Informe de Laboratorio 03

Tema: Laboratorio 03

Nota

Estudiante	Escuela	Asignatura
Victor Mamani Anahua vmamanian@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Fundamentos de la Programación II Semestre: II Código: 20230489

Laboratorio	Tema	Duración
03	Laboratorio 03	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 17 Setiembre 2023	Al 24 Setiembre 2023

1. Tarea

- Cree un Proyecto llamado Laboratorio3
- Usted deberá agregar las clases Nave.java y DemoBatalla.java.
- Analice, complete y pruebe el Código de la clase DemoBatalla.
- Solucionar la Actividad 4 de la Práctica 1 pero usando arreglo de objetos.
- Solucionar la Actividad 5 de la Práctica 1 pero usando arreglos de objetos.

2. Equipos, materiales y temas utilizados

- Sistema Operativo Ubuntu GNU Linux 23 lunar 64 bits Kernell 6.2.v
- Visual Studio Code.
- OpenJDK 64-Bits 19.0.7.
- Git 2.39.2.
- Cuenta en GitHub con el correo institucional.
- Programación Orientada a Objetos.
- Actividades del Laboratorio 03.

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/VictorMA18/fp2-23b.git`
- URL para el laboratorio 01 en el Repositorio GitHub.
- `https://github.com/VictorMA18/fp2-23b/tree/main/Fase01/Lab03`

4. Actividades del Laboratorio 03

4.1. Ejercicio Nave

- En el primer commit agregamos un metodo a la clase Nave llamada toString() con tal que que nos retornara un string con la informacion total del objeto.
- El codigo y el commit seria el siguiente:

Listing 1: Commit

```
$ git commit -m "Agregamos un metodo toString() que nos retorne un string para que
    despues podamos imprimir este string que serian los datos del objeto"
```

Listing 2: Las lineas de codigos del metodo creado:

```
public String toString(){ //CREAMOS ESTE METODO PARA IMPRIMIR LOS DATOS DEl OBJETO
    String join = "Nombre: " + getNombre() + "\nColumna: " + getColumna() + "\nFila: "
        + getFila() + "\nEstado: " + getEstado() + "\nPuntos: " + getPuntos();
    return join;
}
```

4.2. Ejercicio DemoBatalla

- En el segundo commit tambien completamos el metodo mostrarNaves en la cual aplicamos un for each para que cada objeto de este arreglo se imprima sus datos con el metodo toString().
- El codigo , el commit y la ejecucion seria el siguiente:

Listing 3: Commit

```
$ git commit -m "Completamos el metodo mostrarNaves y en esta usamos un for each y el
    metodo toString() para mostrar datos de cada objeto y tambien le ponemos un tamano
    al arreglo de 2 para hacer pruebas"
```

Listing 4: Las lineas de codigo del metodo completado:

```
public static void mostrarNaves(Nave [] flota){ //COMPLETAMOS EL METODO mostrarNaves
    Y NOS AYUDAMOS DE UN FOR EACH PARA EL MUESTREO DE LOS DATOS DE CADA OBJETO Y
    TAMBIEN USAMOS EL METODO toString()
    for(Nave ship: flota){
        System.out.println(ship.toString());
        System.out.println("**********************");
```

```
        }
    }
```

Listing 5: La ejecucion dada:

```
Nave 1 :
Nombre: Hector
Fila: 3
Columna: A
Estado: true
Puntos: 132
Nave 2 :
Nombre: Hernan
Fila: 2
Columna: B
Estado: true
Puntos: 248

Naves creadas:
--------------------------------------
Nombre: Hector
Columna: A
Fila: 3
Estado: true
Puntos: 132
**********************
Nombre: Hernan
Columna: B
Fila: 2
Estado: true
Puntos: 248
**********************
```

4.3. Ejercicio DemoBatalla

- En el tercer commit completamos este metodo mostrarPorNombre donde aplicamos un for each para pasar por todos los elementos y ver cuales son iguales al nombre ingresado y mostrar sus datos ala vez con el metodo toString()
- El codigo , el commit , la ejecucion seria el siguiente:

Listing 6: Commit

```
$ git commit -m "Completamos el metodo mostrarPorNombre con un for each este nos
    permite comparar el nombre con cada elemento del array y en caso de ser iguales
    imprimira sus datos"
```

