

Informe de Laboratorio 07

Tema: Combinando Arreglos Estándar y ArrayList

Nota

Estudiante	Escuela	Asignatura
Ryan Fabian Valdivia Segovia rvaldiviase@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Fundamentos de la programación 2 Semestre: II Código: 1701213

Laboratorio	Tema	Duración
07	Combinando Arreglos Estándar y ArrayList	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 18 de Octubre 2023	Al 23 de Octubre 2023

1. Tarea

1.1. Videojuego

- Cree un Proyecto llamado Laboratorio7
- Usted deberá crear las dos clases Soldado.java y VideoJuego4.java. Puede reutilizar lo desarrollado en Laboratorios anteriores.
- Del Soldado nos importa el nombre, puntos de vida, fila y columna (posición en el tablero).
- El juego se desarrollará en el mismo tablero de los laboratorios anteriores. Para el tablero utilizar la estructura de datos más adecuada.
- Tendrá 2 Ejércitos (utilizar la estructura de datos más adecuada). Inicializar el tablero con n soldados aleatorios entre 1 y 10 para cada Ejército. Cada soldado tendrá un nombre autogenerado: Soldado0X1, Soldado1X1, etc., un valor de puntos de vida autogenerado aleatoriamente [1..5], la fila y columna también autogenerados aleatoriamente (no puede haber 2 soldados en el mismo cuadrado). Se debe mostrar el tablero con todos los soldados creados y sus puntos de vida. Además de los datos del Soldado con mayor vida de cada ejército, el promedio de puntos de vida de todos los soldados creados por ejército, los datos de todos los soldados por ejército en el orden que fueron creados y un ranking de poder de todos los soldados creados por ejército (del que tiene más nivel de vida al que tiene menos) usando diferentes algoritmos de ordenamiento. Finalmente, que muestre qué ejército ganará la batalla (indicar la métrica usada para decidir al ganador de la batalla). Hacer el programa iterativo.

2. Equipos, materiales y temas utilizados

- Sistema Operativo Windows 11 Home Single Language 64 bits 22H2.2283
- VIM 9.0.
- Visual Studio Code 64 bits 1.82.2
- OpenJDK 64-Bits 11.0.16.1
- Git 2.41.0.windows.1
- Cuenta en GitHub con el correo institucional.

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- <https://github.com/RyanValdivia/fp2-23b.git>
- URL para el laboratorio 07 en el Repositorio GitHub.
- <https://github.com/RyanValdivia/fp2-23b/tree/main/fase02/lab07>

4. Actividades

4.1. Actividad 1

- En primer lugar, realicé un commit conteniendo el código de la clase Soldado.java, requerido para la clase principal

Listing 1: Obteniendo la clase Soldado

```
$ git log lab07
commit 9228a8c7c441745ab690009dee69fb3e8bdae06b
Author: RYAN VALDIVIA <rvaldiviase@unsa.edu.pe>
Date: Sat Oct 21 12:25:58 2023 -0500
    Copiando la clase Soldado y reciclando algo de código de laboratorios anteriores
```

- Conteniendo el siguiente código en la clase Soldado

Listing 2: Clase Soldado

```
6 public class Soldado {
7     private String nombre;
8     private int vida;
9     private int fila;
10    private int columna;
11    private String bandera;
12
13    public void setNombre(String s) {
14        this.nombre = s;
15    }
16 }
```

```
17 public void setVida(int n) {
18     this.vida = n;
19 }
20
21 public void setFila(int n) {
22     this.fila = n;
23 }
24
25 public void setColumna(int n) {
26     this.columna = n;
27 }
28
29 public void setBandera(String s) {
30     this.bandera = s;
31 }
32
33 public String getNombre() {
34     return nombre;
35 }
36
37 public int getVida() {
38     return vida;
39 }
40
41 public int getFila() {
42     return fila;
43 }
44
45 public int getColumna() {
46     return columna;
47 }
48
49 public String getBandera() {
50     return bandera;
51 }
52 }
```

- Cabe aclarar, que añadí un atributo de 'bandera', será utilizado más tarde.
- Además de reutilizar el sistema para obtener las coordenadas de los soldados de laboratorios anteriores.

