Informe de Laboratorio 03
Tema: Arreglos de Objetos

Nota

Estudiante	Escuela	Asignatura
Julio Rubén Chura Acabana jchuraaca@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	F. de Programción 2 Semestre: I Código: 20230472

Laboratorio	Tema	Duración
03	Arreglos de Objetos	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 20 Septiembre 2023	Al 25 Septiembre 2023

1. Tarea

- Debe completar el código dado en la práctica de laboratorio
- Usando arreglos de objetos debe hacer la actividad 4 y 5 de la práctica 1
- Usted debe realizar varios commits y al término de la actividad deberá realizar un informe

2. Equipos, materiales y temas utilizados

- Sistema Operativo Windows
- VIM 9.0.
- OpenJDK 64-Bits 17.0.7.
- Git 2.39.2.
- Cuenta en GitHub con el correo institucional.
- Arreglos Estándar

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/JulioChura/fp2-23b.git`
- URL para el laboratorio 01 en el Repositorio GitHub.
- `https://github.com/JulioChura/fp2-23b/tree/main/fase01/lab03`

4. Actividades

4.1. Completar el código de DemoBatalla.java y Nave.java

Listing 1: Inicializando los archivos DemoBatalla.java y Nave.java

```
git config --global user.name "Julio Chura"
git config --global user.email "jchuraaca@unsa.edu.pe"
git clone https://github.com/JulioChura/fp2-23b.git
cd fp2-23b/fase01
mkdir lab03
cd lab03
vim DemoBatalla.java
vim Nave.java
```

Listing 2: Commit: Subiendo archivo DemoBatalla.java incompleto

```
git add DemoBatalla.java
git add Nave.java
git commit -m "Subiendo archivo DemoBatalla.java incompleto"
git push
```

Listing 3: Dándole forma y aumentando algunas líneas

```
vim DemoBatalla.java
```

- En el código base, se nos presenta varios métodos que debemos completar, pero nos damos cuenta que la forma en la que se imprimirá causará algo de desorden, por ello se añaden saltos de líneas y de paso se aprovecha en hacer un bosquejo del método que nos permitirá copiar los elementos de un arreglo pero de forma aleatoria.
- En las líneas 28, 30, 32, 34 se añaden los saltos de línea para tener un mejor seguimiento del código al momento de imprimirse. En las líneas 35 y 36 ya se puede observar que se está implementando el método çopiarNavesAleatoriamente(Naves[] a)"

```
28          System.out.println("Naves creadas:");
29          mostrarNaves(misNaves);
30          System.out.println();
31          mostrarPorNombre(misNaves);
32          System.out.println();
33          mostrarPorPuntos(misNaves);
34          System.out.println("Nave con mayor número de puntos: " + mostrarMayorPuntos(misNaves));
35          Nave[] navesAleatorias = copiarNavesAleatoriamente(misNaves);
36          mostrarNaves(navesAleatorias);
```

- En la línea 54 ya se ve la cabecera del método copiarNavesAleatoriamente(Naves[] a)

```
52      //Crear un método que devuelva un nuevo arreglo de objetos con todos los objetos previamente ingresados
53      //pero aleatoriamente desordenados
54      public static Nave[] copiarNavesAleatoriamente(Nave[] naves){
55
56      }
```

Listing 4: Commit: Aumentando saltos de linea para una mejor presentacion y aumentando unas cosas más

```
git add .
git commit -m "Aumentando saltos de linea para una mejor presentacion y aumentando unas
    cosas mas"
git push -u origin main
```

Listing 5: Completando el método mostrarNaves

```
vim BatallaDemo.java
vim Nave.java
```

- Para este método, se decidió recorrer todo el arreglo e ir imprimiendo, sin embargo al momento de compilar se obtuvo problemas porque en vez de imprimir los datos de cada nave, nos mostraba la dirección de memoria.

```
38      //Método para mostrar todas las naves
39      public static void mostrarNaves(Nave [] flota){
40          for(int i = 0; i < flota.length; i++){
41              System.out.println("Nave "+(i+1)+": "+flota[i]);
42          }
43      }
```

