



Desenvolvimento de aplicações multiplataforma

Contornos 2º avaliação

Fecha: 17 Marzo 2023

Nombre y apellidos:

Dni:

VALORES DE LAS PREGUNTAS

Parte 1: Documentación 1 pto
Parte 2: Optimización 3 ptos
Parte 3: Testing 2 ptos
Parte 4: Refactorización 3 ptos
Parte 5: Complejidad algorítmica 1 pto

Documentación

Del siguiente ejercicio, realiza la **documentación** solamente la clase **Pizza**

- Hay que indicar que el método **obtenerIngredientesFormatoTexto** está deprecated

```
import java.util.ArrayList;

class Ingrediente {
    private String nombre;
    private float precio;

    public Ingrediente(String nombre, float precio) {
        super();
        this.nombre = nombre;
        this.precio = precio;
    }

    public String getNombre() {
        return nombre;
    }
}
```

```
public void setNombre(String nombre) {
    this.nombre = nombre;
}

public float getPrecio() {
    return precio;
}

public void setPrecio(float precio) {
    this.precio = precio;
}

}

public class Pizza {

    final static float PRECIO_BASE = 5;

    private ArrayList<Ingrediente> listaIngredientes = new ArrayList<Ingrediente>();

    public int getCantidadIngredientes() {
        return listaIngredientes.size();
    }

    public float getCostePizza() {
        float coste = PRECIO_BASE;
        for (Ingrediente i : listaIngredientes) {
            coste += i.getPrecio();
        }
        return coste;
    }

    public void addIngrediente(Ingrediente bola) {
        listaIngredientes.add(bola);
    }
}
```

```
public void quitarIngrediente(Ingrediente bola) {
    listaIngredientes.remove(bola);
}

public void quitarTodosLosIngredientes() {
    listaIngredientes.clear();
}

public String obtenerIngredientesFormatoTexto() {
    String resultado = "";
    for (Ingrediente i : listaIngredientes) {
        resultado += i.getNombre();
        resultado += "\n";
    }
    return resultado;
}

@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    for (Ingrediente i : listaIngredientes) {
        sb.append(i.getNombre());
        sb.append("\n");
    }
    return sb.toString();
}
}
```

Optimiza

Se parte del siguiente programa que almacena un conjunto de frutas y realiza una serie de cálculos sobre ellas.

Las frutas no se van a modificar (ni se van a añadir o quitar frutas)

Se pide que **optimices** (en la medida de lo posible) todos los métodos del ejercicio:

- Puede que no todos los métodos requieran ser optimizados.
- Pon, encima de cada método, una breve explicación de lo que has hecho para optimizar.



```
public class Optimiza {  
    String frutas[] = { "platano", "fresa", "pomelo", "limon", "naranja", "pera"};  
  
    /* Este método busca si una fruta está en la lista de frutas */  
    boolean busca(String frutaABuscar) {  
        boolean siEsta = false;  
        boolean noEsta = true;  
        for (String fruta : frutas) {  
            if (fruta.equals(frutaABuscar)) {  
                siEsta = true;  
                noEsta = false;  
            }  
        }  
        noEsta = !siEsta;  
        return siEsta;  
    }  
  
    // Función que determine si hay alguna fruta repetida en la lista  
    public boolean hayFrutaRepetida() {  
        for (int i = 0; i < frutas.length; i++) {  
            for (int j = i + 1; j < frutas.length; j++) {  
                if (frutas[i].equals(frutas[j])) {  
                    return true;  
                }  
            }  
        }  
        return false;  
    }  
  
    /* Este cuenta cuantas frutas hay de como minimo X letras */  
    int cuentaFrutasConMinimoLetras(int minimoLetras) {  
        int contador = 0;  
        int num = contador;  
        for (String fruta : frutas) {  
            if (fruta.length() >= minimoLetras) {  
                num = contador;  
                contador++;  
            }  
        }  
        num = num / frutas.length;  
    }  
}
```

```
        return contador;
    }

    /* Calcula la media de caracteres de todas las frutas*/
    // En el ejemplo la media de todos los caracteres es 7+5+6+5+7+4 / 6 = 30.66^
    float mediaCaracteres(){
        float suma = 0;
        for(String frase: frutas){
            suma+=frase.length();
        }
        return suma / frutas.length;
    }

    /* Por cada fruta, devuelve el numero de caracteres de cada fruta dividido por
    la media de caracteres totales */
    // Contamos numero de caracteres de cada fruta = [7+5+6+5+7+4]
    // Numero total de frutas = 6
    // Media total de caracteres = 7+5+6+5+7+4 / 6 = 30.66^
    // Dividimos el numero de caracteres de cada fruta por la media total de
    caracteres
    // [0.22, 1.16, 0.19, 0.16, 0.22, 0.13]
    float[] numerosCaracteresDivididoEntreMediaTotal(){
        float[] resultado = new float[frutas.length];
        for(int i=0;i<frutas.length;i++){
            resultado[i] = frutas[i].length() / mediaCaracteres();
        }
        return resultado;
    }
}
```



Testing

Del ejercicio anterior, realiza los test unitarios con JUnit de las funciones

- `boolean busca(String fraseBuscar)`
- `int cuentaFrutasConMinimoLetras(int minimoLetras)`
- `float mediaCaracteres()`

Refactorización

Dada la siguiente clase que modela el comportamiento de un Hechizo. Se pide refactorizar la misma para obtener un código más limpio.

```
public class Hechizo {

    enum TiposHechizo {
        Transformacion, Encantamiento, Embrujo, Malefico
    };

    String nombre = "default"; // nombre del hechizo
    String descripcion; // Breve descripción del hechizo
    private boolean preparado = true; // Determina si está preparado para lanzar el
    hechizo

    TiposHechizo tipo = TiposHechizo.Transformacion;

    String nombreHechizero; // Nombre del hechizero
    int nivelHechizero; // Nivel alcanzado por el hechicero
    float destrezaHechizero; // Destreza del hechizero
    float manaHechizero; // Cantidad de maná del hechizero

    boolean puede_lanzar_hechizo() {
        if (preparado) {
            if (manaHechizero > 0) {
                return true;
            } else {
                return false;
            }
        } else {
            return false;
        }
    }

    // Calcula el daño de una carta
    float calcularDannoDelHechizoQueSeVaAAplicarCuandoAtaca() {
        switch (tipo) {
            case Transformacion:
                return nivelHechizero * 2 + destrezaHechizero;
            case Encantamiento:
                return nivelHechizero * 1 + destrezaHechizero;
            case Embrujo:
                return nivelHechizero / 2 + destrezaHechizero;
            case Malefico:
```



```
        return nivelHechizero * 2;
    default:
        return 0;
    }

}

public static void main(String[] args) {

    Hechizo c = new Hechizo();
    c.nivelHechizero=2;
    c.destrezaHechizero=3;
    c.manaHechizero=10;
    c.tipo = TiposHechizo.Embrujo;
    System.out.println(c.calcularDannoDelHechizoQueSeVaAAplicarCuandoAtaca());

}
}
```

Complejidad algorítmica

En el ejercicio de optimizar hay un método llamado **hayFrutaRepetida()**.
¿Qué complejidad algorítmica tiene? Exprésala con notación BigO

En el ejercicio de refactorización hay un método llamado **puedeLanzarHechizo()**.
¿Qué complejidad algorítmica tiene? Exprésala con notación BigO