priority.getValue()); }

	public abstract double basicCost();    //  [1]
	public  double extra() { return 0.0; } //  [1]
	public String toString() {return description + ", "   + this.priority;} // [1]
}

public class Call extends  Service {   // extends [1]
	public static final double MAX_BASIC_COST = 120.0;
	public static final double MIN_BASIC_COST = 30.0;
	public static final double BASIC_COST_PER_MINUTE = 2.0;
	private static int lastId = 0;                  //  [1]
	public static int count() { return lastId; }   //  [1]

	private int minutes;
	public Call(String descripcion, int c1, Priority enumera) {
		super(descripcion + "#"+  ++lastId, enumera); // [2]
		this.minutes = c1;
	}
	public Call(String descripcion, int c1) {
		this(descripcion, c1, Priority.BAJA); // LOW);  //  [2]
	}
	@Override public double  basicCost() { // [1]
		return Math.min(MAX_BASIC_COST,
				Math.max(MIN_BASIC_COST,
					BASIC_COST_PER_MINUTE * this.minutes));
	}
}

public class Visit extends Service { // extends [1]
	private double basicCost;
	private double extra = 0.0;
	public Visit(String descripcion, double cost, Priority enumera) {
		super(descripcion, enumera);  // [2]
		basicCost = cost;
	}
	@Override public double basicCost() { return basicCost; }    // [1]
	@Override public double extra()  { return this.extra;}       // [1]
	public Visit extra(double extra)  { this.extra = extra; return this;} // [2]
}
```