Programación II

Práctica 08: Tipos paramétricos

Versión del 31/03/2015

Introducción

Como se vio en las clases teóricas, siempre que sea posible, es mejor que el TAD reciba el tipo como parámetro, a trabajar con un tipo en particular.

Por ejemplo, el TAD Conjunto<***Genero***> es más general que el TAD ConjInt de la práctica anterior.

A ***Género*** se la denomina variable de tipo porque su dominio son los tipos de datos.
Genero puede ser cualquier Objeto predefinido de java(Integer, Boolean, Char, etc), pero también puede ser un Objeto o clase[1] del usuario.

Siguiendo con el ejemplo:

(1) Conjunto = new Conjunto<Integer>
(2) Conjunto = new Conjunto<Nat> // Recordar el TAD Nat de la practica anterior.

La implementación de Conjunto<G> va a ser muy similar que la de ConjInt para el caso (1).
El caso (2) se verá en las siguientes prácticas.

[1] Sintácticamente la variable de tipo es una clase o objeto en lugar de un TAD que puede involucrar una o más clases.

Ejercicio1: Generalizar

Generalizar los TAD de la práctica de TAD haciendo que reciban el tipo como parámetro.

Por ejemplo, El Árbol binario de búsqueda cuya especificación era:

Especificacíon TAD ANBint

```
ABBint() {}
void eliminar(Integer i) {}
void agregar(Integer i) {}
Integer buscar(Integer i) {}  //devuelve 0 si no encuentra
```

De ahora en adelante tiene que ser:

```
public class ABB<T extends Comparable<T>> //El tipo T tiene que ser
                                          comparable.

ABB() {}                //constructor de ABB de tipo T
void eliminar(T t) {}
void agregar(T t) {}
Nodo buscar(T t) {}  //devuelve null si no encuentra t
```

Ejercicio2: Tulpa

Acceder directamente a las variables privadas de un TAD es una mala práctica.
Entre otras cosas permite que los que utilizan el TAD estropeen su consistencia (Invariante de representación).
De aquí en adelante utilizaremos métodos para acceder y modificar el estado del TAD.

Especificación

```
Tupla(T1 x, T2, y){}          // T1 y T2 son variables de tipo
T1 getX()   {}
T2 getY()   {}
void setX(T1 x) {}
void setY(T2 y)   {}
```

a) Implementar Tupla<T1,T2>
b) Implementar el método Bool mismoGenero() que devuelve True si T1 == T2
   Ayuda: Investigar la palabra clave *instanceof*.

Ejercicio3: Coordenada

Vamos a especializar la Tupla en una Coordenada.
Como queremos poder sumar, las Coordenas no van a poder ser paramétricas.

Especificacíon

```
public class Coordenada extends Tupla

public Coordenada(Integer x,Integer y){...}

public void sumar(Integer x,Integer y){...}
```

a) Implementar Coordenada

Ayuda: Utilizar la palabra clave ***super*** cuando sea necesario.

b) Modificar la implementación de Coordenada, de manera que pueda operar tanto con int como con float.

Ejercicio4: Diccionario sobre ArrayList de Tuplas

a) Re-Implementar el TAD Diccionario, de manera que la clave y el significado sean genéricos.

```
public class Diccionario<CLAVE,SIGNIFICADO> {
	ArrayList<Tupla<T1,T2>> datos;
}
```

b) Re-Implementar el TAD Diccionario sobre conjunto de Tuplas

```
public class Diccionario<CLAVE,SIGNIFICADO> {
	Conjunto<Tupla<T1,T2>> datos;
}
```

Que implementación tiene mejor abstracción? Porque?