Informe de Laboratorio 12

Tema: Definicion de Clases de Usuario

Nota

Estudiante	Escuela	Asignatura
Jhonatan David Arias Quispe jariasq@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Fundamentos de Programacion 2 Semestre: II Código: 1701213

Laboratorio	Tema	Duración
12	Definicion de Clases de Usuario	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 4 Diciembre 2023	Al 11 Diciembre 2023

1. Actividades

- Al ejecutar el videojuego, el programa deberá dar las opciones:
- 1. Juego rápido (tal cual como en el laboratorio 11) Al acabar el juego mostrar las opciones de volver a jugar y de volver al menú principal. También se deberá tener la posibilidad de cancelar el juego actual en cualquier momento, permitiendo escoger entre empezar un juego totalmente nuevo o salir al menú principal.
- 2. Juego personalizado: permite festionar ejércitos. Primero se generan los 2 ejércitos con sus respectivos soldados y se muestran sus datos. Luego se tendrá que escoger cuál de los 2 ejércitos se va a gestionar, después se mostrarán las siguientes opciones:
  - Crear Soldado: permitirá crear un nuevo soldado personalizado y añadir al final del ejército (recordar que límite es de 10 soldados por ejército)
  - Eliminar Soldado (no debe permitir un ejército vacío)
  - Clonar Soldado (crea una copia exacta del soldado) y se añade al final del ejército (recordar que límite es de 10 soldados por ejército)
  - Modificar Soldado (con submenú para cambiar alguno de los atributos nivelAtaque, nivel-Defensa, vidaActual)
  - Comparar Soldados (verifica si atributos: nombre, nivelAtaque, nivelDefensa, vidaActual y vive son iguales)
  - Intercambiar Soldados (intercambia 2 soldados en sus posiciones en la estructura de datos del ejército)

- Ver soldado (Búsqueda por nombre)
- Ver ejército
- Sumar niveles (usando Method-Call Chaining), calcular las sumatorias de nivelVida, nivelAtaque, nivelDefensa, velocidad de todos los soldados de un ejército
  - Por ejemplo, si ejército tendría 3 soldados:
  - s=s1.sumar(s2).sumar(s3);
  - s es un objeto Soldado nuevo que contendría las sumatorias de los 4 atributos indicados de los 3 soldados. Ningún soldado cambia sus valores
- Jugar (se empezará el juego con los cambios realizados) y con las mismas opciones de la opción 1.
- Volver (muestra el menú principal) Después de escoger alguna de las opciones 1) a 9) se podrá volver a elegir uno de los ejércitos y se mostrarán las opciones 1) a 11)

- 3. Salir

2. Equipos, materiales y temas utilizados

- Sistema Operativo ArchCraft GNU Linux 64 bits Kernell
- NeoVim
- OpenJDK 64-Bit 20.0.1
- Git 2.42.0
- Cuenta en GitHub con el correo institucional.
- Programación Orientada a Objetos.
- Creacion de programas con CLI
- Bilioteca Graphics (origen propio)

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/JhonatanDczel/fp2-23b.git`
- URL para el laboratorio 12 en el Repositorio GitHub.
- `https://github.com/JhonatanDczel/fp2-23b/tree/main/fase02/lab12`
- El trabajo de este laboratorio es en su mayor parte lo mismo que en otros laboratorios, por lo que las variaciones seran minimas

4. Proyecto lab12

- Creamos un directorio en fase02 que contenga los archivos del laboratorio y copiamos los archivos del anterior laboratorio.
- Para el tablero se usará un array bidimensional simple.
- La estructura del laboratorio presente es:

```
>>> lab12  git:(main) tree
.
├── latex
│   ├── acts
│   │   ├── a1.tex
│   │   ├── a2.tex
│   │   ├── a3.tex
│   │   ├── a4.tex
│   │   ├── a5.tex
│   │   ├── commits.tex
│   │   ├── exec.tex
│   │   └── main.tex
│   ├── foot
│   │   ├── rubricas.aux
│   │   └── rubricas.tex
│   ├── head
│   │   ├── codeFormat.tex
│   │   ├── dataTable.tex
│   │   ├── format.tex
│   │   ├── labData.tex
│   │   └── pkgs.tex
│   ├── img
│   │   ├── exec.png
│   │   ├── logo_abet.png
│   │   ├── logo_episunsa.png
│   │   ├── logo_unsa.jpg
│   │   └── tree.jpg
│   ├── informe-latex.aux
│   ├── informe-latex.log
│   ├── informe-latex.out
│   ├── informe-latex.pdf
│   └── informe-latex.tex
├── Soldado.class
├── Soldado.java
├── VideoJuego.class
└── VideoJuego.java

6 directories, 29 files
```

