Resolución Primer Parcial de Programación Orientada a Objetos (72.33)

15/04/2020

Ejercicio 1

	m1	m2	m3
A	3	3	3
B	2	2	2
C	1	java.lang.StackOverflowError	java.lang.StackOverflowError

Ejercicios similares

- ***Ejercicio 10 TP N°3: Introducción a POO en Java***
- ***Ejercicio 4 Taller 2***
- ***Ejercicio 4 Repaso Primer Parcial***

Ejercicio 2

```java
public class MathFunctionInterval<E extends Comparable<? super E>> implements
Iterable<E> {

   private E start, end;
   private MathFunction<E> step;

   public MathFunctionInterval(E start, E end, MathFunction<E> step) {
       if(start.compareTo(end) >= 0) { // Si ponían > 0 también estaba bien
           throw new IllegalArgumentException("Start no es menor que end");
       }
       this.start = start;
       this.end = end;
       this.step = step;
   }

   @Override
   public Iterator<E> iterator() {
       return new MathFunctionIntervalIterator();
   }

   private class MathFunctionIntervalIterator implements Iterator<E> {

       private E current = start;

       @Override
       public boolean hasNext() {
           return current.compareTo(end) < 0; // Si ponían <= también estaba bien
       }
```

```
        @Override
        public E next() {
            if(!hasNext()) {
                throw new NoSuchElementException();
            }
            E toReturn = current;
            current = step.evaluate(current);
            return toReturn;
        }

    }

}
```

Errores más comunes

- Reciben `MathFunction` sin usar generics. en vez de `MathFunction<E>` (-0,5)
- `hasNext`: no se puede comparar con <= , deben usar el `compareTo`. ¿Para qué exigió que `T` implemente `Comparable`? (-1)
- Si además lo hacen en `next`, otros -0,5
- la clase debe implementar `Iterable < T >` no `Iterator < T >`. Internamente crear un iterador (con una clase específica o clase anónima) (-1)
- hacer current.func, siendo current de tipo T y func de tipo `MathFunction` (-1)
- debe ser `implements Iterable < T >` no `implements Iterable < T extends ....>` (la restricción al tipo T la hicieron antes, después simplemente la deben referenciar). No se bajaron puntos por esto
- `T` debe implementar `Comparable` (-2 si hacen todo con <, <=, -1 si usan `compareTo`)
- No valida que `start` sea "menor" que `end` (-1)
- La propiedad `step` tiene que ser un atributo del iterador, no de la clase pedida, ¿qué sucede si creo dos iteradores sobre la misma instancia? (-0,5)

Ejercicio 3

Versión 1

```java
public abstract class Element {

   public abstract String getContents();

   @Override
   public String toString() {
       return getContents();
   }

   public Element above(Element element) {
       return new BinaryElement(this, element);
   }

   public Element below(Element element) {
       return new BinaryElement(element, this);
   }

   public Element repeat(int times) {
       return new RepeatableElement(this, times);
   }

}
```

```java
public class TextElement extends Element {

   private String text;

   public TextElement(String text) {
       this.text = text;
   }

   @Override
   public String getContents() {
       return text;
   }

   public void setText(String text) {
       this.text = text;
   }

}
```

```java
public class BinaryElement extends Element {

   private final Element above, below;

   public BinaryElement(Element above, Element below) {
       this.above = above;
       this.below = below;
   }

   @Override
   public String getContents() {
       return above + "\n" + below;
```

```
    }

}
```

```
public class RepeatableElement extends Element {

    private final Element element;
    private final int times;

    public RepeatableElement(Element element, int times) {
        this.element = element;
        this.times = times;
    }

    @Override
    public String getContents() {
        StringBuilder s = new StringBuilder();
        for(int i = 0; i < times; i++) {
            s.append(element.getContents());
            if(i != times - 1) {
                s.append("\n");
            }
        }
        return s.toString();
    }

}
```

Versión 2

Implementación entregada por un alumno/a

Similar a la anterior pero se puede prescindir de crear la clase `RepeatableElement` con la siguiente implementación de `repeat`.

```
public abstract class Element {

    public abstract String getContents();

    @Override
    public String toString() {
        return getContents();
    }

    public Element above(Element element){
        return new BinaryElement(this, element);
    }

    public Element below(Element element){
        return new BinaryElement(element, this);
    }

    public Element repeat(int n) {
        Element aux = this;
        for(int i = 0; i < n-1; i++) {
            aux = new BinaryElement(aux,this);
        }
```

```
        return aux;
    }

}
```

Errores más comunes

- Los métodos `above`, `below` y `repeat` son `void` o retornan `String` en lugar de `Element` incumpliendo el programa de prueba (-3)
- Los métodos `above`, `below` y `repeat` retornan nuevas instancias de `Element` pero éstas contienen una copia de los `String` de los elementos originales en lugar de quedarse con la referencia. En el programa de prueba se cambian los contenidos de elementos y los cambios de texto de un elemento se tienen que ver reflejados en los elementos compuestos que dependen de estos que sufrieron cambios (-2)
- Los métodos `above`, `below` y `repeat` están implementados en `TextElement` y no en `Element`. En el programa de prueba se exige que `below` y `repeat` estén en `Element` (-0,5)
- Se repite código implementando clases `Above` y `Below` (-0,5)
- Errores menores (-0,25)
    - Uso del modificador `abstract` en una clase concreta
    - Uso del modificador `implements` en lugar de `extends`
    - Uso del modificador `default` en métodos concretos de una clase abstracta
    - Castear una instancia de `StringBuilder` a `String`
    - Instanciar una clase abstracta
    - Invocar al constructor sin parámetros cuando no existe (porque se declaró un constructor con parámetros)

Ejercicio similar

- ***Ejercicio 6 TP N°4: Interfaces, Excepciones y Enumerativos***