

Estructura de datos - UFM

David Gabriel Corzo Mcmath

2020-Jan-06 07:08:59

Índice general

1. Clase introductoria - 2020-01-06	5
1.1. Datos del auxiliar	5
1.2. Información preliminar	5
1.3. Dos formas de pasar parámetros a una función	5
1.4. Ver:	5
2. Clase introductoria - 2020-01-08	7
2.1. Charla de programas del curso	7
3. Clase de avance de tarea # 2	9
4. Clase	11
4.1. Unit testing	11
4.2. Postman	11
4.3. Jmeter	11
5. Clase - 2020-01-20	13
5.1. Data structure	13
5.1.1. SOLID	13
5.1.2. The 3 steps	13
5.2. Elementary data structure organization	14
5.2.1. Clasificaciones	14
5.3. Abstract data types	14
5.4. Tarea	14
5.4.1. Asert - es parte de la prueba de integración	14
6. Clase - 2020-10-22	15
6.1. Estructuras de datos \neq Algoritmos	15
7. Clase - 2020-01-27	17
7.1. Array	17
7.1.1. Storing values in the array	17
7.1.2. Caché	18
7.2. Two-Dimensional Array	18
7.3. Multi-Dimensional array	18

Capítulo 1

Clase introductoria - 2020-01-06

1.1. Datos del auxiliar

- tortola@ufm.edu
- 56904805

1.2. Información preliminar

- Sugerencia de lenguaje: Java
- Se puede utilizar el lenguaje que quiera. Ideal Java por el aspecto de ser orientado a objetos.

1.3. Dos formas de pasar parámetros a una función

- ```
num = 0;
func SumOne(n){
 return n + 1;
}
num_2 = SumOne(num)
print(num)
print(num_2)
```

| Por valor       | Por referencia  |
|-----------------|-----------------|
| $\ddot{\cup}^0$ | $\ddot{\cup}^1$ |
| $\ddot{\cup}^1$ | $\ddot{\cup}^1$ |

Los de valor no modifican num, los de por referencia no lo modifican y otra lugar en memoria.

### 1.4. Ver:

- Postman para API's
- Jmeter
- Spring: para API's en Java



## Capítulo 2

# Clase introductoria - 2020-01-08

### 2.1. Charla de programas del curso

- IntelliJ Ultimate: IDE
- Jetbrains: conjunto de servicios para estudiantes
- Junit: para pruebas unitarias.



## Capítulo 3

# Clase de avance de tarea # 2



# **Capítulo 4**

## **Clase**

### **4.1. Unit testing**

### **4.2. Postman**

- Pasos para probar un API:
  - ↓ New → Crear una colección.
  - ↓ Poner el URL al API.

### **4.3. Jmeter**

- ↓ Test plan es mi route de los tests.
- ↓ Add → Thread → Thread groups
- ↓ Basic Test → Sampler → HTTP request
  - ↓ Server name or IP: nombre de dominio
  - ↓ Numero de puerto
- ↓ Agregamos un listener: Basic Test → Listener →



# Capítulo 5

## Clase - 2020-01-20

### 5.1. Data structure

- Es simplemente cómo en memoria tengo esa información disponible mientras estoy ejecutando esa información.
- *Ejemplo: Google, si google hubiera guardado una llave primaria por todo lo que existe estaríamos aún esperando, separan el software y las clasifican en diferentes estructuras de datos; es diferente transformar la data que manipularla.*

#### 5.1.1. SOLID

1. Single responsibility:
  - No hacer dos o tres métodos en una clase llamada “tarea1”.
2. Open - Close
  -
3. L is for substitution
4. I
5. D

#### 5.1.2. The 3 steps

- Analyse the problem to determine basic operations that must be supported to interact with the data.
- Quantify the resource constraints for each operation.
- Select the data structure that best meets these requirements.

#### Recomendaciones

- “Todo trabajo tiene su estructura de datos perfecta.”