Listing 7: Las lineas de codigo del metodo completado:

```
public static void mostrarPorNombre(Nave [] flota){ //COMPLETAMOS EL METODO
    mostrarPorNombre Y NOS AYUDAMOS DE UN FOR EACH CON TAL QUE SI EL NOMBRE INGRESADO
    ERA IGUAL AL OBJETO CREADO MOSTRABA LOS DATOS DEL OBJETO Y TAMBIEN USAMOS EL
    METODO toString()
    Scanner sc = new Scanner(System.in);
```

```
        System.out.println("Ingrese el nombre para buscar a las naves: ");
        String nombre = sc.next();
        System.out.println("**********************");
        for(Nave ship: flota){
            if(ship.getNombre().equals(nombre)){
                System.out.println(ship.toString());
                System.out.println("**********************");
            }
        }
    }
```

Listing 8: La ejecucion dada:

```
Nave 1 :
Nombre: Victor
Fila: 3
Columna: A
Estado: true
Puntos: 167
Nave 2 :
Nombre: Victor
Fila: 6
Columna: B
Estado: true
Puntos: 167

Ingrese el nombre para buscar a las naves:
Victor
**********************
Nombre: Victor
Columna: A
Fila: 3
Estado: true
Puntos: 167
**********************
Nombre: Victor
Columna: B
Fila: 6
Estado: true
Puntos: 167
**********************
```

4.4. Ejercicio DemoBatalla

- En el cuarto commit completamos el metodo mostrarPorPuntos donde usamos un for each donde si el numero de puntos de cada nave es menor o igual al numero ingresado se imprimira sus datos.
- El codigo , el commit y la ejecucion seria el siguiente:

Listing 9: Commit

```
$ git commit -m "Completamos el metodo mostrarPorPuntos donde ingresamos un numero de
    puntos en esta usamos un for each que pase por todos los elementos donde si su
    numero de puntos es menor o igual a este numero ingresado se imprimira sus datos"
```

Listing 10: Las lineas de codigo del metodo completado:

```
public static void mostrarPorPuntos(Nave [] flota){ // Completamos el metodo
    mostrarPorPuntos donde ingresamos un numero de puntos en esta usamos un for each
    que pase por todos los elementos donde si su numero de puntos es menor o igual a
    este numero ingresado se imprimira sus datos
    Scanner sc = new Scanner(System.in);
    System.out.println("Ingrese un numero puntos para buscar a las naves que son menor
        o igual a esta: ");
    int point = sc.nextInt();
    System.out.println("**********************");
    for(Nave ship: flota){
        if(ship.getPuntos() <= point){
            System.out.println(ship.toString());
            System.out.println("**********************");
        }
    }
}
```

Listing 11: La ejecucion dada:

```
Nave 1 :
Nombre: Victor
Fila: 2
Columna: A
Estado: true
Puntos: 34
Nave 2 :
Nombre: Hector
Fila: 3
Columna: B
Estado: true
Puntos: 56

Ingrese un numero puntos para buscar a las naves que son menor o igual a esta:
56
**********************
Nombre: Victor
Columna: A
Fila: 2
Estado: true
Puntos: 34
**********************
Nombre: Hector
Columna: B
Fila: 3
Estado: true
Puntos: 56
**********************
```

4.5. Ejercicio DemoBatalla

- En el quinto commit completamos el metodo mostarMayorPuntos donde creamos un objeto Nave que es ship donde este se va rellenar con la nave que tenga la mayor cantidad de puntos
- El codigo , el commit y la ejecucion seria el siguiente:

Listing 12: Commit

```
$ git commit -m "Completamos el metodo mostarMayorPuntos donde creamos a ship que es un
    objeto en el cual se llenara con el nave que tenga la mayor cantidad de puntos y
    despues se retornara"
```

Listing 13: Las lineas de codigo del metodo completado:

```
public static Nave mostrarMayorPuntos(Nave [] flota){ //COMPLETAMOS ESTE METODO
    mostrarMayorPuntos DONDE CREAMOS UN OBJETO DE LA CLASE NAVE QUE ES SHIP EN ESTE
    PODREMOS GUARDAR LOS DATOS DE LA NAVE CON LA MAYOR CANTIDAD DE PUNTOS Y DESPUES
    RETONARNLO
    Nave ship = new Nave();
    for(int i = 0; i < flota.length - 1; i++){
        if(flota[i].getPuntos() < flota[i + 1].getPuntos()){
            ship = flota[i + 1];
        }else{
            ship = flota[i];
        }
    }
    return ship;
}
```

