Ing. de Sistemas e Informática
Estructuras de Datos y Algoritmos
Jorge Mario Londoño P.

Ejercicios 1

ADTs y Estructuras básicas

Tipos de datos abstractos

1. Una aplicación para un parqueadero necesita mantener el número de espacios disponibles para visualizarlos en la la puerta de entrada. Observar que la aplicación debe iniciar con el total de espacios y decrementarse cuando entra un carro hasta que no haya espacios libres. Asi mismo cuando los carros salen se debe incrementar.

a. Diseñar un ADT para representar el número de espacios. Indicar constructor y API en forma de una `Interface`.
b. Escribir un cliente para hacer pruebas unitarias del ADT.
c. Implementar el ADT como una clase de Java.

2. Un *número racional* es aquel que puede escribirse como el cociente de 2 enteros, es decir r es racional si existen enteros p,q tales que r=p/q.
Diseñar un ADT Racional para representar los números racionales presevando los 2 enteros que lo conforman. Implementar las operaciones aritméticas básicas (+,-,*,/) con Racionales. Nota: No convertir los racionales a tipo float o double para implementar las operaciones, pues se pierde la precisión de la representación como fracción.
Implementar pruebas unitarias que permitan comprobar el correcto funcionamiento de la implementación para algunos casos ejemplo.

3. Los números racionales constituyen un *orden total* respecto a las relaciones mayor, menor (<, >). Implementar la interfaz `Comparable` en el ADT Racional, con el fin de poder comparar cualquier par de números racionales. Debe ser claro que no se deben convertir en ningún momento los racionales a tipos de punto flotante (double,float) para de esta forma garantizar la precisión de la implementación.

```
public interface Comparable<T>
int compareTo(T otro)    // Devuelve -1 si this<otro,
                         // 0 si son iguales, 1 si this>otro.
```

a. Hacer la implementación de la interfaz Comparable.
b. Hacer un cliente de prueba para validar que la interfaz opere correctamente con varios racionales. Pensar si hay posibles casos especiales.

4. Se dice que un `String` es una rotación circular de otro si los caracteres de ambos coinciden al desplazarlos un número dijo de posiciciones. Por ejemplo, ACGTATTG y GTATTGAC son rotaciones el uno del otro, desplazadas dos lugares. Diseñar una función de biblioteca que determine si un `String` es una rotación de otro y retorne el número de lugares desplazados. En caso negativo, returnar -1.

5. En el texto guia se da una implementación de un evaluador de expresiones infijas con paréntesis completos. Adaptar esta implementación para hacer un evaluador de expresiones infijas con números racionales.

Estructuras de datos básicas

1. Se quiere implementar una de las estructuras básicas (Bag, Queue, Stack) con tipos genéricos para garantizar la integridad de los datos en la estructura, sin embargo, el ADT a almacenar tiene múltiples implementaciones (e.g. Punto2DCartesiano y Punto2DPolar). Cómo sería posible lograr esto? Indicar los cambios necesarios para implementar el tipo genérico.
2. Considerar que tipo de estructura sería aplicable a cada uno de los problemas siguientes, justificando brevemente:
   - En un help-desk se reciben peticiones de soporte. Las peticiones deben atenderse en el orden en que fueron recibidas.
   - Se quiere guardar el historia de notas, para sacar promedios por estudiante, por grupo, por curso.
   - Una bodega recibe cajas (con un número de stock) y las guarda en forma vertical. Así mismo, cuando se despachan, cada camión se carga con las cajas empezando desde la parte superior.

Preguntas conceptuales

1. Son ADTs los tipos de datos primitivos del lenguaje Java? (Explique brevemente)
2. Se pueden tener métodos estáticos en clases que implementen un ADT? Y que tal métodos privados?
3. Los parámetros de tipo permiten postergar la decisión sobre el tipo de datos que va a contener una colección hasta tiempo de ejecución. Suponga que se quiere hacer una colección con datos de varios tipos. Qué alternativas hay para lograrlo?
4. Explicar brevemente las diferencias entre las interfaces y la herencia (también llamadas `herencia de interfaces` y `herencia de implementación`)