Ejercicio Nivel 7- CupiCava

Descripción global

El Equipo Cupi2 necesita construir una aplicación para manejar la información de los vinos almacenados en una cava. La aplicación debe permitir ordenar los vinos según su nombre, año de elaboración y lugar de origen.

Objetivos del ejercicio

El objetivo de este ejercicio es que el estudiante comprenda y adquiera práctica en:

- Conceptos vistos en el curso anterior (Mundo e Interfaz).
- Desarrollará una aplicación siguiendo un proceso incremental.
- Construirá los invariantes de las clases del mundo del ejercicio.
- Utilizará la instrucción **`assert`** de Java para verificar invariantes.
- Desarrollará pruebas unitarias en **`Junit`** para las clases del ejercicio.
- Entenderá y aplicará el concepto de comparación de objetos.
- Entenderá y desarrollará tres algoritmos de ordenamiento (burbuja, inserción y selección).
- Entenderá y desarrollará algoritmos de búsqueda binaria y secuencial sobre una lista ordenada o no ordenada.
- Aprenderá a representar un objeto como un texto con el método **`toString().`**
- Utilizará **`JList`** y **`JScrollPane`** para presentar listas en la interfaz gráfica.

Los siguientes pasos conforman el plan sugerido para desarrollar el ejercicio. La idea es ir desarrollando y probando incrementalmente los métodos de las clases. No se pre ocupe si las clases de la interfaz o de las pruebas (test) tienen errores. Estos desaparecerán cuando termine (correctamente) los cambios en el modelo del mundo.

Este ejercicio debe ser realizado de manera INDIVIDUAL.

Preparación

1. Descargue del sitio web del curso el archivo demo de la aplicación (del enlace llamado **`n7_cupiCava_demo.mp4`**) y ejecútelo para conocer el funcionamiento esperado del programa.
2. Descargue del sitio web del curso el esqueleto del ejercicio (del enlace llamado **`n7_cupiCava_Esqueleto`**). Descomprima este archivo e importe el proyecto llamado **`n7_cupiCava`** en **`Eclipse`**.

3. Lea el enunciado del problema disponible en: **`n7_cupiCava/docs/specs/Descripcion.pdf`**.
4. Estudie el documento de requerimientos funcionales disponible en: **`n7_cupiCava/docs/specs/RequerimientosFuncionales.pdf`**.
5. Estudie los modelos para este ejercicio. Este modelo se encuentra en: **`n7_cupiCava/docs/specs/<nombremodelo>.jpg`**. Identifique las clases, relaciones entre clases, constantes, enumeraciones, atributos y métodos.
6. Desde Eclipse revise la documentación de las clases del mundo. Esto le permitirá entender para qué sirve cada método y cada uno de los atributos. También puede generar la documentación del proyecto (**`archivos .html`**) ejecutando el programa **`doc`** que se encuentra en **`n7_cupiCava/bin/win`** (para Windows) y en **`n7_cupiCava/bin/mac`** (para Mac).

Dentro del código del esqueleto se encuentran indicados los puntos donde usted debe realizar alguna modificación (añadir atributos, completar métodos, construir nuevos métodos, etc.), por medio de comentarios de la siguiente forma:

```
// TODO: Breve explicación de la modificación que debe realizar.
```

A continuación encuentra una guía para completar el ejercicio:

Enumeraciones y métodos estáticos

1. Complete la enumeración **`TipoVino`**
    a. Complete el método constructor
    b. Complete el método estático **`darTipoPorContenidoDeAzucar`**
2. Cree la enumeración **`ColorVino`**
    a. Use como guía el modelo del mundo y siga el ejemplo de la enumeración **`PresentaciónVino`**
3. En la clase **`Vino`**
    a. Declare el atributo que permite almacenar el tipo del vino
    b. Declara el atributo que permite almacenar el color del vino
    c. Complete el método constructor inicializando el atributo que guarda el tipo del vino

Pruebas sobre búsquedas

4. En la clase **`CupiCavaTest`**
    a. Complete el método **`testBuscarVinoMasDulce`**
    b. Complete el método **`testBuscarVinoMasSeco`**
    c. Complete el método **`testBuscarBinarioPorNombre`**

Búsquedas

5. En la clase **`CupiCava`**
    a. Complete el método **`buscarBinarioPorNombre`**

b. Complete el método **buscarVinoMasDulce**
c. Complete el método **buscarVinoMasSeco**
d. Complete el método **buscarVinosDeTIpo**

Invariantes

6. En la clase **Vino**
    a. Defina y documente el invariante de la clase
    b. Implemente el método **verificarInvariante**
    c. Utilice el método **verificarInvariante** donde sea necesario
7. En la clase **CupiCava**
    a. Defina y documente el invariante de la clase
    b. Implemente el método **verificarInvariante**
    c. Utilice el método **verificarInvariante** donde sea necesario

Elemento gráfico JList

8. En la clase **Vino**
    a. Complete el método **toString**
9. En la clase **PanelListaVinos**
    a. Declare el atributo que permite pintar la lista de vinos
    b. Inicialice el atributo que permite pintar la lista de vinos
    c. Cree e inicialice un elemento gráfico que permite manejar las barras de desplazamiento alrededor de la lista de vinos
    d. Complete el método **refrescarLista**
    e. Complete el método **valueChanged**

Pruebas sobre comparadores y ordenamientos

10. En la clase **ComparadorTest**
    a. Complete el método **setup**
    b. Complete el método **compare**
11. En la clase **OrdenadorTest**
    a. Complete el método **testOrdenarDescendente**

Comparadores

12. En la clase **ComparadorVinoNombre**
    a. Complete el método **compare**
13. En la clase **ComparadorAnho**
    a. Complete el método **compare**
14. Cree la clase **ComparadorVinoLugar** la cual se encarga de comparar dos vinos según su lugar de origen.
    a. La clase debe implementar el contrato **Compare<T>**
    b. Use como referencia las otras clases comparadoras

Ordenamientos

15. En la clase **`Ordenador`**
    a. Complete el método **`ordenarInsercion`**
    b. Complete el método **`ordenarSeleccion`**
    c. Complete el método **`ordenarBurbuja`**
    d. Complete el método **`ordenarShaker`**
    e. Complete el método **`ordenarGnome`**

Elemento gráfico JComboBox

16. En la clase **`PanelOpciones`**
    a. Declare el atributo que permite mostrar los algoritmos de ordenamiento disponibles
    b. En el método constructor inicialice el atributo que permite mostrar los algoritmos de ordenamiento disponibles
    c. Complete el método **`actionPerformed`**