Programación II

Práctica 02b: Tipos Abstractos de Datos (TAD) Avanzados

Versión del 01/01/2017

Ejercicio1: Agenda

Implementar una agenda basada en el TAD diccionario
La agenda deberá tener una clave basada en el dni
Y un significado basado en: String nombre, Integer teléfono y String dirección

Ayuda: Considerar utilizar otro TAD como abstracción del significado del diccionario.
Se puede modificar el TAD Diccionario, de manera que

```
Void agregar (Integer i, Significado s){}
```

Ejercicio 2: Matriz infinita de booleanos

El departamento de matemáticas de la UNGS nos pidió ayuda para implementar el TAD "Matriz infinita de booleanos"

La implementación(trivial) actual es la siguiente:

```
public class MIB {

	private int i;
	private int j;
	private Boolean [][] mat;

	public MIB(int i, int j){
		this.i=i;
		this.j=j;
		mat = new Boolean[i][j];
	}

	Boolean leerValor(int i, int j){
		return mat[i][j];
	}
	public void setearValor(int i, int j, Boolean x){
		mat[i][j]=x;
	}

}
```

El problema de esta implementación, es que ocupa demasiada memoria.

Por ejemplo "`MIB x = new MIB(100000,200000)`" arroja:
java.lang.OutOfMemoryError: Java heap space

Lo que se solicita es:

a) Implementar la función test() que implemente el siguiente testeo:

```
MIB x = new MIB(100000,200000);
x.setearvalor(5000,3000,true);
System.out.Println(x.leervalor(5000,3000)); // debe devolver true
System.out.Println(x.leervalor(5000,3001)); // debe devolver false
```

b) Hacer un diseño de un TAD que represente la matriz de manera mas eficiente.
Ayuda: Se puede asumir que la mayoría de los valores van a ser ***false***.

c) Calcular el orden de complejidad para los métodos ***leerValor*** y ***setearValor*** de la implementación trivial.

d) Calcular el orden de complejidad para los métodos ***leerValor*** y ***setearValor*** de la implementación mejorada.

Ejercicio3: Abstracción: Monopolio

Se desea modelar el clásico juego Monopolio, pero con la cantidad de casilleros definida por el usuario. Para ello se tiene un tablero de **n** posiciones y **2** jugadores.

Instancia: para n = 40

Sistema de juego simplificado:

Se pide por única vez el tamaño del tablero(n), y los precios de cada casilla.

En cada turno cada jugador tira un dado (un número de 1 a 5).

El tablero es circular, de manera que cuando se llega al final, se vuelva a dar otra vuelta.

Cada jugador comienza con $1000.

Cuando el jugador cae en una casilla, pueden pasar dos cosas:

1) Que la casilla no se haya comprado aun. En ese caso el jugador está obligado a comprarla y de ahí en más la casilla pertenecerá a dicho jugador
2) Que la casilla se encuentre comprada.
   Si la casilla pertenece al jugador que compro la casilla no se hace nada.
   Si la casilla pertenece al otro jugador, se debe pagar un alquiler, igual al 1% del monto de compra de dicha casilla.

El juego termina cuando alguno de los dos jugadores se queda sin dinero.

Esto puede suceder por alguno de los siguientes motivos:

1) El jugador no puede comprar una casilla.
2) El jugador no puede pagar el alquiler.

Diseño:

Diseñar e implementar el TAD Juego de "Monopolio" (Mono)

Es obligatorio que el TAD Mono utilice al menos otro TAD como TAD soporte.

Es decir:

-Que Mono no podrá estar implementado únicamente sobre los tipos primitivos de Java (ni java.util).

-Que Mono utilice varias clases para ser implementado.

Ayuda: Ver cuáles de las siguientes clases son necesarias para el TAD Mono, cuáles no, y por qué:

-Jugador

-Dado

-Tablero

-Reglamento

-Mono

Interfaz obligatoria:

```
Mono(n)                  //tamaño del tablero, donde n > 1
String ganador()         //devuelve el nro de jugador ganador o 0 si no hay
                         ganador por el momento
Void agregarCasilla(int casilla,double precio)
                         //Asigna el precio a la casilla
void jugar()             // tira los dos dados
String ver()             // muestra el estado del tablero y los jugadores
```

Ejemplo del código principal:

```
Mono mono = new Mono(6);            // Instancia: n == 6

Mono.agregarCasilla(0,100);   //La Casilla 0 vale $100
Mono.agregarCasilla(3,10);    //La Casilla 3 vale $10
while mono.ganador() == “”{         //como máximo hacerlo 1000 veces
      mono.jugar();                 // Los 2 jugadores “tiran” los dados
      System.out.println(mono.ver())     //Se  muestra  un  resumen    del
tablero
}
System.out.println(mono.ganador());
```

Implementación

Implementar un test que al menos pruebe el ejemplo del código principal.

Ejercicio4: Cohesión

a) Se proponen dos modelos para el TAD “Lista de coordenadas” LC

i) TAD LC

ArrayList<Integer> x

ArrayList<Integer> y

ii) TAD LC

ArrayList<Tupla<Integer,Integer>> coordenadas

¿Qué modelo tiene más cohesión? Justifique

Ejercicio5: Abstracción: Un TAD basado en otro TAD

Por lo general realizaremos diseños utilizando otro diseños ya probados, como soporte.

1) Implementar el TAD Conj<T> sobre el TAD ArregloEstatico <T> de la teórica.
   Al que llamaremos ConjEstatico<T>

Ayuda: Considerar agregar al TAD ArregloEstatico<T> el método

ArregloEstatico<T> copiar(int nuevotamano)

que dado un ArregloEstatico<T> t1 devuelve otro arreglo t2 mas grande que t1, que conserva los elementos de t1

De esta manera el conjunto puede crecer “indefinidamente”.

Ejemplo para el arreglo estático de java

```
int[] a = {1, 2, 3};
// hago una copia de a con un elemento mas
a = Arrays.copyOf(a, a.length + 1);
for (int i : a)
    System.out.println(i);
```

2) Calcular la complejidad de

   ConjEstatico.agregar()
   ConjEstatico.iesimo()

   ConjEstatico.agregar()
   ConjEstatico iesimo()