## 5.2. Elementary data structure organization

- Data
- Record
- File
- Key
- Values

### 5.2.1. Clasificaciones

- Fundamental data types: Linear, Non-linear.
- Primitive: int, char, float, double.
- Non-primitive: linked list, array.
- Data structures: **Nos preguntamos:** ¿una clase es una estructura de datos? depende, para el compilador sí; en memoria sí es una estructura de datos; Para nosotros no es una estructura de datos.

## 5.3. Abstract data types

- Solo la voy a usar.

## 5.4. Tarea

- Testing*  
■ Integration  $\nearrow \neq \searrow$  Unit
- Prueba unitaria va directo a la clase. En este probas tu API
  - Prueba de integración va al medio de exposure.

### 5.4.1. Asert - es parte de la prueba de integración

- Es una prueba de confirmación, se puede colocar que algo está incluído.
- Es pasarle los asserts a Jmeter.

# Capítulo 6

## Clase - 2020-10-22

### 6.1. Estructuras de datos $\neq$ Algoritmos

- *Interesante:* Algoritmos no implican código.
- 20 % de esfuerzo va a construir el 80 % del trabajo.
- *Definición de “Modularización”:* proceso de dividir un algoritmo en varios módulos. Divide el problema en micro-problemitas.
- Top down approach: voy considerando las funcionalidades.
- Bottom up approach: decidí hacer una funcionalidad específica.
- *Consultar el siguiente recurso:*
  - Behaviour driven design
  - Test driven design
- Algoritmo, tres bloques:
  - Recursos que va a necesitar
  - Tiempo que se va a tardar
  - Cantidad de memoria que se va a utilizar
- Time complexity:
  - Running times
  - Nos dice qué comportamiento va a tener
- Time-Space Trade-off:
  - Va ser útil tener una estructura de datos que pueda ocupar poco en memoria y al mismo tiempo tener una cantidad grande de operaciones.
- Expressing time and space complexity:
  - Tiempo es muy importante.
- Eficiencia de una algoritmo:
  - Un ciclo multiplica la complejidad del algoritmo.
  - Algunos algoritmos van a tener instrucciones inefficientes, cargar en memoria todos los datos y filtrarlos, esto es ineficiente.

■ Algorithm efficiency:

- Linear loops: la eficiencia es linal  $y = x$ .
- Logarithmic loops: la eficiencia es logarítmicas  $f(n) = \log_{10}(n)$
- Quadratic & doendent quadratic:  $f(n) = n^2$

■ Notations:

1. Big theta ( $\Theta$ ): en el peor de los escenarios, me lleva un poco a la realidad, enfocada en entender los dos límites, inferior o superior, que son el mínimo p máximo que me voy a tardar.
  2. Big O notation ( $O$ ): lo contrario que omega, se basa en el peor escenario, **Nos preguntamos:** ¿cuanto se va a tardar en el peor de los casos?; **Pésima condición**.
  3. Big omega notation ( $\Omega$ ): el mejor escenario, nos da una notación en la que se puede decir que en el mejor de los casos dado un input  $g(n)$  se va a tardar tal cantidad de tiempo. Si todo va bien → nos va dar una guía de escenarios óptimos en los que van a resultar. **Si todo va bien puedo llegar a tal punto.**
- Un algoritmo funcional va de la mano de algoritmos no funcionales.
- **Nos preguntamos:** ¿Cómo verifico si lo que cree es lo que me imaginé inicialmente?

|                         |                                                              |                               |
|-------------------------|--------------------------------------------------------------|-------------------------------|
| Behaviour Driven Design | Specification Testing<br>Integration testing<br>Init testing | ↓ Load testing<br>↑ Profiling |
| Test driven design      |                                                              |                               |

■ Observaciones:

