Prueba 2 de Evaluación Continua

Análisis y Diseño de Software (2013/2014)

Contesta AL DORSO o en las áreas sombreadas, usa hojas adicionales si fuese necesario

Apellidos:
Nombre:

Ejercicio 1: Programación Orientada a Objetos (3,5 puntos)

Se va a desarrollar una aplicación orientada a objetos para una empresa que organiza actividades de ocio. Cada actividad se caracteriza por una descripción y un precio base. El precio total de cada actividad se calcula sumando su precio base y un coste extra que se calcula de manera diferente para cada tipo de actividad. Hay dos tipos de actividades: práctica deportiva y visita a museo. Cada práctica deportiva lleva asociado un nivel de riesgo, que se utiliza para calcular su coste extra (multiplicado riesgo por precio base). Todas las visitas a museos tienen un precio base constante de 5.00€, y su coste extra se calcula según sea una visita guiada (4.00€), bilingüe (8.00€), escolar (descuento de 1.00€), o individual (sin coste extra). Además, se debe llevar automáticamente una contabilidad de la suma de los precios totales de todas las actividades deportivas que se hayan creado.

Se pide escribir el código Java necesario para modelar los objetos descritos arriba de forma que al ejecutar la siguiente clase de prueba se produzca la salida mostrada abajo. No escribas métodos innecesarios para esta ejecución, pero presta especial atención al diseño de las clases necesarias y las relaciones entre ellas y entres sus métodos.

```
public class Apartado1 {
        public static void main(String[] args) {
           Actividad[] actividades = { new VisitaMuseo("Reina Sofia", TipoVisitaMuseo.BILINGUE),
                                       new VisitaMuseo("Arqueológico", TipoVisitaMuseo.ESCOLAR),
                                       new PracticaDeportiva("Cañada Real Bicicleta", 20.0, 0.1),
                                       new PracticaDeportiva("Descenso Rapido Sella", 50.0, 0.25)
                                     };
          System.out.println("Precio base fijo para museos: " + VisitaMuseo.COSTE_FIJO);
          for (Actividad a : actividades)
             System.out.println(a + ": precio " + a.precio() + " incluye " + a.extras() +" de complementos");
          System.out.println("Suma de precios de actividades deportivas " + PracticaDeportiva.total());
        }
}
```

Salida esperada:

```
Precio base fijo para museos: 5.0
Reina Sofia: precio 13.0 incluye 8.0 de complementos
Arqueológico: precio 4.0 incluye -1.0 de complementos
Cañada Real Bicicleta: precio 22.0 incluye 2.0 de complementos
Descenso Rapido Sella: precio 62.5 incluye 12.5 de complementos
Suma de precios de actividades deportivas 84.5
```

Prueba 2 de Evaluación Continua

Análisis y Diseño de Software (2013/2014)

Contesta AL DORSO o en las áreas sombreadas, usa hojas adicionales si fuese necesario

Apellidos:
Nombre:

Ejercicio 2: Colecciones (3,5 puntos)

Se quiere realizar una aplicación para el departamento de repartos de Correos, que maneje direcciones. Para ello, se ha creado una clase Direccion. Completa la clase y el programa de prueba, para que la salida sea la indicada más abajo.

```
import java.util.*;
class Direccion /* completar si es necesario*/ ..........{
  private String calle;
  private int    numero;
  private int    codigoPostal;

  public Direccion(String calle, int num, int cp) {
        this.calle = calle; this.numero = num; this.codigoPostal = cp;
  }
  @Override public String toString() {
        return this.calle+" "+this.numero+" ("+this.codigoPostal+")";
  }
  // Completar con métodos, si es necesario
```

```
  public String getDireccion() {return this.calle;}
  public int getCodigoPostal() {return this.codigoPostal;}
  public int getNumero()       {return this.numero;}
}

public class RepartoCorreos {
  public static void main(String ...args) {
        Direccion[] direcciones = { new Direccion("San Vicente Ferrer", 37, 28047),
                                    new Direccion("Barquillo", 3, 28014),
                                    new Direccion("Alcala", 45, 28014),
                                    new Direccion("Alcala", 45, 28014),
                                    new Direccion("San Vicente Ferrer", 35, 28047)
                                  };
        ..........calles = new .......... (Arrays.asList(direcciones)); // Completar
        System.out.println("Direcciones distintas:\n "+calles); // Sólo sale una Direccion C/ Alcalá
        // Completar el Comparator de más abajo
        Comparator<Direccion> comparaCallesYNumero = new ..........{ // Orden por Direccion y número
                        @Override public ..... {
                        // Completar
                        }
        };
        ..........ordenados = new ..........(); // Completar
        for (Direccion d : direcciones) {
          if (!ordenados.containsKey(d.getCodigoPostal()))
             ordenados.put(d.getCodigoPostal(), new TreeSet<Direccion>(comparaCallesYNumero));
          ordenados.get(d.getCodigoPostal()).add(d);
        }
        System.out.println("Direcciones por CP y Calle:\n "+ordenados);
    }
}
```

```
Salida:
Direcciones distintas:
  [San Vicente Ferrer 37 (28047), Barquillo 3 (28014), Alcala 45 (28014), San Vicente Ferrer 35 (28047)]
Direcciones por CP y Calle:
 {28014=[Alcala 45 (28014), Barquillo 3 (28014)], 28047=[San Vicente Ferrer 35 (28047), San Vicente
Ferrer 37 (28047)]}
```