- El problema se debía a que en la clase Nave, no se escribió un método que devuelva los datos del objeto, por ello se implementa el método toString() permitiendo ahora si visualizar los datos que se han ingresado.

```
42      public String toString() {
43          return "Nombre: " + nombre + ", Fila: " + fila + ", Columna: " +
44      columna + ", Estado; "+ estado + ", Puntos: "+ puntos;
45      }
46  }
```

Listing 6: Compilando el programa pero para ahorrar pasos, se reduce la cantidad de elementos siendo ahora de 1.

```
javac Nave.java
javac DemoBatalla.java
java DemoBatalla
Nave 1
Nombre: arconte
Fila
3
Columna: 4
Estado: true
Puntos: 34
Naves creadas:
Nave 1: Nombre: arconte, Fila: 3, Columna: 4, Estado; true, Puntos: 34
Nave con mayor numero de puntos: null
```

Listing 7: Commit: "Metodo mostrarNaves esta terminado y modificamos el tamaño del arreglo para compilar mas rapido"

```
git add .
git commit -m "Metodo mostrarNaves esta terminado y modificamos el tamano del arreglo
    para compilar mas rapido""

git push -u origin main
```

Listing 8: Completando método mostrarPorNombre

```
vim DemoBatalla.java
```

- Para hacer la búsqueda, se decide aprovechar la propiedad que tiene el atributo Nombre de la clase Nave ya que es de tipo String, por lo que podremos usar el método equalsIgnoreCase(String str) y usaremos también el método toString() para ir imprimiendo las coincidencias.

```
44      //Método para mostrar todas las naves de un nombre que se pide por teclado
45      public static void mostrarPorNombre(Nave [] flota){
46          Scanner sc = new Scanner(System.in);
47          System.out.println("Ingrese el nombre de la nave que desea buscar");
48          String naveNombre = sc.nextLine();
49          Boolean encontrado = false;
50          for(int i = 0; i < flota.length; i++){
51              if(naveNombre.equalsIgnoreCase(flota[i].getNombre() ) ){
52                  System.out.println(flota[i].toString());
53                  encontrado = true;
54               }
55           }
```

Listing 9: Procedemos a compilar y ejecutar el programa para probar este método con una modificación en el tamaño del arreglo el cual se pondrá en 2.

```
javac Nave.java
javac DemoBatalla.java
java DemoBatalla
Nave 1
Nombre: alfa
Fila
34
Columna: 5
Estado: true
Puntos: 43
Nave 2
Nombre: beta
Fila
34
Columna: 7
Estado: true
Puntos: 87
Naves creadas:
Nave 1: Nombre: alfa, Fila: 34, Columna: 5, Estado; true, Puntos: 43
Nave 2: Nombre: beta, Fila: 34, Columna: 7, Estado; true, Puntos: 87

Ingrese el nombre de la nave que desea buscar
```

```
beta
Nombre: beta, Fila: 34, Columna: 7, Estado; true, Puntos: 87

Nave con mayor numero de puntos: null
```

Listing 10: Commit:"Metodo para mostrar todas las naves de un nombre que se pide por teclado"

```
git add BatallaDemo.java
git commit -m "Metodo para mostrar todas las naves de un nombre que se pide por teclado"
git push -u origin main
```

Listing 11: Método mostrarPorPuntos

```
vim DemoBatalla.java
```

- Del mismo modo, aprovecharemos que el atributo puntos es de tipo int para realizar un búsqueda en todo el arreglo, haremos uso de getPuntos() para ir recorriendo a lo largo del arreglo y cuando se encuentren coincidencias se imprimirán con el método toString
- En caso no encontrar coincidencias, se ha declarado un variable del tipo Boolean el cual se le asignó un valor de false y en caso no haber coincidencias, este boolean conservará su valor (pero si se encuentran coincidencias, su valor cambiará a true) y pasará al if ejecutándose el mensaje "No fue encontrado".

```
62      public static void mostrarPorPuntos(Nave [] flota){
63          Scanner sc = new Scanner(System.in);
64              System.out.println("Ingrese los puntos para hacer la búsqueda de resultados menores o igual ");
65              int puntos = sc.nextInt();
66              Boolean encontrado = false;
67              for(int i = 0; i < flota.length; i++) {
68                  if(puntos >= flota[i].getPuntos()){
69                      System.out.println(flota[i].toString());
70                      encontrado = true;
71                  }
72              }
73              if(encontrado==false){
74                  System.out.println("No fue encontrado");
75              }
76              System.out.println();
77      }
```

