ITID433
Programación Avanzada
Facultad de Ingeniería Eléctrica y Electrónica

Práctica 04

Encapsulamiento

Objetivos:

Esta práctica de la asignatura persigue los siguientes objetivos:

- Practicar el encapsulamiento.
- Reforzar la comprensión sobre modificadores de acceso.
- Reforzar la comprensión sobre variables y métodos de clase y sobre variables y métodos de instancia.

Instrucciones y Actividades:

1. Marco Teórico

El encapsulamiento es uno de los cuatro pilares de la Programación Orientada a Objetos. Consiste en ocultar los detalles internos de una clase y exponer solo lo necesario a través de métodos públicos. Esto ayuda a proteger los datos y a mantener la integridad del estado de los objetos.

Modificadores de Acceso

Java proporciona varios modificadores de acceso para controlar la visibilidad de los miembros de una clase:

- `public`: accesible desde cualquier clase.
- `private`: accesible solo dentro de la misma clase.
- `protected`: accesible dentro del mismo paquete y por subclases.
- (sin modificador): accesible dentro del mismo paquete.

Variables de Clase y de Instancia

Variables de Instancia: Son variables declaradas dentro de una clase, pero fuera de cualquier método. Cada objeto tiene sus propios valores en las variables de instancia.

Variables de Clase: Son variables declaradas con el modificador `static`. Solo hay una copia de las variables de clase compartida por todas las instancias de la clase.

Métodos de Clase y de Instancia

Métodos de Instancia: Operan sobre las variables de instancia y requieren una instancia de la clase para ser llamados.

Métodos de Clase: Pueden ser llamados sin crear una instancia de la clase. Solo pueden operar sobre variables de clase.

ITID433
Programación Avanzada
Facultad de Ingeniería Eléctrica y Electrónica

Getters y Setters

Los getters y setters son métodos públicos utilizados para acceder y modificar los valores de las variables privadas de una clase.

2. Clase Persona

1. Define la clase `Persona`, con la cual se ilustrarán los conceptos relacionados al encapsulamiento. La clase `Persona` tendrá dos variables de instancia para almacenar el nombre y la edad, así como una variable de clase para llevar un registro de la cantidad de objetos de tipo `Persona` creados. La clase `Persona` debe ser pública, mientras que el estado debe ser privado.

```
public class Persona {
    // Variables de instancia
    private String nombre;
    private int edad;

    // Variable de clase
    private static int contadorPersonas = 0;

    // Constructor

    // métodos de instancia

    // métodos de clase
}
```

2. Define un constructor sin parámetros, el cual acepte como argumento el nombre y la edad, e inicialice el estado del objeto, así también incremente el contador. Agrega el constructor en la sección `// constructor` de la clase.

```
public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
        contadorPersonas++;
}
```

3. Define getter y setter para el nombre de la persona. Los métodos deben ser públicos. Agrégalos en la sección `// métodos de instancia`.

```
 // Getter para nombre
public String getNombre() {
       return nombre;
 }

// Setter para nombre
public void setNombre(String nombre) {
        this.nombre = nombre;
```

```
}
```

4. Define getter y setter para la edad de la persona. Los métodos deben ser públicos. Agrégalos en la sección `// métodos de instancia`.

```
// Getter para edad
public int getEdad() {
        return edad;
}

// Setter para edad
public void setEdad(int edad) {
        this.edad = edad;
}
```

5. Agrega un método denominado `mostrarInfo` que presente en consola el nombre y edad de la persona. Agrégalo en la sección `// métodos de instancia`.

```
public void mostrarInfo() {
        System.out.println("Nombre: " + nombre +
", Edad: " + edad);
}
```

6. Define getter para recuperar el contador de personas. Agrégalo en la sección `// métodos de clase`.

```
public static int getContadorPersonas() {
        return contadorPersonas;
}
```

7. Emplea la clase `Persona`, crea dos objetos de tipo `Persona` (Alicia con 20 años y Juan con 15 años), presenta la información. Modifica el nombre de Alicia por Alice y configura su edad en 31 años y presenta la información. Presenta el número de personas creadas.

```
public class Main {
    public static void main(String[] args) {
        // Crear objetos de la clase Persona
        Persona p1 = new Persona("Alicia", 20);
        Persona p2 = new Persona("Juan", 15);

        // Usar métodos de instancia
        p1.mostrarInfo();
        p2.mostrarInfo();

        // Modificar valores usando setters
        p1.setNombre("Alice");
        p1.setEdad(31);
```

```
        // Mostrar información actualizada
        p1.mostrarInfo();

        // Usar método de clase
        System.out.println("Número de personas
creadas: " + Persona.getContadorPersonas());
    }
}
```

8. Crea dos objetos de tipo `Persona` (uno con tu información, y el otro con la información de tu compañero) y presenta la información. Cambia tu edad a -1 y la edad de tu compañero a 0. Presenta la información modificada, así como presenta el número de personas creadas.