Prueba 2 de Evaluación Continua

Análisis y Diseño de Software (2013/2014)

Contesta AL DORSO o en las áreas sombreadas, usa hojas adicionales si fuese necesario

Apellidos:
Nombre:

Ejercicio 3: Distribuidor genérico (3 puntos)

Dadas las siguientes interfaces que modelan un sistema de toma de decisiones y ejecución de tareas:

```
public interface Decision<I, O> {
	O decide(I input); //dado un input de tipo I, devuelve el resultado de la decisión, que será de tipo O
}
public interface Ejecutor <I, O>{
	O ejecuta(I input);//Ejecuta la tarea usando input como entrada, y devuelve un resultado
}
```

Se quiere programar una clase **Distribuidor** cuyas instancias se crean con un árbitro que toma decisiones a partir de las entradas al distribuidor y un Map que asocia un Ejecutor con cada resultado de la decisión tomada por el árbitro. El objetivo es que el método `ejecuta` de Distribuidor delegue el trabajo al Ejecutor que corresponda dependiendo del resultado de la decisión tomada por del árbitro.
Nota: Un Distribuidor es también un Ejecutor.

Para ello rellena a continuación la parte del código sombreada:

```
public class Distribuidor <I, D, O> implements                                  {
	final Decision<                                  > arbitro;
	final Map<                                  , Ejecutor<                                  >> ejecutores;
	public Distribuidor(Decision<                                  > arbitro,
		Map<                                  , Ejecutor<                                  >>ejecutores){
		this.arbitro=arbitro;
		this.ejecutores=ejecutores;
	}
	public O ejecuta(I input){
	  Ejecutor<                                  > ejecutor=
	}
}
```

A la hora de configurar los parámetros genéricos aprovecha el uso de los comodines para permitir un uso lo más flexible posible de la implementación de Distribuidor. Para ello puede ser útil fijarse en el siguiente ejemplo de uso:

```
Decision<Object, String> getKind= new Decision<Object, String>(){
	@Override
	public String decide(Object input) {
		return input.getClass().getSimpleName(); //Devuelve el nombre de la clase
	}
};

Ejecutor<Tarea, Integer> ejecutorEnteros= new EjecutorEnteros();
Ejecutor<Tarea, Double> ejecutorReal= new EjecutorReal();
Map<Object, Ejecutor<? super Tarea, ? extends Number>> ejecutores= new HashMap<>();
ejecutores.put("TareaEnteros", ejecutorEnteros);
ejecutores.put("TareaReal", ejecutorReal);

Ejecutor<Tarea, Number> distribuidor=
		new Distribuidor<Tarea, Object, Number>(getKind, ejecutores);
TareaEnteros te= new TareaEnteros(); //TareaEnteros es subclase de Tarea
//... introducción de datos en la te
TareaReal tr = new TareaReal(); //TareaReal es sublcase de Tarea
//... introducción de datos en la tr
Number n1= distribuidor.ejecuta(te);
Number n2= distribuidor.ejecuta(tr);
System.out.println("Resultado de ejecutar te: "+n1);  //muestra un resultado entero
System.out.println("Resultado de ejecutar tr: "+n2);  //muetra un resultado en coma flotante
```