Listing 12: Compilando y ejecutando DemoBatalla.java con este último método implemenatdo

```
javac Nave.java
javac DemoBatalla.java
java DemoBatalla
Nave 1
Nombre: alfa
Fila
43
Columna: 5
Estado: true
Puntos: 543
Nave 2
Nombre: beta
Fila
65
```

```
Columna: 45
Estado: true
Puntos: 342
Naves creadas:
Nave 1: Nombre: alfa, Fila: 43, Columna: 5, Estado; true, Puntos: 543
Nave 2: Nombre: beta, Fila: 65, Columna: 45, Estado; true, Puntos: 342

Ingrese el nombre de la nave que desea buscar
beta
Nombre: beta, Fila: 65, Columna: 45, Estado; true, Puntos: 342

Ingrese los puntos para hacer la busqueda de resultados menores o igual
400
Nombre: beta, Fila: 65, Columna: 45, Estado; true, Puntos: 342

Nave con mayor numero de puntos: null
```

Listing 13: Commit: Corrigiendo commit, el metodo culminado es mostrarPorPuntos

```
git add .
git commit -m "Corrigiendo commit, el metodo culminado es mostrarPorPuntos"
git push -u origin main
```

Listing 14: Método mostrarMayorPuntos

```
vim DemoBatalla.java
```

- Del mismo modo, aprovecharemos que el atributo puntos es de tipo int para realizar un búsqueda en todo el arreglo, haremos uso de getPuntos() para ir recorriendo a lo largo del arreglo y cuando se encuentren coincidencias se imprimirán con el método toString
- En caso no encontrar coincidencias, se ha declarado un variable del tipo Boolean el cual se le asignó un valor de false y en caso no haber coincidencias, este boolean conservará su valor (pero si se encuentran coincidencias, su valor cambiará a true) y pasará al if ejecutándose el mensaje "No fue encontrado"

```
78      //Método que devuelve la Nave con mayor número de Puntos
79      public static Nave mostrarMayorPuntos(Nave [] flota){
80          int puntosMaximos = flota[0].getPuntos();
81          int indice = 0;
82          for(int i = 1; i < flota.length; i++ ){
83              if(puntosMaximos < flota[i].getPuntos()){
84                  puntosMaximos = flota[i].getPuntos();
85                  indice = i;
86              }
87          }
88          return flota[indice];
```

Listing 15: Compilando y ejecutando el programa con 2 elementos en el arreglo

```
javac Nave.java
javac DemoBatalla.java
java DemoBatalla
```

```
Nave 1
Nombre: alfa
Fila
23
Columna: 54
Estado: true
Puntos: 34
Nave 2
Nombre: beta
Fila
32
Columna: 5
Estado: true
Puntos: 43
Naves creadas:
Nave 1: Nombre: alfa, Fila: 23, Columna: 54, Estado; true, Puntos: 34
Nave 2: Nombre: beta, Fila: 32, Columna: 5, Estado; true, Puntos: 43

Ingrese el nombre de la nave que desea buscar
alfa
Nombre: alfa, Fila: 23, Columna: 54, Estado; true, Puntos: 34

Ingrese los puntos para hacer la busqueda de resultados menores o igual
56
Nombre: alfa, Fila: 23, Columna: 54, Estado; true, Puntos: 34
Nombre: beta, Fila: 32, Columna: 5, Estado; true, Puntos: 43
Nave con mayor numero de puntos: Nave 2: Nombre: beta, Fila: 32, Columna: 5, Estado;
    true,
```

Listing 16: Commit: "Metodo mostrarMayorPuntos culmiando"

```
git add .
git commit -m "Metodo mostrarMayorPuntos culmiando"
git push -u origin main
```

