

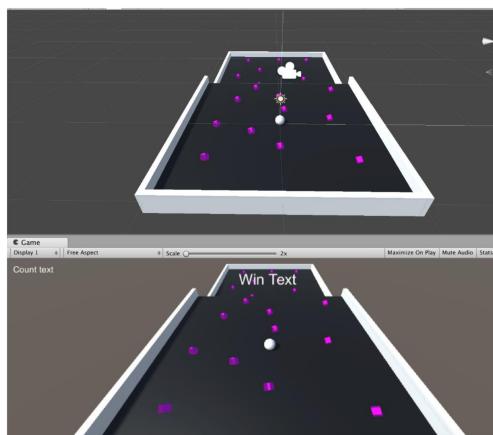
Journal – Programación 2

3.14Q2

October 15, 2018

Sprint 1

1. Se presentó un demo de lenguajes **interpretados** en el que se mostró sus ventajas y desventajas frente a los lenguajes **compilados**. Se hizo fallar una aplicación web hecha en Python para demostrar que un lenguaje interpretado va a arrojar error hasta el momento que se interprete esa línea a diferencia de un compilado donde se arroja el error cuando se intenta compilar.
2. Se realizó un video en *Final Cut Pro X* explicando las ventajas y desventajas de los lenguajes interpretados y compilados.
3. Se realizó la serie de tutoriales del proyecto *Roll a Ball* de Unity para entender los conceptos de **clases, objetos y atributos**.



4. Se realizó un cuadro comparativo entre programación estructurada y programación orientada a objetos para entender las diferencias entre estas.

PROGRAMACIÓN >OOP<	PROGRAMACIÓN ESTRUCTURADA
<ul style="list-style-type: none">• Los programas son más fáciles de entender.• El seguimiento de los errores del programa se facilita debido a su estructura.• La estructura del programa es más clara puesto que las instrucciones están más relacionadas.• Unidad de programación: la función.• Los códigos son más fácilmente entendibles dado que tienen una coherencia (t+4)	<ul style="list-style-type: none">• Intuye datos y procedimientos en una clase, por lo que se complica su entendimiento.• No es tan fácil encontrar los errores, pues su estructura no es sencilla.• Los procedimientos están separados y sin relación.• Unidad de programación: la clase.• Los códigos son más compactos pero más difíciles en su comprensión dada la lógica usada en la relación de objetos.

5. Se hizo un doodle que contiene el funcionamiento y la arquitectura del lenguaje de programación Java y se explicó en una presentación.

Sprint 2

1. Cada miembro del equipo realizó 15 preguntas de los capítulos 1 al 3 de *Think Java*.
2. El equipo aprendió acerca de los tipos de datos primitivos de Java, para lo cual se creó un documento y una presentación que contienen dicha información y se discutió entre los miembros del equipo.
3. Se realizó este documento PDF que lista los *Stories* en *done* de este *Sprint*.

Sprint 3

1. Se realizó un programa para entender que es el I/O de Java.

```
1 import java.util.Scanner;
2 class JavaIO{
3     public static void main(String[] args){
4         Scanner sc = new Scanner(System.in);
5
6         System.out.println("Ingrese un texto: ");
7
8         String texto = sc.nextLine();
9
10        System.out.println("Tu nuevo texto es: "+texto.toUpperCase());
11    }
12 }
```

2. El equipo buscó entender las estructuras de datos y se realizó el ejercicio de números perfectos.

```
↳ java NumerosPerfectos
Por favor ingrese un numero
1000
listado de perfectos menores a 1000: 6-28-496
```

3. El equipo quiso comprender la estructura de los objetos en Java para lo cual se realizó una infografía explicando la gramática de la declaración de métodos.



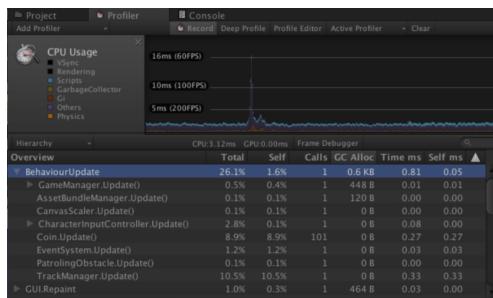
- Se buscó entender qué es una clase y un objeto. Luego se explicó ambos por medio de una cátedra en la cual se presentó una demostración en vivo de una aplicación nativa de *Android* mostrando así la independencia de objetos y sus atributos.



- Se realizó este documento PDF que lista los *Stories* en *done* de este *Sprint*.

Sprint 4

- Se investigó sobre las funciones del *Garbage Collector* en Java, los momentos en los que se ejecuta y cuáles son los candidatos a eliminación de la memoria.



- Se realizaron los ejercicios 003 de MiU para comprender qué es un *Constructor* y un *Destructor* y se subió al repo en GitHub.