Listing 3: Código parcial

```
53 public static void main(String[] args) {
54     Scanner sc = new Scanner(System.in);
55     Soldado[][] tablero = new Soldado[10][10];
56     int ej1 = (int) (Math.random() * 10 + 1);
57     int ej2 = (int) (Math.random() * 10 + 1);
58     int[] filas1 = numerosRandom(ej1);
59     int[] columnas1 = numerosRandom(ej1);
60     int[] filas2;
61     int[] columnas2;
62     do {
63         filas2 = numerosRandom(ej2);
```

```
64         columnas2 = numerosRandom(ej2);
65     } while (!diffCoordenadas(filas1, filas2, columnas1, columnas2));
66 }
67 public static int[] numerosRandom(int q) {
68     int[] nums = new int[q];
69     for (int i = 0; i < nums.length; i++) {
70         nums[i] = nums.length;
71     }
72     for (int i = 0; i < q; i++) {
73         int n;
74         do {
75             n = (int) (Math.random() * 10);
76         } while (estaEnArreglo(nums, n, i));
77         nums[i] = n;
78     }
79     return nums;
80 }
81
82 public static boolean estaEnArreglo(int[] arreglo, int num, int indice) {
83     for (int i = 0; i < indice; i++) {
84         if (arreglo[i] == num) {
85             return true;
86         }
87     }
88     return false;
89 }
90
91 public static boolean diffCoordenadas(int[] filas1, int[] filas2, int[] columnas1, int[]
92     columnas2) {
93     if (filas1.length > filas2.length) {
94         for (int i = 0; i < filas2.length; i++) {
95             if (filas1[i] == filas2[i] && columnas1[i] == columnas2[i]) {
96                 return false;
97             }
98         }
99     } else {
100         for (int i = 0; i < filas1.length; i++) {
101             if (filas1[i] == filas2[i] && columnas1[i] == columnas2[i]) {
102                 return false;
103             }
104         }
105     }
106     return true;
107 }
```

- Donde, ya que me daban a elegir la estructura de datos más adecuada para el tablero, utilicé un arreglo bidimensional porque se me hacía más sencillo al momento de inicializar e instanciar el tablero.
- Posteriormente, creé e inicialicé mis dos ejércitos, para esto, utilicé arreglos simples, para mayor conveniencia. Utilizando un método para crear un arreglo en base a las coordenadas ya creadas antes.

Listing 4: Creando ejércitos

```
108 Soldado[] ejercito1 = crearArreglo(filas1, columnas1, 1);
109 Soldado[] ejercito2 = crearArreglo(filas2, columnas2, 2);
110
111 public static Soldado[] crearArreglo(int[] x, int[] y, int nro) {
112     int len = x.length;
113     Soldado[] army = new Soldado[len];
114     for (int i = 0; i < len; i++) {
115         int v = (int) (Math.random() * 5 + 1);
116         army[i] = new Soldado();
117         army[i].setNombre("Soldado" + i + "X" + nro);
118         army[i].setVida(v);
119         army[i].setFila(x[i]);
120         army[i].setColumna(y[i]);
121         if (nro == 1) {
122             army[i].setBandera("#####");
123         } else {
124             army[i].setBandera("*****");
125         }
126     }
127     return army;
128 }
```

- Entonces, este método recibe las filas y columnas (coordenadas) y empieza a inicializar los soldados de ambos ejércitos, además de darle una bandera a cada soldado (esto con el propósito de diferenciarlos al momento de imprimir el tablero).
- Además, añadí un método para 'desplegar' cada ejército en el tablero, en base a los arreglos con cada ejército.

Listing 5: Desplegando los ejércitos

```
129 desplegarEjercito(tablero, ejercito1);
130 desplegarEjercito(tablero, ejercito2);
131
132 public static void desplegarEjercito(Soldado[][] table, Soldado[] ej) {
133     for (int i = 0; i < ej.length; i++) {
134         table[ej[i].getFila()][ej[i].getColumna()] = ej[i];
135     }
136 }
```

- Este método toma el arreglo de Soldados y asigna la referencia de cada uno de los Soldados al tablero, en base a los atributos de coordenadas de cada Soldado.