Listing 17: Implementando el método copiarNavesAleatoriamente y cambiando la cantidad de elementos del arreglo, pasando de 2 a 10

```
vim DemoBatalla.java
```

- Primero se copian los elementos de un arreglo a otro, para ello se usa el método System.arrayCopy, luego se recurre a la ayuda de un algoritmo de ordenación el cual se llama Fisher-Yates shuffle que consiste en que la variable de control recorra el arreglo empezando desde la última posición, dentro del bucle se genera un número aleatorio de 0 hasta la variable de control sumado con uno, ese aleatorio representa el indice del elemento del arreglo a intercambiar con la posición en la que se encuentra la variable de control. En cada iteración la variable de control va ir disminuyendo el índice, esto con el fin de no generar números aleatorios en los espacios que ya fueron mezclados los elementos. Con el método ya realizado, lo implementamos en el main, luego cambiamos el tamaño del arreglo misNaves pasando de 2 a 10

```
92      public static Nave[] copiarNavesAleatoriamente(Nave[] naves){
93          Nave[] arrayCopia = new Nave[naves.length];
94          System.arraycopy(naves,0,arrayCopia,0,naves.length);
95          Random random = new Random();
96          for (int i = arrayCopia.length - 1; i > 0; i--) {
97              int indiceAleatorio = random.nextInt(i + 1);
98              Nave temporal = arrayCopia[indiceAleatorio];
99              arrayCopia[indiceAleatorio] = arrayCopia[i];
100             arrayCopia[i] = temporal;
101         }
102         return arrayCopia;
103     }
```

Listing 18: Compilando y ejecutando el programa con 10 elementos en el arreglo

```
javac Nave.java
javac DemoBatalla.java
java DemoBatalla
Nave 1
Nombre: Alfa
Fila
1
Columna: 2
Estado: true
Puntos: 23
Nave 2
Nombre: Beta
Fila
4
Columna: 5
Estado: true
Puntos: 34
Nave 3
Nombre: Spider
Fila
6
Columna: 4
Estado: false
Puntos: 56
Nave 4
Nombre: Caza
Fila
4
Columna: 8
Estado: true
Puntos: 57
Nave 5
Nombre: Alfa
Fila
4
Columna: 8
Estado: true
Puntos: 65
Nave 6
Nombre: Alcon
Fila
5
```

```
Columna: 9
Estado: true
Puntos: 90
Nave 7
Nombre: Arconte
Fila
3
Columna: 6
Estado: false
Puntos: 68
Nave 8
Nombre: Venganza
Fila
4
Columna: 8
Estado: true
Puntos: 79
Nave 9
Nombre: Destroyer
Fila
4
Columna: 7
Estado: true
Puntos: 34
Nave 10
Nombre: Huascar
Fila
3
Columna: 9
Estado: true
Puntos: 91
Naves creadas:
Nave 1: Nombre: Alfa, Fila: 1, Columna: 2, Estado; true, Puntos: 23
Nave 2: Nombre: Beta, Fila: 4, Columna: 5, Estado; true, Puntos: 34
Nave 3: Nombre: Spider, Fila: 6, Columna: 4, Estado; false, Puntos: 56
Nave 4: Nombre: Caza, Fila: 4, Columna: 8, Estado; true, Puntos: 57
Nave 5: Nombre: Alfa, Fila: 4, Columna: 8, Estado; true, Puntos: 65
Nave 6: Nombre: Alcon, Fila: 5, Columna: 9, Estado; true, Puntos: 90
Nave 7: Nombre: Arconte, Fila: 3, Columna: 6, Estado; false, Puntos: 68
Nave 8: Nombre: Venganza, Fila: 4, Columna: 8, Estado; true, Puntos: 79
Nave 9: Nombre: Destroyer, Fila: 4, Columna: 7, Estado; true, Puntos: 34
Nave 10: Nombre: Huascar, Fila: 3, Columna: 9, Estado; true, Puntos: 91

Ingrese el nombre de la nave que desea buscar
Alfa
Nombre: Alfa, Fila: 1, Columna: 2, Estado; true, Puntos: 23
Nombre: Alfa, Fila: 4, Columna: 8, Estado; true, Puntos: 65

Ingrese los puntos para hacer la busqueda de resultados menores o igual
70
Nombre: Alfa, Fila: 1, Columna: 2, Estado; true, Puntos: 23
Nombre: Beta, Fila: 4, Columna: 5, Estado; true, Puntos: 34
Nombre: Spider, Fila: 6, Columna: 4, Estado; false, Puntos: 56
Nombre: Caza, Fila: 4, Columna: 8, Estado; true, Puntos: 57
Nombre: Alfa, Fila: 4, Columna: 8, Estado; true, Puntos: 65
Nombre: Arconte, Fila: 3, Columna: 6, Estado; false, Puntos: 68
```

```
Nombre: Destroyer, Fila: 4, Columna: 7, Estado; true, Puntos: 34

Nave con mayor numero de puntos: Nombre: Huascar, Fila: 3, Columna: 9, Estado; true,
    Puntos: 91
Nave 1: Nombre: Alcon, Fila: 5, Columna: 9, Estado; true, Puntos: 90
Nave 2: Nombre: Alfa, Fila: 4, Columna: 8, Estado; true, Puntos: 65
Nave 3: Nombre: Venganza, Fila: 4, Columna: 8, Estado; true, Puntos: 79
Nave 4: Nombre: Destroyer, Fila: 4, Columna: 7, Estado; true, Puntos: 34
Nave 5: Nombre: Beta, Fila: 4, Columna: 5, Estado; true, Puntos: 34
Nave 6: Nombre: Spider, Fila: 6, Columna: 4, Estado; false, Puntos: 56
Nave 7: Nombre: Caza, Fila: 4, Columna: 8, Estado; true, Puntos: 57
Nave 8: Nombre: Alfa, Fila: 1, Columna: 2, Estado; true, Puntos: 23
Nave 9: Nombre: Huascar, Fila: 3, Columna: 9, Estado; true, Puntos: 91
Nave 10: Nombre: Arconte, Fila: 3, Columna: 6, Estado; false, Puntos: 68
```