This screenshot shows a terminal window with the title 'java ToDoList'. The command 'java ToDoList' was run, and the output shows the usage of the application. It includes instructions for creating a To Do List, adding tasks, and printing the list. A specific task named 'Task1' is added to the list.

```

java ToDoList
+ 003 x java ToDoList
andres@andres-pc:~/Documents/Programación 2 [Java]/003$ ls
task.class Task.java TaskList.class TaskList.java ToDoList.class ToDoList.java
andres@andres-pc:~/Documents/Programación 2 [Java]/003$ java ToDoList
para comenzar cree su To Do List
1. Crear lista vacia
2. Crear lista con una Task
3.
Ingres el nombre de su lista: Lista1
Para ello ingrese las opciones posteriores
1. Agregar una nueva tarea a la lista
2. Imprimir lista completa
1. Task rápida
2. Task normal
3. Task con notas
4. La opción que deseé utilizar: 1
Ingres el nombre:
Task1

```

3. Se realizó un programa en base a un caso de la vida real, que ejemplifique la técnica de *overloading* y se subió al repo de GitHub.

```

+ 003 x java Main
andres@andres-pc:~/Documents/Programación 2 (.Java)/ControlPagos$ ls
Main.class Main.java Pago.class Pago.java
andres@andres-pc:~/Documents/Programación 2 (.Java)/ControlPagos$ java Main
Ingresar su monto inicial: 400
1. Realizar un pago
2. Programar un pago futuro
3. Ver mis pagos
4. Revisar mis transacciones
5. Ver estado de cuenta
6. Salir
Elegir su opción: 1
Ingresar la cantidad:
400
Ingresar la descripción:
Pago!
Pago realizado exitosamente!
1. Realizar un pago

```

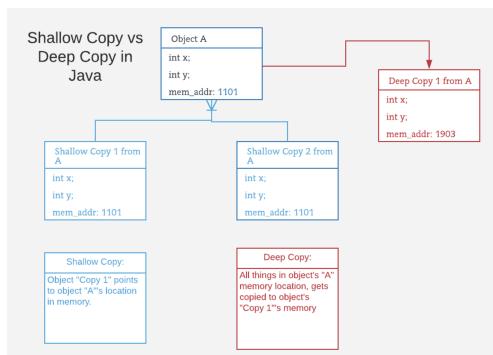
4. Se realizó 25 ejercicios de los conocimientos obtenidos sobre programación y Java y se subió al repo de GitHub.

```

+ 003 x java Programa
andres@andres-pc:~/Documents/Programación 2 (.Java)/002$ java Programa
seleccione el programa que desea utilizar:
! 
Ingresar el número 1: 1
Ingresar el número 2: 2
El resultado es: 3
seleccione el programa que desea utilizar:
! 
Ingresar el número 1: 2
Ingresar el número 2: 5
Los números son iguales: false
seleccione el programa que desea utilizar:
! 
Ingresar un número: 45
i*1=45
i*2=90
i*3=135
i*4=180

```

5. Se investigó la diferencia entre clonación y asignación de objetos en Java y se realizó un diagrama.



6. Cada integrante del grupo redactó un ensayo explicando los conceptos de *static* y *final* y sus diferencias.

Modificador de tipo *Estático*

El modificador *estático* se puede utilizar en cuatro diferentes escenarios: En variables, métodos, bloques de código y clases anidadas.

1. Variables:

Es una variable que pertenece a la clase y no a el objeto (instancia). Las variables de tipo estáticas son inicializadas una sola vez y esto es al inicio de la ejecución del programa. Estas variables son inicializadas antes que cualquier otra instancia de variables. Esta variable es una sola copia que todos los objetos la comparten. Esta no necesita un objeto para ser utilizada.

Sintaxis:

`Class.variable`

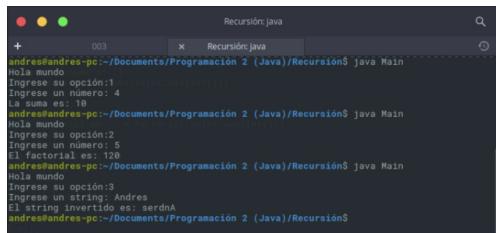
Ejemplo:

```

1 // Se crea una clase que contiene una variable de tipo estática ('static')
2 public class MyClass {
3     public static int myVariable = 0;
4 }
5 // Se inicializan dos instancias de la misma clase
6 MyClass instance1 = new MyClass();
7 MyClass instance2 = new MyClass();
8
9 MyClass.myVariable = 5; // Este cambio se ve reflejado en ambas instancias de la clase

```

7. Se realizaron los ejercicios de *recursión*, se subieron a MiU y al repo de GitHub.



```
+ 003 x Recursión.java
Hola mundo
Ingresé su opción:1
Ingresé un número: 4
La suma es: 10
Hola mundo
Ingresé su opción:2
Ingresé un número: 5
El factorial es: 120
Hola mundo
Ingresé su opción:3
Ingresé un string: Andres
El string invertido es: sredna
Andrés@andres-pc:~/Documents/Programación 2 (Java)/Recursión$
```

8. Se realizó el ejercicio 004 de MiU, se subió a MiU y al repo de GitHub.
9. Se realizó este documento PDF que lista los *Stories* en *done* de este *Sprint*.