Listing 14: La ejecucion dada:

```
Nave 1 :
Nombre: Victor
Fila: 2
Columna: A
Estado: true
Puntos: 45
Nave 2 :
Nombre: Hector
Fila: 3
Columna: B
Estado: true
Puntos: 89

Nave con mayor numero de puntos:
Nombre: Hector
Columna: B
Fila: 3
Estado: true
Puntos: 89
-------------------------------------
```

4.6. Ejercicio DemoBatalla

- En el sexto commit CREAMOS ESTE METODO positionsNew DONDE PONEMOS EN UBICACIONES ALEATORIAS NUESTRAS NAVES QUE YA HABIAN SIDO INGRESADAS NOS AYUDAMOS CON LA CLASE RANDOM DONDE ESTA NOS PERMITE INTERCAMBIAR POSICIONES Y TAMBIEN CREAMOS UN ARRAY DE STRINGS DONDE SERIAN LAS POSICIONES DE LAS COLUMNAS Y ESTAS TAMBIEN PUEDAN CAMBIAR DESPUES DE TODO ESTO IMPRIMIMOS LOS DATOS DE LAS NAVES

- El codigo y el commit seria el siguiente:

Listing 15: Commit

```
$ git commit -m "Creamos este metodo que nos ayuda a intercambiar aleatoriamente las
    posiciones en columna y fila de las naves y despues mostrar los datos de cada nave"
```

Listing 16: Las lineas de codigo del metodo creado:

```
public static void positionsNew(Nave [] fleet){ //CREAMOS ESTE METODO positionsNew
    DONDE PONEMOS EN UBICACIONES ALEATORIAS NUESTRAS NAVES QUE YA HABIAN SIDO
    INGRESADAS NOS AYUDAMOS CON LA CLASE RANDOM DONDE ESTA NOS PERMITE INTERCAMBIAR
    POSICIONES Y TAMBIEN CREAMOS UN ARRAY DE STRINGS DONDE SERIAN LAS POSICIONES DE
    LAS COLUMNAS Y ESTAS TAMBIEN PUEDAN CAMBIAR DESPUES MOSTRAMOS LOS RESULTADOS
    Random rdm = new Random();
    String[] posCol = {"A" , "B", "C" , "D" , "E" , "F" , "G" , "H" , "I" , "J"};
    for(int i = 0; i < fleet.length; i++){
        int randomfil = rdm.nextInt(10) + 1;
        int randomcol = rdm.nextInt(10) + 1;
        fleet[i].setFila(randomfil);
        fleet[i].setColumna(posCol[randomcol]);
    }
    mostrarNaves(fleet);
}
```

4.7. Ejercicio DemoBatalla

- En el septimo commit Corregimos un error que cometi en el numero aleatorio randomcol ya que este pasaba del tamaño del arreglo poscol
- El codigo , el commit y La ejecucion completa del codigo seria el siguiente:

Listing 17: Commit

```
$ git commit -m "Arreglando valor de randomcolpara su funcionamiento ya que pasaba del
    numero de elementos del arreglo poscol"
```

Listing 18: Las lineas de codigo del metodo creado:

```
public static void positionsNew(Nave [] fleet){ //CREAMOS ESTE METODO positionsNew
    DONDE PONEMOS EN UBICACIONES ALEATORIAS NUESTRAS NAVES QUE YA HABIAN SIDO
    INGRESADAS NOS AYUDAMOS CON LA CLASE RANDOM DONDE ESTA NOS PERMITE INTERCAMBIAR
    POSICIONES Y TAMBIEN CREAMOS UN ARRAY DE STRINGS DONDE SERIAN LAS POSICIONES DE
    LAS COLUMNAS Y ESTAS TAMBIEN PUEDAN CAMBIAR DESPUES MOSTRAMOS LOS RESULTADOS
    Random rdm = new Random();
    String[] posCol = {"A" , "B", "C" , "D" , "E" , "F" , "G" , "H" , "I" , "J"};
    for(int i = 0; i < fleet.length; i++){
        int randomfil = rdm.nextInt(10) + 1;
        int randomcol = rdm.nextInt(10);
        fleet[i].setFila(randomfil);
        fleet[i].setColumna(posCol[randomcol]);
    }
    mostrarNaves(fleet);
}
```

