JESUÏTES El Clot
Escola del Clot

# M05. Entorns de desenvolupament

## UF2. Optimització de programari

## 1. Jocs de proves amb JUnit

Hèctor López
hector.lopez@fje.edu

## EXERCICI 0. Queue & Stack

Aquest exercici consta de dues fases.

**Fase 1:**

Per parelles, desenvolupeu els següents tipus abstractes de dades (un cadascú), amb els mètodes que s'especifiquen a continuació. Podeu especificar la mida estàtica màxima que vulgueu per a l'estructura de dades.

**Cua (queue):** Estructura de dades que permet emmagatzemar elements de tal manera que el primer que entra és el primer que surt (FIFO – *First In First Out*).

Mètodes del TAD **Cua**:

```
boolean add(T a);        //Retorna si l'element ha entrat a la cua o no.
T get();                 //Retorna i elimina l'element que toca sortir de la cua.
                         //Retorna null si la cua està buida.
boolean isEmpty();       //Retorna si la cua està buida o no.
int getSize();           //Retorna la mida màxima estàtica de la cua.
```

**Pila (stack):** Estructura de dades que permet emmagatzemar elements de tal manera que el darrer element que entra és el primer element que surt (LIFO – *Last In First Out*).

Mètodes del TAD **Pila**:

```
boolean push(T a);       //Retorna si l'element ha entrat a la pila o no.
T pop();                 //Retorna i elimina l'element que toca sortir de la pila.
                         //Retorna null si la pila està buida.
boolean isEmpty();       //Retorna si la pila està buida o no.
int getSize();           //Retorna la mida màxima estàtica de la pila.
```

Quan hàgeu implementat els TADs, intercanvieu els codis entre vosaltres i passeu a la fase 2 de l'exercici.

**Fase 2:**

Implementeu una nova classe amb mètodes de test per a provar totes les casuístiques segons l'especificació de cadascuna de les funcions de l'estructura de dades que us hagi passat el vostre company.

Exemple de classe de test per al TAD pila:

```
public class PilaTest {
	Pila pila = new Pila();

	@Test
	public void testPush() {
		//codi
	}
	...
}
```

### EXERCICI 1. Test mètodes `StringBufferTest`

Dissenyeu i codifiqueu un test unitari **`StringBufferTest`** que permeti provar els mètodes: *`charAt (int index)`*, *`setChatAt (int index, char ch)`* i *`append(String str)`* de la classe `StringBuffer` de l'SDK de Java.

Podeu consultar l'API aquí.

A banda dels que considereu necessaris, definiu obligatòriament els mètodes de test:

- **`testAppendWithNullString`**, que provi el mètode `append(String str)` quan se li passa per paràmetre un valor `null` enlloc d'un `String`.
- **`testCharAtWithInvalidIndexes`**, que provi les excepcions que pot llençar el mètode `charAt(int index)`.

### EXERCICI 2. Test de `MyDate`

Donats els fitxers incomplerts (noteu que contenen errors) **MyDate.java** i **MyDateTest.java** (els trobareu al grup aula de l'assignatura), se us demana:

**2.1.** Provar el mètode estàtic: `boolean isValidDate(int day, int month, int year)`. Completeu el test unitari **`MyDateTest`** per cobrir les diferents classes d'equivalències.

| CE vàlides | CE invàlides |
|---|---|
| 1 <= dia <= 31 &&<br>1 <= mes <= 12 &&<br>any >= 0 | dia < 1 |
| | dia > 31 |
| | mes > 1 |
| | mes > 12 |
| | any > 2015 |
| | any < 1812 |
| mes € {2, 4, 6, 9, 11} → 1 <=dia<=30 | mes € {2,4,6,9,11} && (dia<1 \|\| dia>30) |
| (mes=2) && (no any traspàs) → dia<=28 | (mes=2) && (no any traspàs) && (dia>28) |
| (mes=2) && (any traspàs) → dia <= 29 | (mes=2) && (any traspàs) && (dia>29) |

**2.2.** Definir el mètode **`nextDay`** (o l'endemà) dins la classe **`MyDate`**. Posar-lo en condicions, amb un nou test unitari **`MyDateNextDayTest`**.

**EXERCICI 3. Test d'un custom `String`**

Tenim la necessitat de funcionalitats de manipulació de cadenes de caràcters que van més enllà de les classes estàndard **`String`** i **`StringBuffer`** (package `java.lang`). Per aquest motiu desenvoluparem les classes **`MyString`** i la classe de test **`MyStringTest`**.

Al contrari que la classe **`String`**, i pel bé de la seva eficiència, **`MyString`** serà ''mutable'': implementareu doncs reutilitzant la classe **`StringBuffer`** (noteu que no podreu heretar d'ella, ja que la classe **`StringBuffer`** està declarada com a **`final`**).

Exemple:

```
public class MyString {
	// instance variables
	private StringBuffer _sb;

	/**
	* Constructor for objects of class MyString
	*/
	public MyString(String s) {
		// initialise instance variables
		_sb = new StringBuffer(s);
	}

	// Accessors
	public String getString() { /* TODO */ }
	public void setString(String s) { /* TODO */ }

	// ... TODO
…
```

A continuació la llista de funcionalitats que manquen, i que desitgem:

**3.1.** Convertir tots els caràcters de la cadena a majúscules: **`void toUpperCase()`**.
**3.2.** Obtenir el nombre d'ocurrències d'un caràcter donat: **`int numberOfChar(char c)`**.
**3.3.** Obtenir el nombre de vocals (Y considerada consonant): **`int numberOfVocals()`**.
**3.4.** Obtenir el primer mot de la cadena: **`String getFirstWord()`**.

De manera incremental, per a (els accessors i) cada funcionalitat **`func<`**i**`>`**:

**3.5.** Definir un mètode de test **`testFunc<`**i**`>`** dins la classe **`MyStringTest`** (la qual hauria d'ajudar a definir la interfície del mètode **`func<`**i**`>`** a provar).

**3.6.** Implementar el mètode **`func<`**i**`>`** dins la classe **`MyString`** (es demana que sigui de la manera més simple possible).

**3.7.** Testejar.

**3.8.** Optimitzar (si és possible i/o necessari) la implementació de **`func<`**i**`>`** dins la classe **`MyString`**.

**3.9.** Retornar al punt 3.7.

**3.10.** De manera, opcional, podeu definir més mètodes de test dins la classe **`MyStringTest`** per provar una mateixa funció.