Listing 19: Commit: Codigo DemoBatalla.java terminado

```
git add .
git commit -m " Codigo DemoBatalla.java terminado "
git push -u origin main
```

4.2. Resolver la actividad 4 y 5 de la práctica01 de laboratorio pero usando arreglos de Objetos

Listing 20: Se está copiando el archivo VideoJuego.java para resolver la actividad 4 y 5. Tambiénse está creando la clase Soldier para poder los objetos

```
cd ..
cd lab01
Copy-Item -Path "VideoJuego.java" -Destination "..\lab03"
cd ..
cd lab03
vim VideoJuego.java
vim Soldier.java
```

Listing 21: Se resuelve el ejercicio 4 de la praactica de laboratorio 01 usando arreglo con objetos.

- Se crea la clase Soldier.java la cual contiene los atrbutos y métodos que son empleados en la clase principal. Se crea un arreglo de Soldier y para llenar dicho arreglo con los nombres y puntos de vida se usa un bucle for, en cada itereación se irá preguntando, luego se almacenarán en las variables name y lifePoint para luego mandarlos a guardar en el arreglo usando los set

```java
// Laboratorio Nro 1 - Ejercicio 4
// Autor: Julio
// Tiempo: 30 minutos
// No hubo colaboradores
import java.util.*;
public class VideoJuego{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        Soldier soldier = new Soldier();
        Soldier[] army = new Soldier[5];
        for(int i = 0; i < army.length; i++){
            System.out.print("Enter the name of soldier "+(i+1)+": " );
            String name = sc.next();
            System.out.print("Enter the points life "+(i+1)+": " );
            int points = sc.nextInt();
            System.out.println();
            army[i] = new Soldier();
            army[i].setName(name);
            army[i].setLifePoints(points);
        }
        System.out.println();
        printArray(army);
    }
    public static void printArray(Soldier[] a){
        for(int i = 0; i < a.length; i++){
            System.out.println(a[i].toString() );
        }
    }
}
```

