Cenco Master 2018 04Programación II - TP1 recup - 1er Cuatrimestre 2018

Fecha de presentación: 7/6/18
Fecha de entrega por mail: 21/6/18

Requerimientos técnicos:
Grupos de 1 o 2 personas
Se debe utilizar al menos una vez iteradores y stringbuilder (Tecnologías java).
Además de pasar el junit suministrado en el TP, la cátedra testeará los ejercicios con otro junit adicional, por lo que se recomienda armar un junit propio para probar, antes de entregar el TP.
Se debe escribir el Irep del ej1 en el informe de no más de una página.

Ejercicio 1 Diseño

a) Se desea modelar el juego del Jenga con k jugadores.

Modelar al TAD Jenga

Escribir el Irep
Justificar los TAD o clases soporte. Es obligatorio que al menos haya otro TAD que no sea una clase de java, que no sea una "cascara" que tan solo enmascare una estructura.

Implementar el TAD

Debe ser consistente con el Irep elegido
Debe respetar la interfaz propuesta por la cátedra
Debe cumplir satisfactoriamente el junit de la cátedra

Utilizar lineamientos de diseño que consideren las nociones de *Cohesión* y *Acoplamiento* explicadas en la teórica.
Se puede utilizar como guia el siguiente ejemplo:
https://prog2-ungs.github.io/codigo/PPTLSv2.pdf

Para la siguiente implementación

Jenga
    Torre

Torre
    Lista de Nivel, donde Nivel tiene entre 0 y 3 piezas

El Irep elegido (para esta implementación)

Ningún nivel puede estar vacio, ni si quiera el nivel n-1
Notar que “quitar” en realidad debería llamarse “mover” y que si bien cuando genero un nivel nuevo comienza vacio, al finalizar el “mover” el nivel no puede quedar vacio.

El Irep nada tiene que ver con las reglas elegidas.
Hay un Irep distinto para cada implementación.
El Irep determina que instancias de la estructura de representación representan instancias validas del TAD

Caso de uso de ejemplo1 "modo automático"

```
ArrayList<String> jugadores = new ArrayList<String>()
Jugadores.add("jug0");
Jugadores.add("jug1");
Jugadores.add("jug2");

Jenga unJenga = new Jenga(20,jugadores)  // Jenga de 20 niveles 0..19

While unJenga.ganador() <>
      unJenga.jugar()  // juegan los dos jugadores
      system.out.println(unJenga)

system.out.println(unJenga.ganador())
```

- jugar() debe mantener el Jenga consistente.
- "system.out.println(unJenga)" invoca a unJenga.toString(), que debería generar un string consistente en un "resumen" de todos los niveles.

Caso de uso de ejemplo2 "modo semi automático"

```
Jenga unJenga = new Jenga(20, jugadores)  // Jenga de 20 niveles 0..19
Nivel ni
While unJenga.ganador() <> ""

      //unJenga.quitar(2,0)    quita la pieza 0 del nivel 2*

      ni = unJenga.primerNivelPosible()**
      if (ni ¡= null)
            unJenga.quitar( ni , unJenga.piezaRecomendada(ni))***

      system.out.println(unJenga)

system.out.println(unJenga.ganador())
```

Ejercicio 2 Árboles

Implementar los siguientes métodos del TAD ABB<Integer> extends AB<Integer>.
Implementar también los métodos auxiliares o privados.

Sea abb una instancia de ABB<Integer>

a) Void eliminar(Integer elem): que elimina elem si existe. Si no existe no se puede arrojar una excepción.

b) boolean balanceado(): que devuelve verdadero si el árbol esta balanceado.

Debe hacerse en O(n).

Para lograr esto, modificar la estructura del nodo de manera de tener la altura en cada nodo

Nodo<Integer>
Integer Info
Nodo<Integer> izq
Nodo<Integer> der
Integer altura

De manera que se pueda evitar llamar llamar a altura() en balanceado.

Ayuda: modificar el agregar/ eliminar para actualizar la altura de los nodos afectados.
No se penaliza la complejidad de agregar / eliminar

En todos los ítems se debe justificar la complejidad de la solución elegida.

Respecto del Irep de ABB
En cualquier implementación se debe chequear que para todos los nodos n_i, los nodos a la izquierda de n_i sean menores, y mayores los que están a la derecha.
Además de chequear que para todo n_i IrepAB(n_i), que todo n_i sea AB.

Apéndice I: Condiciones de entrega y tutorial de cómo instalar Junit

https://prog2-ungs.github.io/tp/entregas

Apéndice II: Test obligatorio ej1

```
import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;


public class TestEj1 {


	private Jenga jenga1, jenga2;
	ArrayList<String> jugadores;

	@Before
	public void setUp() {

		jugadores = new ArrayList<String>();
		Jugadores.add("jug0");
		Jugadores.add("jug1");
		Jugadores.add("jug2");

		jenga1 = new Jenga(10 jugadores);
		jenga2 = new Jenga(10, jugadores);

	}


	@Test
	public void test1() {

		int alturaInicial = jenga1.altura();

		jenga1.Jugar();
		jenga1.Jugar();
		jenga1.Jugar();

		//System.out.println(alturaInicial +","+ jenga1.altura());
		// deberia cambiar la altura
		assertTrue(alturaInicial != jenga1.altura());

	}

	@Test
	public void test2() {
```

```
            int nivel = jenga2.primerNivelPosible();

            jenga2.quitar(nivel,0);
            jenga2.quitar(nivel,1);
            jenga2.quitar(nivel,2);


            System.out.println(jenga2.ganador());
            // deberia haberse caido el jenga!
            assertTrue(!jenga2.ganador().equals(""));

      }

}
```

Apéndice III: Test obligatorio ej2

```
import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class TestE2 {

      ABB abBalanceado, abVacio, abDesbalanceado;

      @Before
      public void setUp() throws Exception {
            abVacio = new ABB();

            abDesbalanceado = new ABB();
            abDesbalanceado.insertar(5);
            abDesbalanceado.insertar(3);
            abDesbalanceado.insertar(1);

            abBalanceado = new ABB();
            abBalanceado.insertar(8);
            abBalanceado.insertar(3);
            abBalanceado.insertar(10);
            abBalanceado.insertar(1);
            abBalanceado.insertar(6);
            abBalanceado.insertar(4);
            abBalanceado.insertar(7);
```

```
            abBalanceado.insertar(14);
            abBalanceado.insertar(9);
      }

      @Test
      public void testBalanceado()
      {
            assertTrue(abBalanceado.balanceado());
            assertTrue(abVacio.balanceado());
            assertFalse(abDesbalanceado.balanceado());
      }

      @Test
      public void testRebalancear()
      {
            abDesbalanceado.rebalancear();
            assertTrue(abDesbalanceado.balanceado());
      }

      @Test
      public void testIesimo()
      {
            assertEquals(abBalanceado.iesimo(0), new Integer(1));
            assertEquals(abBalanceado.iesimo(5), new Integer(8));
            assertEquals(abBalanceado.iesimo(8), new Integer(14));

            boolean thrown = false;
            try {
                  abBalanceado.iesimo(88);
            } catch (Exception e) {
                  thrown = true;
            }
            assertTrue(thrown);
      }

      @Test
      public void testIrep() {
            assertTrue(abVacio.irep());
            assertTrue(abDesbalanceado.irep());
            assertTrue(abBalanceado.irep());

            abBalanceado.romperIrep();
            assertFalse(abBalanceado.irep());
      }
}
```