Listing 6: Inicializar la lista

```
137 public static void inicializarLista(ArrayList<ArrayList<Soldado>> army) {
138     for (int i = 0; i < 10; i++) {
139         army.add(new ArrayList<>());
140     }
141     for (int i = 0; i < 10; i++) {
142         for (int j = 0; j < 10; j++) {
143             army.get(i).add(new Soldado());
144             army.get(i).get(j).setNombre(" ");
145         }
146     }
147 }
```

```
146     }  
147  
148 }
```

- Ahora solo falta inicializar el tablero, ya que por defecto, inician todas las entradas en 'null'.

Listing 7: Inicializando tablero

```
149     inicializarEjercito(tablero);  
150  
151     public static void inicializarEjercito(Soldado[] [] table) {  
152         for (int i = 0; i < table.length; i++) {  
153             for (int j = 0; j < table[i].length; j++) {  
154                 table[i][j] = new Soldado();  
155                 table[i][j].setNombre(" ");  
156                 table[i][j].setBandera(" ");  
157             }  
158         }  
159     }
```

- Este método inicializa todas las entradas del arreglo bidimensional, y les asigna atributos de nombre y bandera vacíos, esto con el propósito de facilitar la tarea de imprimir el tablero.
- Entonces, este método se llama antes de desplegar los ejércitos.
- A continuación, solo quedaba mostrar el tablero de juego, para lo cual reutilicé código de los laboratorios anteriores, mejorándolo para que sea mucho más bonito y funcional, mostrando información completa de los soldados, como sus puntos de vida, nombre y una bandera única por ejércitos, para diferenciar unos de los otros.

Listing 8: Mostrar el tablero

```
160     public static void mostrarTablero(Soldado[] [] tb) {  
161         String vacio = " ";  
162         System.out.println(crearTecho());  
163         for (int i = 0; i < tb.length; i++) {  
164             System.out.println(separadorSup());  
165             for (int j = 0; j < tb[i].length; j++) {  
166                 if (j == tb[i].length - 1) {  
167                     System.out.print("| " + tb[i][j].getBandera() + " |\n");  
168                 } else {  
169                     System.out.print("| " + tb[i][j].getBandera() + " ");  
170                 }  
171             }  
172             for (int j = 0; j < tb[i].length; j++) {  
173                 if (j == tb[i].length - 1) {  
174                     System.out.print("| " + tb[i][j].getNombre() + " |\n");  
175                 } else {  
176                     System.out.print("| " + tb[i][j].getNombre() + " ");  
177                 }  
178             }  
179             for (int j = 0; j < tb[i].length; j++) {  
180                 if (tb[i][j].getVida() != 0) {  
181                     if (j == tb[i].length - 1) {
```

```
182         System.out.print("| " + tb[i][j].getVida() + " HP" + " |\n");
183     } else {
184         System.out.print("| " + tb[i][j].getVida() + " HP" + " ");
185     }
186 } else {
187     if (j == tb[i].length - 1) {
188         System.out.print("| " + vacio + " |\n");
189     } else {
190         System.out.print("| " + vacio + " ");
191     }
192 }
193 }
194 System.out.println(separadorInf());
195 }
196 System.out.println();
197 }
198
199 public static String crearTecho() {
200     String franky = "";
201     for (int i = 0; i < 131; i++) {
202         franky += "_";
203     }
204     return franky;
205 }
206
207 public static String separadorInf() {
208     String franky = "";
209     for (int i = 0; i < 131; i++) {
210         if (i % 13 == 0) {
211             System.out.print("|");
212         } else {
213             System.out.print("_");
214         }
215     }
216     return franky;
217 }
218
219 public static String separadorSup() {
220     String franky = "";
221     for (int i = 0; i < 131; i++) {
222         if (i % 13 == 0) {
223             System.out.print("|");
224         } else {
225             System.out.print(" ");
226         }
227     }
228     return franky;
229 }
```

- Es así que el código realiza varios recorridos por fila, imprimiendo primero la bandera del soldado (dependiendo de a qué ejército pertenece), su nombre y su nivel de vida (siendo HP, Health Points).
- Imprimiendo lo siguiente al momento de ejecutar el código:

				***** Soldado0X2 2 HP	##### Soldado5X1 1 HP				
##### Soldado2X1 5 HP				***** Soldado4X2 4 HP					
								##### Soldado0X1 3 HP	
							##### Soldado6X1 5 HP		***** Soldado2X2 1 HP
			##### Soldado3X1 4 HP		***** Soldado3X2 3 HP				
	***** Soldado5X2 3 HP								##### Soldado4X1 4 HP
			***** Soldado1X2 1 HP						
	##### Soldado7X1 2 HP								
				##### Soldado1X1 3 HP					
		##### Soldado8X1 4 HP							

- Una vez terminado el tablero, pasé a trabajar el resto de requerimientos para el programa.
- Para mostrar el soldado con mayor vida del ejército, solo necesitaba recorrer el arreglo una vez para obtener el índice del objeto con mayor vida y mostrar sus datos, algo mucho más sencillo.