Listing 19: La ejecucion del codigo completo:

```
Nave 1 :
Nombre: Victor
Fila: 3
Columna: A
Estado: true
Puntos: 65
Nave 2 :
Nombre: Pepe
Fila: 4
Columna: B
Estado: true
Puntos: 87

Naves creadas:
--------------------------------------
Nombre: Victor
Columna: A
Fila: 3
Estado: true
Puntos: 65
**********************
Nombre: Pepe
Columna: B
Fila: 4
Estado: true
Puntos: 87
**********************
--------------------------------------
Ingrese el nombre para buscar a las naves:
Victor
**********************
Nombre: Victor
Columna: A
Fila: 3
Estado: true
Puntos: 65
**********************
--------------------------------------
Ingrese un numero puntos para buscar a las naves que son menor o igual a esta:
98
**********************
Nombre: Victor
Columna: A
Fila: 3
Estado: true
Puntos: 65
**********************
Nombre: Pepe
Columna: B
Fila: 4
Estado: true
Puntos: 87
**********************
--------------------------------------
Nave con mayor numero de puntos:
```

```
Nombre: Pepe
Columna: B
Fila: 4
Estado: true
Puntos: 87
**********************
--------------------------------------
Naves ordenadas Aleatoriamente:
Nombre: Victor
Columna: G
Fila: 5
Estado: true
Puntos: 65
**********************
Nombre: Pepe
Columna: J
Fila: 6
Estado: true
Puntos: 87
**********************
--------------------------------------
```

4.8. Ejercico02 del Lab03

- En el octavo commit creamos una CREAMOS LA CLASE SOLDIER PARA PODER USAR ARREGLO DE OBJETOS EN LA ACTIVDAD 04 DONDE SE NOS PIDE EL NOMBRE Y LA VIDA DEL SOLDADO
- El codigo y el commit seria el siguiente:

Listing 20: Commit

```
$ git commit -m "Creamos una clase soldier en Ejercicio02 para poder usarla en la
    actividad04 del lab01 ya que nos pide usar arreglo de objetos donde cada soldado se
    le pide su nombre y su vida , tambien movimos los archivos DemoBatalla.java y
    Nave.java a una nueva carpeta"
```

Listing 21: Las lineas de codigo de lo creado:

```
// Laboratorio Nro 3 - Activdad 4 - Practica 1
// Autor: Mamani Anahua Victor Narciso
// Colaboro:
// Tiempo:
public class Soldier { //CREAMOS LA CLASE SOLDIER PARA PODER USAR ARREGLO DE OBJETOS EN
    LA ACTIVDAD 04 DONDE SE NOS PIDE EL NOMBRE Y LA VIDA DEL SOLDADO

   private String name;
   private int heatlh;

   // Metodos mutadores
   public void setName(String n){
      name = n;
   }
   public void setHealth(int p){
```

```
        heatlh = p;
    }

    // Metodos accesores
    public String getName(){
        return name;
    }
    public int getHealth(){
        return heatlh;

    }
    // Completar con otros metodos necesarios
    public String toString(){ //CREAMOS ESTE METODO PARA IMPRIMIR LOS DATOS DEl OBJETO
        String join = "Nombre: " + getName() + "\nVida: " + getHealth();
        return join;
    }
}
```

4.9. Ejercico02 del Lab03

- En el noveno commit EN ESTE EJERCICIO USAMOS ARREGLO DE OBJETOS CON LA CALSE SOLDIER DONDE INGRESAMOS UN NOMBRE Y VIDA PARA CADA SOLDADO Y DESPUES IMPRIMIMOS SUS DATOS CON LA AYUDA DE LA ESTRUCTURA FOR
- El codigo , el commit y la ejecucion seria el siguiente:

Listing 22: Commit

```
$ git commit -m "En el archivo Soldier.java modificamos un barra espaciadora y en el
    otro archivo creamos el arreglo de objetos donde pedimos un nombre y una vida para
    cada soldado , le anadimos y imprimimos sus datos de cada soldado esto usando la
    estructura for"
```

Listing 23: Las lineas de codigo de lo creado:

```
import java.util.*;
public class Ejercicio02_lab03 {
    public static void main(String args[]){ //EN ESTE EJERCICIO USAMOS ARREGLO DE
        OBJETOS CON LA CALSE SOLDIER DONDE INGRESAMOS UN NOMBRE Y VIDA PARA CADA SOLDADO
        Y DESPUES IMPRIMIMOS SUS DATOS CON LA AYUDA DE LA ESTRUCTURA FOR
        Scanner sc = new Scanner(System.in);
        Soldier[] soldiers = new Soldier[5];
        for(int i = 0; i < soldiers.length; i++){
            System.out.println("Soldado " + (i + 1) + " : ");
            System.out.print("Ingrese su nombre: ");
            String name = sc.next();
            System.out.print("Ingrese su vida: ");
            int heatlh = sc.nextInt();
            soldiers[i] = new Soldier();
            soldiers[i].setName(name);
            soldiers[i].setHealth(heatlh);
        }
        for(int i = 0; i < soldiers.length;i++){
            System.out.print("\nLos datos del soldado " + (i + 1) + " : ");
```

```
            System.out.println(soldiers[i].toString());
            System.out.println("************************************");
        }
    }
}
```

