Resolución Recuperatorio Primer Parcial de Programación Orientada a Objetos (72.33)

26/06/2019

Ejercicio 1

```
public class Movie implements Comparable<Movie> {

   private String title;
   private int year;
   private double rating;

   public Movie(String title, int year, double rating) {
       if ( rating < 0.0 || rating > 10.0 ) {
          throw new ...
       }
       this.title = title;
       this.year = year;
       this.rating = rating;
   }

   @Override
   public int compareTo(Movie o) {
       int cmp = Double.compare(o.rating, rating);
       if(cmp == 0) {
           cmp = title.compareTo(o.title);
       }
       return cmp;
   }

   public double getRating() {
       return rating;
   }

   public int getYear() {
       return year;
   }

   public String getTitle() {
       return title;
   }

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

}
```

```
Arrays.sort(bttfTrilogy, new Comparator<Movie>() {
   @Override
   public int compare(Movie o1, Movie o2) {
       int cmp = Integer.compare(o2.getYear(), o1.getYear());
       if (cmp == 0) {
           cmp = o1.getTitle().compareTo(o2.getTitle());
       }
```

```
        return cmp;
    }
});
System.out.println(Arrays.toString(bttfTrilogy));
```

Ejercicio Similar del Taller:

- ***Taller 4: Ejercicios 3 y 4.*** *Crear una clase que implementa Comparable y crear un Comparator como clase anónima en el main.*

Ejercicio 2

```
public interface TimesCollection<E> extends Iterable<E> {

    void add(E element);

    void setTimes(int n);

}
```

```
public class ArrayTimesCollection<E> implements TimesCollection<E> {
```

```
...
```

```
@Override
public void add(E element) {
    if(dim == array.length) {
        resize();
    }
    array[dim++] = element;
}

private void resize() {
    array = Arrays.copyOf(array, array.length + INITIAL_DIM);
}

@Override
public void setTimes(int n) {
    if ( n <= 0)
        throw new IllegalArgumentException();
    this.times = n;
}

@Override
public Iterator<E> iterator() {
    return new Iterator<E>() {
        private int topTimes = times;
        private int timesCurrent = 0;
        private int index = 0;
        @Override
        public boolean hasNext() {
            return index < dim;
        }

        @Override
        public E next() {
            if ( ! hasNext()) {
```

```
                throw new NoSuchElementException();
            }
            E ans = array[index];
            timesCurrent++;
            if ( timesCurrent == topTimes) {
                index++;
                timesCurrent = 0;
            }
            return ans;
        }
    };
}
```

Ejercicio Similar del Taller:

- ***Taller 5: Ejercicio 2.*** *Agregar elementos a un arreglo, agrandarlo cuando corresponda, interfaz que extienda a Iterable y clase privada Iterator.*

Ejercicio 3

```
public abstract class AbstractElement {

    public abstract int getWidth();

    public abstract int getHeight();

    public String toString() {
        StringBuilder s = new StringBuilder();
        for(int i = 0; i < getWidth(); i++) {
            for(int j = 0; j < getHeight(); j++) {
                s.append(getCharacter());
            }
            s.append('\n');
        }
        return s.toString();
    }

    private String getCharacter() {
        return "#";
    }

}
```

```
public class RectangularElement extends AbstractElement {

    private int width, height;

    public RectangularElement(int width, int height) {
        this.height = height;
        this.width = width;
    }

    @Override
    public int getWidth() {
        return width;
    }

    @Override
```

```
    public int getHeight() {
        return height;
    }

    public void setWidth(int width) {
        this.width = width;
    }

    public void setHeight(int height) {
        this.height = height;
    }

}
```

```
/* En este caso privilegiamos reusar comportamiento que ya tiene la clase
** RectangularElement y no repetir codigo. Si bien tal vez un cuadrado no tendria que
** tener los metodos setWidth y setHeight.
** Otra posibilidad es que extienda AbstractElement, tenga solamente setDim
** y sobreescriba getWidth y getHeight para que ambos retornen una propiedad privada
** dim.
*/
public class SquareElement extends RectangularElement {
    public SquareElement(int dim) {
        super(dim, dim);
    }

    public void setWidth(int width) {
        super.setWidth(width);
        super.setHeight(width);   // ¿Por qué no setHeight(width) ?

    }

    public void setHeight(int height) {
        super.setHeight(height);
        super.setWidth(height);   // ¿Por qué no setWidth(height) ?
    }

}
```

```
public class ResizableElement extends AbstractElement {

    private int multiplier;
    private AbstractElement element;

    public ResizableElement(AbstractElement element, int multiplier) {
        this.element = element;
        this.multiplier = multiplier;
    }

    @Override
    public int getWidth() {
        return element.getWidth() * multiplier;
    }

    @Override
    public int getHeight() {
        return element.getHeight() * multiplier;
    }
```

```
    public void resize(int multiplier) {
        this.multiplier = multiplier;
    }

}
```