- Cosas difíciles de determinar es qué estructura de datos usar.
- **Definición de “specification testing”:** basarse en lo más alto de especificación de un user story.
- **Definición de “integration testing”:** CI, continuous integration.
- **Definición de “load testing”:** simular muchas interacciones con tu sistema, mecanismo que genera de manera dinámica simulación de uso en exceso.
  - Genero volumen
  - Genero Uso
  - Pero no me dice qué pasó solo en  $n$  cantidad de usuarios en un determinado momento con mi aplicación.
- **Definición de “profiling”:** enfocado a entender qué está pasando en tiempo de ejecución.
  - Call tree view
  - CPU usage
  - Memory usage
  - Time spent
  - Networking

■ Transformaciones:

- Las manipulaciones en transformaciones

# Capítulo 7

## Clase - 2020-01-27

### 7.1. Array

- Colección de datos del mismo tipo.
- En memoria se hace de manera consecutiva.
- Con arraylists no se almacena de manera consecutiva.
- Elementos:
  - Data type: Any valid data type.
  - Name: How we'll identify.
  - Size: Maximum numbers of values to hold.
- En el procesador:
  - Tiene una eficiencia mayor.
  - Como se guarda consecutivamente solo se tiene que buscar una vez en memoria por ser consecutiva.
  - Por eso se necesita declarar con cuántos elementos inicializar el array.
- Tener presente cuánto ocupa en memoria cada tipo de elementos.
- La función .length recorre todo el array y retorna el número de elementos que tiene.

#### 7.1.1. Storing values in the array

- Cuando ingresamos información a un array lo hacemos por orden de índices.
- Accesar: por el índice [i].
  - `System.out.println(A[i]); // Prints the element stored in that index in memory.`
- Operations on arrays:
  - Deleting elements: cuando se elimina un elemento, se tiene que hacer un corrimiento de las cantidades correspondientes; si por ejemplo tiene que un array de  $n$  elementos, se desplaza indices  $n$  por los  $n + 1$ .
    - `A[i] = A[i + 1];`
- Merging arrays:
  - `array1;`

- array2;
  - merged = array1 + array2;
- Passing arrays to functions:
1. Passing entire arrays:
    - Le pasa todo el arreglo, con todo y la posición de memoria.
  2. Passing individual elements: esto permite trabajar el contenido del array por aparte, creo una copia y trabajo con la información, entonces no modiflico el array original, si paso la dirección de memoria del índice o de la posición de memoria si voy a alterar el arreglo original.
    - Passing data values
    - Passing addresses

### 7.1.2. Caché

1. Almacenamiento
2. Temporal
3. Fácil Acceso

#### Beneficios

- Tener más efectividad.
- Reduce el impacto cuando tenemos que interactuar .
- Está enfocado en leer.

## 7.2. Two-Dimensional Array

- Almacena matrices en memoria, tiene varias implicaciones:
  - Tener contabilidad de dos indices.
  - Agregar una línea implica agregar n cantidad de el array anidado.
- Se almacena de la misma manera que un array de una dimensión.
- Cuando se almacena un array bidimensional, se hacen operaciones matemáticas para ubicar en memoria los indices.
- Inicialización:
  - Se declaran valores al momento de declarar.
  - Se declaran valores parametrados.

## 7.3. Multi-Dimensional array

- Multidimensional array:
- *Ejemplo: Graficador de imágenes*

$$\text{grafo} \quad \overbrace{[t_1]}^{\text{Temporalidad}} \quad \underbrace{[x_1][y_1]}_{\text{Cuadrícula}} = \underbrace{\text{Color}}_{255}$$