Listing 9: Soldado con mayor vida

```

230 public static void soldadoMayorVida(Soldado[] army, int ej) {
231     int max = 0;
232     for (int i = 0; i < army.length; i++) {
233         if (army[i].getVida() > army[max].getVida()) {
234             max = i;
235         }
236     }
237     System.out.println("El soldado con mayor vida del ejercito " + ej + " es: ");
238     mostrarSoldado(army, max);
239     System.out.println();
240 }

```

- Una vez encontrado el índice, solo debía mostrar el soldado de esa posición en el arreglo, para lo cuál, viendo que ese proceso sería algo que repita con frecuencia, decidí elaborar un método para mostrar un soldado.

Listing 10: Mostrar soldado

```

241 public static void mostrarSoldado(Soldado[] army, int i) {
242     String columna;

```



```
243     System.out.println("Nombre: " + army[i].getNombre());
244     System.out.println("Vida: " + army[i].getVida() + " HP");
245     switch (army[i].getColumna() + 1) {
246         case 1:
247             columna = "A";
248             break;
249         case 2:
250             columna = "B";
251             break;
252         case 3:
253             columna = "C";
254             break;
255         case 4:
256             columna = "D";
257             break;
258         case 5:
259             columna = "E";
260             break;
261         case 6:
262             columna = "F";
263             break;
264         case 7:
265             columna = "G";
266             break;
267         case 8:
268             columna = "H";
269             break;
270         case 9:
271             columna = "I";
272             break;
273         case 10:
274             columna = "J";
275             break;
276         default:
277             columna = "K";
278             break;
279     }
280     System.out.println("Posicion: " + (army[i].getFila() + 1) + "-" + columna);
281     System.out.println();
282 }
```

- Después de ello, quedaba mostrar el total de vida de cada ejército y su vida promedio, para lo cual creé un método que imprimiera ambas cosas, recorriendo el arreglo una sola vez.

Listing 11: Vida total y promedio

```
284     public static void vidaPromedio(Soldado[] army, int ej) {
285         int total = 0;
286         for (Soldado s : army) {
287             total += s.getVida();
288         }
289         System.out.println("La vida total del ejercito " + ej + " es: " + total);
290         System.out.println("La vida promedio del ejercito " + ej + " es: " + total / (1.0 *
291             army.length));
292     }
```

292

}

- Esto imprimía lo siguiente (Siguiendo con la ejecución de la captura anterior):

```
El soldado con mayor vida del ejercito 1 es:  
Nombre: Soldado2X1  
Vida: 5 HP  
Posición: 2-A  
  
El soldado con mayor vida del ejercito 2 es:  
Nombre: Soldado4X2  
Vida: 4 HP  
Posición: 2-D  
  
La vida total del ejercito 1 es: 31  
La vida promedio del ejercito 1 es: 3.4444444444444446  
  
La vida total del ejercito 2 es: 14  
La vida promedio del ejercito 2 es: 2.3333333333333335
```

- Una vez terminado eso, seguía mostrar todos los soldados según el orden de creación, para eso, solo recorrí todo el arreglo y mostré cada soldado (para eso era el método anterior).

