Primer examen integrador - Programación II
Lunes Noche - 1C 2025

Profesor Monzón, Nicolás Alberto

14 de abril de 2025

Requisitos para aprobar el integrador

A. Por actos de deshonestidad académica será sancionado.

B. Para poder aprobar este examen Ud. deberá estar administrativamente en condiciones de poder rendir el examen. En caso de no estarlo y rendir el examen, este no va a ser corregido y quedará anulado.

C. Deberá comprimir la carpeta `src` de su proyecto junto a los `txt` utilizados (también se admiten `pdf` y `MD`). El archivo comprimido deberá tener como nombre el número de grupo. Deberá enviarlo por mail a `nimonzon@uade.edu.ar`.

D. Deberá desarrollar un set de prueba para demostrar que funciona su código (no se piden test unitarios, pero si alguna prueba en el `main` del proyecto).

E. Solo podrá utilizar técnicas vistas en este curso. Está prohibido el uso de librerías, estructuras que vienen por defecto en la JRE y el uso de genéricos. No respetar este punto es suficiente para desaprobar el integrador.

F. Deberá ser entregado dentro de las 3 horas y 30 minutos. a partir del horario de inicio.

G. CONDICIONES PARA APROBAR: el examen se mide con una escala logarítmica donde obtener al menos 60 puntos corresponderá a un 4 y se deberá cumplir con todos los puntos anteriores. En caso de no cumplir con alguno de los puntos anteriores, el examen queda desaprobado sin excepción.

Considere los siguientes predicados sobre los legajos de los integrantes.

$\mathcal{P}(\texttt{a})$ = la suma de los legajos módulo 3 es 0

$\mathcal{P}(\texttt{b})$ = la suma de los legajos módulo 3 es 1

$\mathcal{P}(\texttt{c})$ = la suma de los legajos módulo 3 es 2

1. Ejercicio 1 (40 %)

Crear el TDA `VersionedStack`, imitando (en lo que se pueda) la implementación de `Stack` desarrollada en clase. Será una estructura lineal destructiva, que además permitirá moverse entre versiones. *Tip: pensar en un CRUD de versiones.*

Cada vez que se realice una operación que modifique la estructura, se deberá generar una versión. Dado que cada versión se refleja con cada acción que puede modificar la estructura, no se pide aplicar inmutabilidad.

Se espera:

- Definir un conjunto de métodos que permita resolver la mayor cantidad de algoritmos.
- Si $\mathcal{P}(\texttt{a})$, permitir borrar una versión. Si $\mathcal{P}(\texttt{b})$, permitir crear una versión nueva a partir de otra existente. Si $\mathcal{P}(\texttt{c})$, a elección.
- Agregar a la definición de los métodos cuáles son las precondiciones, postcondiciones y estrategia de implementación.
- Indicar además en un archivo `txt` (`pdf` y `MD` también están permitidos) cuáles son los invariantes que consideran para la estructura.

2. Ejercicio 2 (20 %)

Modificar la estructura del ejercicio 1, tal que

- Si $\mathcal{P}(\texttt{a})$, considerar que el total de versiones no puede definir o limitar el tamaño del arreglo donde están siendo almacenadas. *Deberán crear el arreglo con tamaño 1 inicialmente, y cada vez que se crea una versión nueva, el arreglo debe incrementarse 1 en longitud y agregar la nueva versión. Como extender un arreglo no es posible porque tienen tamaño fijo, deberán crear un nuevo arreglo que copie el contenido anterior.*
- Si $\mathcal{P}(\texttt{c})$, considerar que como las versiones no se van a modificar, el tamaño del arreglo creado para almacenar los valores de cada versión no necesita ser más grande que la cantidad de elementos de la pila. *Guardar las versiones sin depender de* `count`.

Si $\mathcal{P}(\texttt{b})$, a elección.

3. Ejercicio 3 (10 %)

Modificar la implementación estática de `Queue`, para que en lugar de usar un arreglo nativo, se utilice el TDA `VersionedStack`.

4. Ejercicio 4 (10 %)

Desarrolle un método `map` que permita convertir una instancia de `VersionedStack` a una instancia de `A`, y otro método `map` que permita convertir una instancia de `A` a una `VersionedStack`.

- Si $\mathcal{P}(\texttt{b})$, considerar `A` = `Stack`.
- Si $\mathcal{P}(\texttt{c})$, considerar `A` = `Queue`.

Si $\mathcal{P}(\texttt{a})$, a elección.

5. Ejercicio 5 (10 %)

Crear el TDA `Coord` que cumpla las propiedades usuales de una coordenada de 3 componentes. Dar una implementación, invariantes, precondiciones, postcondiciones y definir una clase utilitaria con una función que proyecte la i-ésima componente (en este caso $i = 0$, $i = 1$, o $i = 2$).

6. Ejercicio 6 (10 %)

Crear una clase utilitaria que permita calcular la distancia entre dos coordenadas, saber si dos coordenadas son iguales y decidir si una coordenada pertenece al primer octante.