Listing 22: Commit: Se realizan 2 commit para subir tanto Soldier.java y VideoJuego.java

```
git add VideoJuego.java
 git commit -m "Ejercicio04 de lab01 usando arreglos de objetos "
 git push -u origin main
 git add Soldier.java
 git commit -m "Subiendo la clase Soldier.java "
 gut push .u origin main
```

Listing 23: Se realiza la actividad 5 del laboratorio01 la cual consiste en generar dos arreglos de tamaño aleatorio y gana el que tenga más soldados

```
vim VideoJuego.java
```

- Se crean dos métodos, uno es para imprimir los arreglos y el otro para generar un ejército de cantidad n que está entre 1 y 5. Para determinar el ganador, se emplean condiconales donde se evalúan los tamaños de los 2 arreglos arreglos de tipo Soldier

```java
// Laboratorio Nro 3 - Ejercicio 5
// Autor: Julio
// Tiempo: 20 minutos
// No hubo colaboradores
import java.util.*;
public class VideoJuego{
    public static void main(String[] args){
        Soldier[] army1 = generateArmy();
        Soldier[] army2 = generateArmy();

        printArray(army1);
        System.out.println();
        printArray(army2);
        if (army1.length < army2.length ) {
            System.out.println("The winner is the army 2");
        } else if ( army1.length > army2.length) {
            System.out.println("The winner is the army 1");
        } else {
            System.out.println("IT WAS A TIE");
        }
    }
    public static void printArray(Soldier[] a){
        for(int i = 0; i < a.length; i++){
            System.out.println(a[i].toString() );
        }
    }
    public static Soldier[] generateArmy(){
        Random random = new Random();
        int amount =  random.nextInt(5)+1;
        Soldier[] myArmy = new Soldier[amount];
        for(int i = 0; i < myArmy.length; i++){
            String theName = "Soldier"+i;
            myArmy[i] = new Soldier();
            myArmy[i].setName(theName);
        }
        return myArmy;
    }

}
```

Listing 24: Compilando y ejecutando VideoJuego.java

```
javac VideoJuego.java
javac Soldier.java
java VideoJuego
Soldier{name=Soldier0, lifePoints=0}
Soldier{name=Soldier1, lifePoints=0}
Soldier{name=Soldier2, lifePoints=0}

Soldier{name=Soldier0, lifePoints=0}
Soldier{name=Soldier1, lifePoints=0}
Soldier{name=Soldier2, lifePoints=0}
Soldier{name=Soldier3, lifePoints=0}
The winner is the army 2
```

Listing 25: Commit: Ejercicio 05 de la practica 01 de lab culminada

```
git add VideoJuego.java
git commit -m "Ejercicio 05 de la practica 01 de lab culminada"
git push -u origin main
```

4.3. Estructura de laboratorio 03

- El contenido que se entrega en este laboratorio es el siguiente:

```
lab03/
|---DemoBatalla.java
|---Nave.java
|---Soldier.java
|---VideoJuego.java
|
|---latex
   |---- programacion_lab03_rescobedoq_v1.0.pdf
   |-----programacion_lab03_rescobedoq_v1.0.tex
   |
   |-----img
   |      1.jpg
   |      10.jpg
   |      2.jpg
   |      3.jpg
   |      4.jpg
   |      5.jpg
   |      6.jpg
   |      7.jpg
   |      8.jpg
   |      9.jpg
   |      logo_abet.png
   |      logo_episunsa.png
   |      logo_unsa.jpg
   |
   |-----src
```

5. Rúbricas

5.1. Entregable Informe

Tabla 1: Tipo de Informe

Informe	
Latex	El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y facil de leer.

5.2. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumplio con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos lo items.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: Niveles de desempeño

	Nivel			
Puntos	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 3: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	2	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
6. Fechas	Las fechas de modificación del código fuente estan dentro de los plazos de fecha de entrega establecidos.	2	X	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	X	2	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	2	
Total		20		16	

6. Referencias

- https://drive.google.com/file/d/1gF5iR4EpCOfMuwdQCGPfbErUFeA3cp_U/view?usp=drive_link