Listing 12: Mostrar ejército

```
293 public static void mostrarEjercito(Soldado[] army, int ej) {  
294     System.out.println("Ejercito " + ej);  
295     System.out.println(army[0].getBandera());  
296     for (int i = 0; i < army.length; i++) {  
297         mostrarSoldado(army, i);  
298     }  
299     System.out.println();  
300 }
```

- Esto imprime lo siguiente:

```

Ejercito 1
#####
Nombre: Soldado0X1
Vida: 3 HP
Posición: 3-I

Nombre: Soldado1X1
Vida: 3 HP
Posición: 9-E

Nombre: Soldado2X1
Vida: 5 HP
Posición: 2-A

Nombre: Soldado3X1
Vida: 4 HP
Posición: 5-D

Nombre: Soldado4X1
Vida: 4 HP
Posición: 6-J

Nombre: Soldado5X1
Vida: 1 HP
Posición: 1-F

Nombre: Soldado6X1
Vida: 5 HP
Posición: 4-H

Nombre: Soldado7X1
Vida: 2 HP
Posición: 8-B

Nombre: Soldado8X1
Vida: 4 HP
Posición: 10-C

Ejercito 2
*****
Nombre: Soldado0X2
Vida: 2 HP
Posición: 1-E

Nombre: Soldado1X2
Vida: 1 HP
Posición: 7-C

Nombre: Soldado2X2
Vida: 1 HP
Posición: 4-J

Nombre: Soldado3X2
Vida: 3 HP
Posición: 5-F

Nombre: Soldado4X2
Vida: 4 HP
Posición: 2-D

Nombre: Soldado5X2
Vida: 3 HP
Posición: 6-B

```

- Una vez terminado con esto, continúe con realizar el ranking de Soldados según el nivel de vida (de mayor a menor), para esto, reciclé código de laboratorios anteriores para implementar los algoritmos de ordenamiento necesarios. Además de implementar un menú para escoger el algoritmo a usar.

Listing 13: Algoritmos de ordenamiento y ranking

```

301 System.out.println("Bajo que algoritmo de ordenamiento le gustaria ordenar su ejercito?");
302 System.out.println("1. Ordenamiento por burbuja");
303 System.out.println("2. Ordenamiento por insercion");
304 switch (sc.nextInt()) {
305     case 1:
306         ordenamientoBurbuja(ejercito1);
307         ordenamientoBurbuja(ejercito2);
308         break;
309     case 2:
310         ordenamientoInsercion(ejercito1);
311         ordenamientoInsercion(ejercito2);
312         break;
313     default:
314 }
315 System.out.println();
316 System.out.println("Ranking de ambos ejercitos del soldado con mayor a menor vida:
    \n");
317 mostrarEjercito(ejercito1, 1);
318 mostrarEjercito(ejercito2, 2);
319 public static void ordenamientoInsercion(Soldado[] army) {

```

```

320     for (int i = 1; i < army.length; i++) {
321         Soldado valor = army[i];
322         int j = i;
323         for (j = i; 0 < j && army[j - 1].getVida() < valor.getVida(); j--) {
324             army[j] = army[j - 1];
325         }
326         army[j] = valor;
327     }
328 }
329
330 public static void ordenamientoBurbuja(Soldado[] army) {
331     for (int i = 0; i < army.length; i++) {
332         for (int j = 0; j < army.length - 1; j++) {
333             if (army[j].getVida() < army[j + 1].getVida()) {
334                 intercambiar(army, j, j + 1);
335             }
336         }
337     }
338 }
339
340 public static void intercambiar(Soldado[] flota, int i, int j) {
341     Soldado temp;
342     temp = flota[i];
343     flota[i] = flota[j];
344     flota[j] = temp;
345 }

```

- Entonces, se ordenan los ejércitos según el algoritmo seleccionado y posteriormente, se muestran en consola.
- Mostrando lo siguiente al momento de ejecutar.

```

Bajo que algoritmo de ordenamiento le gustaria ordenar su ejercito?
1. Ordenamiento por burbuja
2. Ordenamiento por insercion
1

Ranking de ambos ejercitos del soldado con mayor a menor vida:

Ejercito 1
#####
Nombre: Soldado2X1
Vida: 5 HP
Posición: 2-A

Nombre: Soldado6X1
Vida: 5 HP
Posición: 4-H

Nombre: Soldado3X1
Vida: 4 HP
Posición: 5-D

Nombre: Soldado4X1
Vida: 4 HP
Posición: 6-J

Nombre: Soldado8X1
Vida: 4 HP
Posición: 10-C

Nombre: Soldado0X1
Vida: 3 HP
Posición: 3-I

Nombre: Soldado1X1
Vida: 3 HP
Posición: 9-E

Nombre: Soldado7X1
Vida: 2 HP
Posición: 8-B

Nombre: Soldado5X1
Vida: 1 HP
Posición: 1-F

Ejercito 2
*****
Nombre: Soldado4X2
Vida: 4 HP
Posición: 2-D

Nombre: Soldado3X2
Vida: 3 HP
Posición: 5-F

Nombre: Soldado5X2
Vida: 3 HP
Posición: 6-B

Nombre: Soldado0X2
Vida: 2 HP
Posición: 1-E

Nombre: Soldado1X2
Vida: 1 HP
Posición: 7-C

Nombre: Soldado2X2
Vida: 1 HP
Posición: 4-J