5. Clase Soldado

- Los atributos para la clase soldado son:

```
1  private String nombre;
2  private int fila;
3  private int nivelAtaque = random(5);
4  private int nivelDefensa = random(5);
5  private int columna;
6  private int nivelVida;
7  private int vidaActual;
8  private int velocidad;
9  private String actitud;
10 private boolean vive;
11 private String team;
```

- Cada atributo que lo requiere, tiene sus metodos setters y getters para encapsular la información.
- Se usan 3 constructores sobrecargados que son los siguientes:

```java
public Soldado(String t) {
  team = t;
  velocidad = 0;
  vive = true;
  actitud = "ataque";

}
public Soldado(int v, String t) {
  team = t;
  velocidad = v;
  vive = true;
  actitud = "ataque";
}
public Soldado(int v, int nV, String t) {
  team = t;
  vive = true;
  velocidad = v;
  nivelVida = nV;
  actitud = "ataque";
}
```

- Estos constructores nos servirán cuando vayamos a crear soldados con distintos datos base.
- Adicionalmente tenemos los metodos de accion del soldado:

```java
public void atacar() {
  actitud = "ofensiva";
}
public void defender() {
  actitud = "defensiva";
}
public void huir() {
  actitud = "fuga";
  velocidad += 2;
}
public void avanzar() {
  velocidad += 1;
}

public void serAtacado() {
  vidaActual -= 1;
  if(vidaActual == 0) morir();
}
public void morir() {
  vive = false;
}

public void retroceder() {
  if (velocidad > 0) {
    velocidad = 0;
    actitud = "defensiva";
  } else if (velocidad == 0) {
    velocidad = -1;
  }
}
```

- Las acciones cambian los estados de los atributos del soldado, a los que accederemos despues con los metodos accesores:

```java
public String getTeam() {
  return team;
}

public void setNombre(String n) {
  nombre = n;
}

public void setFila(int f) {
  fila = f;
}

public void setColumna(int c) {
  columna = c;
}

public void setNivelVida(int p) {
  nivelVida = p;
}

public String getNombre() {
  return nombre;
}

public int getFila() {
  return fila;
}

public int getColumna() {
  return columna;
}

public int getNivelVida() {
  return nivelVida;
}

public int getNivelAtaque() {
  return nivelAtaque;
}

public int getNivelDefensa() {
  return nivelDefensa;
}

public void setNivelAtaque(int n) {
  nivelAtaque = n;
}

public void setNivelDefensa(int n) {
  nivelDefensa = n;
}

public boolean isLive() {
  return vive;
}
```

- Estos metodos accesores nos servirán para manejar la lógica interna del videojuego
- Adicionalmente tenemos algunos metodos auxiliares que usamos en la misma clase:

```java
public String toString() {
  return "Nombre: " + nombre +
```

```
    " | Ubicacion: " + fila + ", " + columna +
    " | nivelVida: " + nivelVida +
    " | Estado: " + (vive ? "Vivo" : "Muerto") +
    " | Actitud: "+ actitud +"\n" ;
  }
  private int random(int n) {
    return (int) (Math.random() * n + 1);
  }
```