Listing 24: La ejecucion del codigo completo:

```
Soldado 1 :
Ingrese su nombre: Victor
Ingrese su vida: 34
Soldado 2 :
Ingrese su nombre: Tato
Ingrese su vida: 56
Soldado 3 :
Ingrese su nombre: Pepe
Ingrese su vida: 54
Soldado 4 :
Ingrese su nombre: Pablo
Ingrese su vida: 23
Soldado 5 :
Ingrese su nombre: Tasha
Ingrese su vida: 12

Los datos del soldado 1 :
Nombre: Victor
Vida: 34
************************************

Los datos del soldado 2 :
Nombre: Tato
Vida: 56
************************************

Los datos del soldado 3 :
Nombre: Pepe
Vida: 54
************************************

Los datos del soldado 4 :
Nombre: Pablo
Vida: 23
************************************

Los datos del soldado 5 :
Nombre: Tasha
Vida: 12
************************************
```

4.10. Ejercico03 del Lab03

- En el decimo commit creamos una CREAMOS LA CLASE SOLDIER PARA PODER USAR ARREGLO DE OBJETOS EN LA ACTIVDAD 04 DONDE SE NOS PIDE EL NOMBRE Y LA VIDA DEL SOLDADO

- El codigo y el commit seria el siguiente:

Listing 25: Commit

```
$ git commit -m "Creamos una clase soldier en Ejercicio02 para poder usarla en la
    actividad04 del lab01 ya que nos pide usar arreglo de objetos donde cada soldado se
    le pide su nombre y su vida , tambien movimos los archivos DemoBatalla.java y
    Nave.java a una nueva carpeta"
```

Listing 26: Las lineas de codigo de lo creado:

```
// Laboratorio Nro 3 - Actividad 4 - Practica 1
// Autor: Mamani Anahua Victor Narciso
// Colaboro:
// Tiempo:
public class Soldier { //CREAMOS LA CLASE SOLDIER PARA PODER USAR ARREGLO DE OBJETOS EN
    LA ACTIVDAD 04 DONDE SE NOS PIDE EL NOMBRE Y LA VIDA DEL SOLDADO

   private String name;
   private int heatlh;

   // Metodos mutadores
   public void setName(String n){
      name = n;
   }
   public void setHealth(int p){
      heatlh = p;
   }

   // Metodos accesores
   public String getName(){
      return name;
   }
   public int getHealth(){
      return heatlh;

   }
   // Completar con otros metodos necesarios
   public String toString(){ //CREAMOS ESTE METODO PARA IMPRIMIR LOS DATOS DEl OBJETO
      String join = "Nombre: " + getName() + "\nVida: " + getHealth();
      return join;
   }
}
```

4.11. Ejercico03 del Lab03

- En el undecimo commit creamos la clase Soldier01 y tambien en el otro archivo creamos 2 ejercitos conun arreglo de objeto de la calse Soldier01 donde solo necesitamos el nombre de cada soldado para esto creamos un metodo fillinName() que nos retornara un arreglo ya relleno con un numero de soldados que es aleatorio y sus nombres y otro metodo showSoldiers() que nos muestra los datos de cada soldado de cada ejercito y el metodo battleResult() que nos da el resultado de la batalla dependiendo del numero de soldados en cada ejercito.

- El codigo , el commit y la ejecucion seria el siguiente:

Listing 27: Commit

```
$ git commit -m " Creamos la clase Soldier01 y tambien en el otro archivo creamos 2
    ejercitos conun arreglo de objeto de la calse Soldier01 donde solo necesitamos el
    nombre de cada soldado para esto creamos un metodo fillinName() que nos retornara
    un arreglo ya relleno con un numero de soldados que es aleatorio y sus nombres y
    otro metodo showSoldiers() que nos muestra los datos de cada soldado de cada
    ejercito y el metodo battleResult() que nos da el resultado de la batalla
    dependiendo del numero de soldados en cada jercito"
```