```

- A continuación falta lo último, que es determinar cuál de los dos ejércitos gana.
- Entonces, reciclé el mismo sistema de VideoJuegos anteriores, el ejército con mayor cantidad de vida gana.

Listing 14: Ejército ganador

```
346 public static void ejercitoGanador(Soldado[] f1, Soldado[] f2){
347     int total1 = 0, total2 = 0;
348     for(int i = 0; i < f1.length; i++){
349         total1 += f1[i].getVida();
350     }
351     for(int i = 0; i < f2.length; i++){
352         total2 += f2[i].getVida();
353     }
354     if(total1 > total2){
355         System.out.println("El ejercito 1 es ganador!");
356     }else if(total1 == total2){
357         System.out.println("Hay empate!");
358     }else{
359         System.out.println("El ejercito 2 es ganador!");
360     }
361     System.out.println("Bajo la metrica de que ejercito tiene mas vida");
362 }
```

- Finalmente, una vez todo está implementado, solo falta que el VideoJuego sea iterativo, para esto, puse todo el método main en un do-while, para que el usuario pueda volver a jugar si desea, o desea salir.

Listing 15: Método main final

```
363 public static void main(String[] args) {
364     Scanner sc = new Scanner(System.in);
365     do {
366         Soldado[] [] tablero = new Soldado[10][10];
367         int ej1 = (int) (Math.random() * 10 + 1);
368         int ej2 = (int) (Math.random() * 10 + 1);
369         int[] filas1 = numerosRandom(ej1);
370         int[] columnas1 = numerosRandom(ej1);
371         int[] filas2;
372         int[] columnas2;
373         do {
374             filas2 = numerosRandom(ej2);
375             columnas2 = numerosRandom(ej2);
376         } while (!diffCoordenadas(filas1, filas2, columnas1, columnas2));
377         Soldado[] ejercito1 = crearArreglo(filas1, columnas1, 1);
378         Soldado[] ejercito2 = crearArreglo(filas2, columnas2, 2);
379         inicializarEjercito(tablero);
380         desplegarEjercito(tablero, ejercito1);
381         desplegarEjercito(tablero, ejercito2);
382         mostrarTablero(tablero);
383         soldadoMayorVida(ejercito1, 1);
384         soldadoMayorVida(ejercito2, 2);
385         vidaPromedio(ejercito1, 1);
386         vidaPromedio(ejercito2, 2);
```

```

387     mostrarEjercito(ejercito1, 1);
388     mostrarEjercito(ejercito2, 2);
389     System.out.println("Bajo que algoritmo de ordenamiento le gustaria ordenar su
        ejercito?");
390     System.out.println("1. Ordenamiento por burbuja");
391     System.out.println("2. Ordenamiento por insercion");
392     switch (sc.nextInt()) {
393         case 1:
394             ordenamientoBurbuja(ejercito1);
395             ordenamientoBurbuja(ejercito2);
396             break;
397         case 2:
398             ordenamientoInsercion(ejercito1);
399             ordenamientoInsercion(ejercito2);
400             break;
401         default:
402     }
403     System.out.println();
404     System.out.println("Ranking de ambos ejercitos del soldado con mayor a menor vida:
        \n");
405     mostrarEjercito(ejercito1, 1);
406     mostrarEjercito(ejercito2, 2);
407     System.out.println("Presione q para salir, o cualquier otra tecla para volver a
        jugar");
408 } while (!sc.next().equals("q"));
409
410 }

```

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 cumple 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 los ítems.
- 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 están 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	4	
Total		20		18	

6. Referencias

- Fundamentos de la programación 2 - Tópicos de la programación Orientada a Objetos (Marco Aedo)