6. Mostrando el tablero por pantalla

- Para generar la grafica nos apoyaremos de la biblioteca graphics, desarrollada el anterior semestre
- Tendremos dos maneras de manejar la grafica, la primera es un array dibimensional, que contiene las unbicaciones de los soldados en el tablero, y la segunda es un objeto de tipo Picture que contiene la representacion grafica del tablero en un determinado momento
- Para eso necesitaremos dos metodos, el primero de ellos genera un tablero a partir de un array bidimensional (Atributo global)

```
public static void makeGBoard(){
  for(int i = 0; i < 10; i++){
    Picture fila = null;
    for(int j = 0; j < 10; j++){
      Picture c = Picture.casilleroBlanco();
      if(board[i][j] != null){
        c = Picture.soldier().superponer(c);
        if(board[i][j].isNegro())
          c = Picture.soldier().invertir().superponer(c);
      }
      if(j == 0){
        fila = c;
        continue;
      }
      fila = fila.alLado(c);
    }
    if(i == 0){
      gBoard = fila;
      continue;
    }
    gBoard = gBoard.encima(fila);
  }
}
```

- El metodo itera sobre todos los elementos del array bidimensional "board"
- Con esto contruye el tablero tomando uno a uno sus elementos, en caso de haber un null en cierta posicion, solo imprime un casillero en blanco, en caso de haber un soldado, se pregunta si este tiene coloracion negro, en cuyo caso imprime un soldado negro sobre un fondo blanco, y caso contrario imprime un soldado blanco en casillero blanco
- El segundo metodo que usamos es el que agarra un objeto Picture, y lo grafica:

```
public static void displayBoard(){
  Graphics g = new Graphics(gBoard);
  g.print();
}
```

- El metodo es arto simple, agarra un objeto Picture, genera un nuevo objeto Graphics a partir de el, y lo muestra en pantalla

7. Ordenando un HashMap

- Antes de continuar, las especificaciones como, mostrar al soldado con mayor vida, mostrar el nivel global de vida y los datos por ejercito de todos los soldados ya han sido cubiertas desde muchos laboratorios anteriores, asi que no hay mayor cambio en su funcionamiento
- Asi que ahora nos centraremos en el ultimo cambio que se pide hacer en el laboratorio: el ranking de soldados por vida
- En anteriores laboratorios era demasiado simple implementar un metodo que una dos arrays de Soldados y luego aplicar un algoritmo de ordenamiento por vida
- Ahora las cosas se complican un poco mas, ya que los HashMap no tienen un orden especifico
- Para lograr trabajar con el HashMap, lo tendremos que convertir a un array, y para eso usamos el metodo:

```
public static Soldado[] toArray(HashMap<String, Soldado> armyH){
  Soldado[] army = new Soldado[armyH.size()];
  int i = 0;
  for(String name : armyH.keySet()){
    army[i] = armyH.get(name);
    i++;
  }
  return army;
}
```

- El codigo es simple, devuelve un array de Soldados, a partir de un HashMap
- Ahora con esto, podemos implementar otro metodo llamado ranking, que tomara dos arrays de Soldados como parametro, los unira, y los ordenara usando un algoritmo de ordenamiento:

```
public static HashMap<String, Soldado> ranking(HashMap<String, Soldado> army1, HashMap<String,
    Soldado> army2){
  Soldado[] a1 = toArray(army1);
  Soldado[] a2 = toArray(army2);
  Soldado[] total = new Soldado[a1.length + a2.length];
  int i = 0;

  for(Soldado s : a1){
    total[i] = s;
    i++;
  }
  for(Soldado s : a2){
    total[i] = s;
    i++;
  }
  bubbleSortLife(total);
  HashMap<String, Soldado> ranking = new HashMap<>();
  for(Soldado s : total){
    ranking.put(s.getName(), s);
  }
  return ranking;
}
```

- Como vemos, este metodo se apoya del metodo toArray que creamos anteriormente
- Con el metodo ranking ya creado, ahora podemos proceder a mostrar los datos por pantalla

8. Metodo principal

- Ahora procedere a mostrar el metodo main que controla todas las acciones del programa

```
public static void main(String[] args){
  HashMap<String, Soldado> army1 = initializeArmyHashMap(0, false);
  HashMap<String, Soldado> army2 = initializeArmyHashMap(1, true);
  displayArmy(army1, "Ejercito 1");
  displayArmy(army2, "Ejercito 2");
  System.out.println("Soldado con maxima vida:");
  displaySoldier(maxLife);

  HashMap<String, Soldado> ranking = ranking(army1, army2);
  displayArmy(ranking, "Ranking de soldados:");
  makeGBoard();
  displayBoard();

}
```