Listing 28: Las lineas del codigo Soldier01 de lo creado:

```
// Laboratorio Nro 3 - Ejercico03 - Lab03
// Autor: Mamani Anahua Victor Narciso
// Colaboro:
// Tiempo:
public class Soldier01{ //CREAMOS LA CLASE SOLDIER PARA PODER USAR ARREGLO DE OBJETOS
    EN LA ACTIVDAD 04 DONDE SE NOS PIDE EL NOMBRE Y LA VIDA DEL SOLDADO

   private String name;

   // Metodos mutadores
   public void setName(String n){
      name = n;
   }

   // Metodos accesores
   public String getName(){
      return name;
   }

   // Completar con otros metodos necesarios
   public String toString(){ //CREAMOS ESTE METODO PARA IMPRIMIR LOS DATOS DEl OBJETO
      String join = "\nNombre: " + getName();
      return join;
   }
}
```

Listing 29: Las lineas del codigo Ejercico03-lab03 de lo creado:

```
import java.util.*;
public class Ejercicio03_lab03 {
   public static void battleResult(Soldier01[] army1, Soldier01[] army2){
      if(army1.length < army2.length){
         System.out.println("El ganador de la batalla es el Ejercito 02 con " +
             (army2.length) + " soldados");
         System.out.println("********************************");
      }else if(army1.length > army2.length){
         System.out.println("El ganador de la batalla es el Ejercito 01 con " +
             (army1.length) + " soldados");
         System.out.println("********************************");
      }else{
         System.out.println("El resultado de la batalla es un EMPATE");
         System.out.println("********************************");
      }
   }
```

```
    public static void showSoldiers(Soldier01[] army){
       for(int i = 0; i < army.length; i++){
          System.out.print("Los datos del Soldado" + (i + 1));
          System.out.println(army[i].toString());
          System.out.println("*******************************");
       }
    }
    public static Soldier01[] fillinName(){
       Random rdm = new Random();
       Soldier01[] army = new Soldier01[rdm.nextInt(5) + 1];
       for(int i = 0; i < army.length; i++){
          army[i] = new Soldier01();
          army[i].setName("Soldado" + (i + 1));
       }
       return army;
    }
    public static void main(String args[]){
       Soldier01[] army1 = fillinName();
       Soldier01[] army2 = fillinName();
       System.out.println("-----------------------------------");
       System.out.println("Los soldados del Ejercito 01: ");
       showSoldiers(army1);
       System.out.println("///////////////////////////////");
       System.out.println("Los soldados del Ejercito 02: ");
       showSoldiers(army2);
       System.out.println("-----------------------------------");
       battleResult(army1, army2);
    }
}
```

Listing 30: La ejecucion del codigo completo:

```
-----------------------------------
Los soldados del Ejercito 01:
Los datos del Soldado1
Nombre: Soldado1
*******************************
Los datos del Soldado2
Nombre: Soldado2
*******************************
Los datos del Soldado3
Nombre: Soldado3
*******************************
Los datos del Soldado4
Nombre: Soldado4
*******************************
Los datos del Soldado5
Nombre: Soldado5
*******************************
///////////////////////////////
Los soldados del Ejercito 02:
Los datos del Soldado1
Nombre: Soldado1
*******************************
Los datos del Soldado2
Nombre: Soldado2
```

```
    ********************************
    Los datos del Soldado3
    Nombre: Soldado3
    ********************************
    Los datos del Soldado4
    Nombre: Soldado4
    ********************************
    Los datos del Soldado5
    Nombre: Soldado5
    ********************************
    ------------------------------------
    El resultado de la batalla es un EMPATE
    ********************************
```

4.12. Estructura de laboratorio 03

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

```
/Lab03
|----Ejercicio02
       |----Ejercicio02_lab03.java
       +----Soldier.java
|----Ejercicio03
       |----Ejercicio03_lab03.java
       +----Soldier01.java
|----Latex
       |----img
              |----logo_abet.png
              |----logo_episunsa.png
              |----logo_unsa.jpg
              +----pseudocodigo_insercion.png
       |----Informe03.aux
       |----Informe03.fdb_latexmk
       |----Informe03.fls
       |----Informe03.log
       |----Informe03.out
       |----Informe03.pdf
       |----Informe03.synctex.gz
       |----Informe03.tex
       +----src
          +----Nave01.java
+----NaveyDemoBatalla
   |----DemoBatalla.java
   +----Nave.java
```

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	1	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	1	
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		14	

6. Referencias

- https://drive.google.com/file/d/1gF5iR4EpCOfMuwdQCGPfbErUFeA3cp_U/view