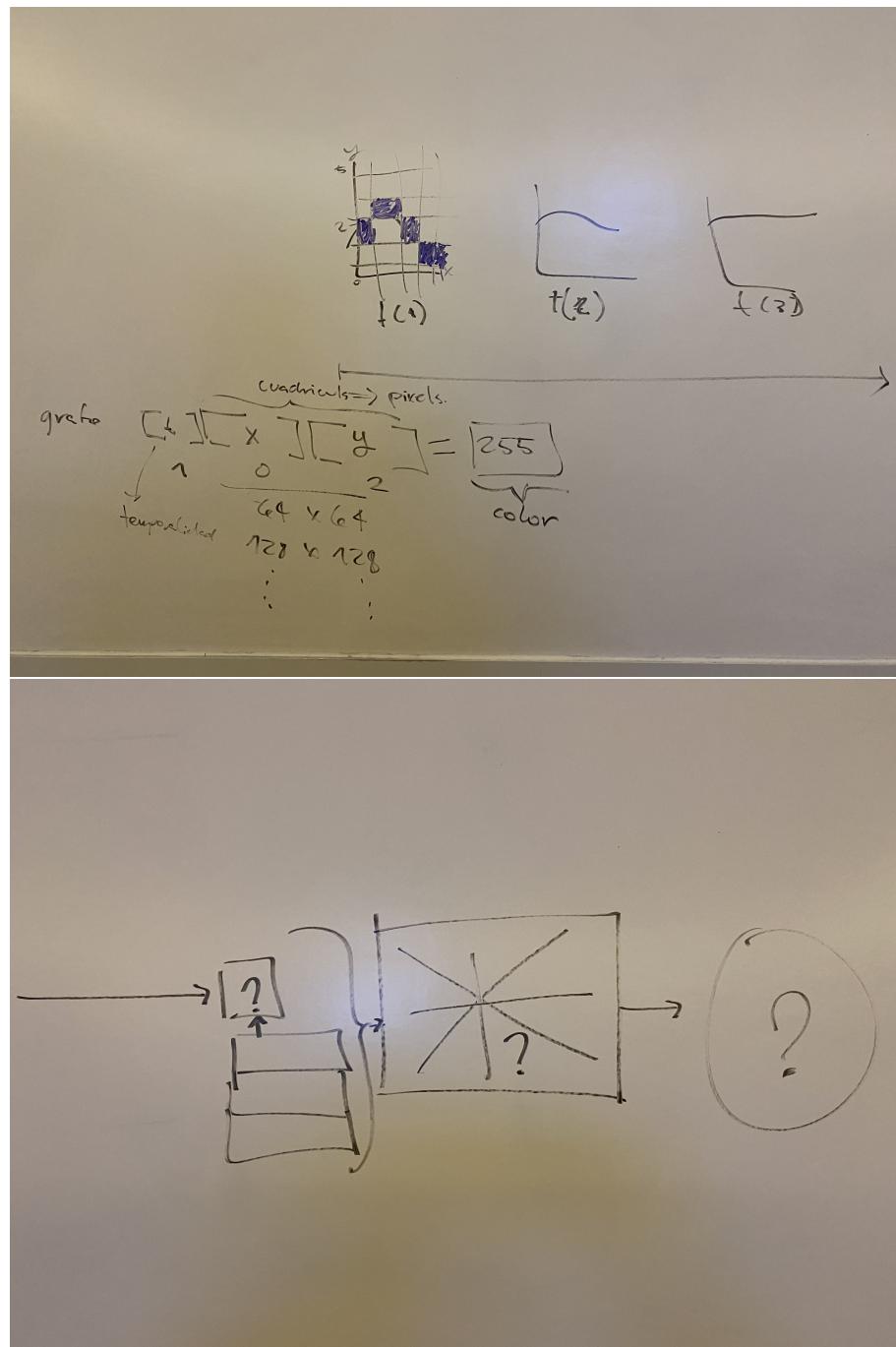


Figura 7.1: Ejemplo de multidimensional arrays &amp; el black box exercise

- Entonces el tiempo puede tener manifestación en un gif, las imágenes que resultan se pintan en la pantalla. Mientras más pixeles tenemos mejor la calidad.
- Los arrays multidimensionales podemos combinarlos con objetos :
  - Nos va permitir más que un solo valor.
  - Se puede almacenar objetos en el array, por lo general se va a tener que tener el mismo tamaño.
  - En Java se puede declarar un array de tipos de datos objetos.