- El metodo inicia generando dos HashMaps
- Luego muestra usando el metodo displayArmy que veremos a continuacion:

```
public static void displayArmy(HashMap<String, Soldado> army, String str){
  System.out.println("\n===== " + str + " =====");
  for(String soldado : army.keySet()){
    displaySoldier(army.get(soldado));
  }
}

public static void displaySoldier(Soldado s){
  System.out.println(" " + s.getName() + ":");
  System.out.println(" Nivel de vida: " + s.getLife());
  System.out.println(" Fila: " + (s.getRow() + 1));
  System.out.println(" Columna: " + (s.getColumn() + 1));
  System.out.print("\n");
}
```

- El metodo se apoya de otro, (displaySoldier) que muestra los datos de un soldado, y hace eso con todo el array de Soldados
- Continuando con el metodo principal, luego muestra al soldado con la mayor puntuacion de vida
- Luego se genera un nuevo HashMap ranking"que tendra a los soldados ordenados
- Luego llama al metodo ranking e imprime su resultado
- Finalmente se crea el tablero grafico, y se muestra
- Tenemos las siguientes variables globales:

```
public static Soldado[][] board = new Soldado[10][10];
public static Picture gBoard;
public static Soldado maxLife = new Soldado("sold");
public static int promedio = 0;
```

9. Ejecucion grafica y por consola

- A continuacion se vera la Ejecucion tanto grafica como por consola

9.1. Ejecucion por consola

```
===== Ejercito 1 =====
 Soldado 0x8:
  Nivel de vida: 3
  Fila: 9
  Columna: 2

 Soldado 0x6:
  Nivel de vida: 3
  Fila: 3
  Columna: 1

 Soldado 0x7:
  Nivel de vida: 5
  Fila: 1
  Columna: 1

 Soldado 0x4:
  Nivel de vida: 4
  Fila: 2
  Columna: 2

 Soldado 0x5:
  Nivel de vida: 1
  Fila: 6
  Columna: 3

 Soldado 0x2:
  Nivel de vida: 4
  Fila: 4
  Columna: 6

 Soldado 0x3:
  Nivel de vida: 3
  Fila: 10
  Columna: 1

 Soldado 0x0:
  Nivel de vida: 4
  Fila: 2
  Columna: 6

 Soldado 0x1:
  Nivel de vida: 2
  Fila: 3
  Columna: 3


===== Ejercito 2 =====
 Soldado 1x3:
  Nivel de vida: 1
  Fila: 1
  Columna: 4

 Soldado 1x1:
  Nivel de vida: 3
  Fila: 5
  Columna: 3
```

```
 Soldado 1x2:
  Nivel de vida: 4
  Fila: 3
  Columna: 9

 Soldado 1x0:
  Nivel de vida: 1
  Fila: 6
  Columna: 5

Soldado con maxima vida:
 Soldado 0x7:
  Nivel de vida: 5
  Fila: 1
  Columna: 1


===== Ranking de soldados: =====
 Soldado 0x7:
  Nivel de vida: 5
  Fila: 1
  Columna: 1

 Soldado 0x2:
  Nivel de vida: 4
  Fila: 4
  Columna: 6

 Soldado 1x2:
  Nivel de vida: 4
  Fila: 3
  Columna: 9

 Soldado 0x0:
  Nivel de vida: 4
  Fila: 2
  Columna: 6

 Soldado 0x4:
  Nivel de vida: 4
  Fila: 2
  Columna: 2

 Soldado 1x1:
  Nivel de vida: 3
  Fila: 5
  Columna: 3

 Soldado 0x8:
  Nivel de vida: 3
  Fila: 9
  Columna: 2

 Soldado 0x6:
  Nivel de vida: 3
  Fila: 3
  Columna: 1

 Soldado 0x3:
  Nivel de vida: 3
  Fila: 10
  Columna: 1

 Soldado 0x1:
```

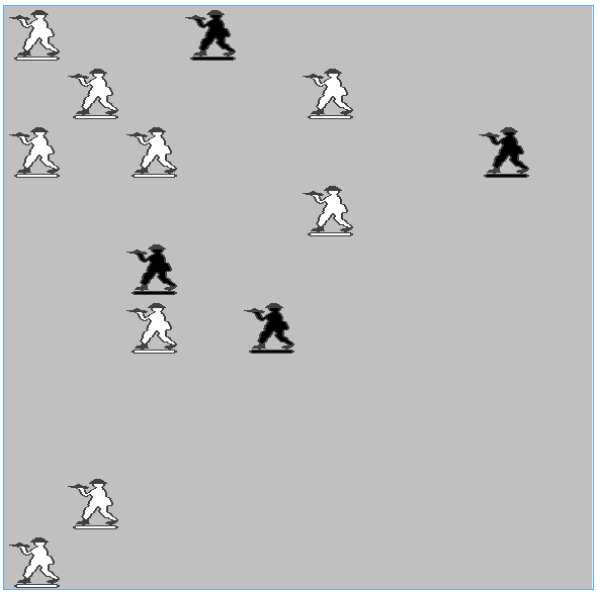
```
 Nivel de vida: 2
 Fila: 3
 Columna: 3

Soldado 1x3:
 Nivel de vida: 1
 Fila: 1
 Columna: 4

Soldado 0x5:
 Nivel de vida: 1
 Fila: 6
 Columna: 3

Soldado 1x0:
 Nivel de vida: 1
 Fila: 6
 Columna: 5
```

- Como vemos, la estructura del metodo main esta representada en la salida por consola
- A continuacion se vera la salida grafica:



10. Commits mas importantes

- A continuacion se muestran los commits mas importantes
- Los commits se hicieron siguiendo la convencion para commits de git, y siguiendo las recomendaciones practicas para hacer mensajes de commits
- Cada mensaje de commit esta estructurado por un titulo y una descripcion separados por una linea en blanco

Listing 1: commits mas importantes

```

```

```
commit c1082e4d2b725108cdbf1f4acb639a89da4174d8
Author: JhonatanDczel <jariasq@unsa.edu.pe>
Date:   Mon Oct 23 14:02:50 2023 -0500

    Creando el proyecto lab08

    Se copiaron los archivos soldado, videojuego y graphics del laboratorio
    anterior

commit a362ecf01678492c2ba977fd8c3d4f866ccc0313
Author: JhonatanDczel <jariasq@unsa.edu.pe>
Date:   Mon Oct 23 14:12:17 2023 -0500

    Hace que el metodo initialice army devuelva hashmap

commit 40035dcc9534d58e9b577105339203b1dd0aa839
Author: JhonatanDczel <jariasq@unsa.edu.pe>
Date:   Mon Oct 23 14:08:29 2023 -0500

    Creacion del metodo initialiceArmyHashMap

commit c78508b8e703041b1d11d8410744bed4c9870f31
Author: JhonatanDczel <jariasq@unsa.edu.pe>
Date:   Mon Oct 23 14:25:05 2023 -0500

    Se cambio la forma de establecer el atributo negro

    Se cambio la forma en que se establece que un ejercito sea de color
    negro, antes del cambio, se usaba un condicional para evaluar el valor
    booleano negro, ahora, se ingresa directemente este valor como atributo
    del soldado

commit 6e7b92190399f9550e32bb58243d0b07ba26d918
Author: JhonatanDczel <jariasq@unsa.edu.pe>
Date:   Mon Oct 23 15:06:45 2023 -0500

    Se crea metodo para ordenar el ejercito

    Como no se puede ordenar directamente un hashmap, ya que no mantiene un
    oprden especifico, se copia el contenido a un array antes de mandarlo al
    metodo de ordenamiento de soldados por vida

commit edfb9c396ca95be5c3038fb18c4bb282894758da
Author: JhonatanDczel <jariasq@unsa.edu.pe>
Date:   Mon Oct 23 15:48:12 2023 -0500

    Adaptando el codigo para hacer el ordenamiento

    Se adapto el codigo para crear un array grande que una a los dos
    ejercitos, luego se creo el metodo ranking que agarra estos dos
    ejercitos, los junta y crea uno mas grande que ordena con el algoritmo
    bubble sort, finalmente el resultado se muestra en pantalla
```

11. Rúbricas

11.1. 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.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 1: 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 2: 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	4	
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	1.5	
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	1.5	
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		19	