9. Crea una variable de tipo `Persona` llamada `q` y asígnala a `p1`. Presenta la información de `q`. Indica el número de personas creadas. Explica por qué el valor cambia, si es que cambio o por qué no cambia, si es que no cambio, en la sección Resultados. Cambia el nombre de `q` a Adriana. Presenta la información de `q` y de `p1`. Explica por qué el nombre almacenado en `q` y el nombre almacenado en `p1` es igual, o explica por qué el nombre almacenado en `p1` y el nombre almacenado en q no es igual, en la sección Resultados.

```
Persona q = p1;
q.mostrarInfo();
System.out.println("Número de personas creadas: "
+ Persona.getContadorPersonas());
q.setNombre("Adriana");
q.mostrarInfo();
p1.mostrarInfo();
```

10. Añade validaciones. Cómo pudiste apreciar se puede colocar valores negativos o 0 en la edad, lo cual no es coherente. Asegúrate de que la `edad` en la clase `Persona` no pueda ser negativa, para lo cual modifica el setter, en caso de que la edad sea menor que cero presenta un mensaje de error.

```
public void setEdad(int edad) {
    if (edad >= 0) {
        this.edad = edad;
    } else {
        System.out.println("La edad no puede ser
negativa.");
    }
}
```

3. Ejemplo básico de clases para D&D

1. Se va a realizar un ejercicio sencillo de un juego de D&D. Para lo cual se van a definir dos clases: `Dado` y `Jugador`.

2. Define la clase `Dado`, la cual permitirá manejar diferentes tipos de dados (los dados pueden tener 3, 4, 5, 6 o más caras; el dado más conocido es el dado de 6 caras). La clase `Dado` debe tener una variable de instancia privada que almacene el número de lados y agrega una variable de clase privada que permita llevar una cuenta del número de dados creados en la clase.

```
public class Dado {
    // variables de instancia
    private int numLados;

    // variables de clase
    private static int numDados;

    // Constructor

    // métodos de instancia

    // métodos de clase
}
```

3. En la clase `Dado` incluye un constructor que inicialice el número de lados, así como incremente en uno el valor del número de dados. Colócalo en la sección `// Constructor`

```
public Dado(int numLados) {
    this. numLados = numLados;
    numDados++;
}
```

4. En la clase `Dado` incluye un método de instancia público que permita lanzar el dado. El método debe devolver un valor considerando que debe estar en el rango entre 1 y el número de lados. Colócalo en la sección respectiva.

```
public int lanzar() {
    return (int) (Math.random() * numLados) + 1;
}
```

5. En la clase `Dado` incluye un getter para conocer el número de lados de un dado.

```
public int getNumLados() {
      return numLados;
}
```

6. En la clase `Dado` incluye un método de clase que permita conocer el número de dados creados.

```
public static int getNumDados() {
      return numDados;
}
```

7. Define la clase `Jugador`, la cual permitirá almacenar el nombre, el tipo, los puntos de vida y el dado que empleará. El tipo debe permitir establecer la clase del jugador, pudiendo escoger entre *fighter*, *wizard* y *archer*; cada tipo tiene un dado con un número de caras específico para controlar su ataque, para el tipo se usará una enumeración. En la enumeración define una variable de tipo final int que establezca el número de caras del dado (debe ser final por que NO puede cambiarse una vez inicializado), define un constructor en el que se defina el número de caras del dado y define el getter para conocer el número de caras del dado. En la enumeración define que *fighter* tendrá un dado de 20 caras, *archer* uno de 10 caras y *wizard* uno de 6.

```
public class Jugador{
    // variables de instancia
    private String nombre;
    private Dado dadoAtaque;
    private Tipo tipoJugador;
    private int puntosVida;

    // definición de enumeración para el tipo
    public enum Tipo {
        FIGHTER(20),
        ARCHER(10),
        WIZARD(6);

        // estado de la enumeración
        private final int caras;

        // constructor
        Tipo(int caras) {
            this.caras = caras;
        }

        // getter
        public int getCaras() {
            return caras;
        }
    }

    // Constructor
    // Métodos de instancia
}
```

8. En la clase `Jugador` incluye un constructor que inicialice el nombre, tipo, y puntos de vida. Para el dado, inicializa el mismo usando el valor definido en la enumeración (usa el getter). Colócalo en la sección `// Constructor`

```
public Jugador(String nombre, Tipo tipoJugador, int
puntosVida) {
   this.nombre = nombre;
   this.tipoJugador = tipoJugador;
   this.dadoAtaque = new Dado(tipoJugador.getCaras());
   this.puntosVida = puntosVida;
}
```

9. En la clase `Jugador` incluye el método `getNombre` para obtener el nombre del jugador.

10. En la clase `Jugador` incluye el método `atacar`, el cual acepta como argumento un objeto de tipo Jugador sobre el cual se realizará el ataque. El ataque se basa en una probabilidad de ataque, para lo cual debes lanzar los dados y si se supera el valor 6 se realiza el ataque, en cuyo caso el que recibe el ataque recibe el daño (es decir se llama al método `recibirAtaque` sobre el objeto que se pasó como argumento); caso contrario se indica que el jugador falló en el ataque. Para recibir el ataque se usará un método que permita obtener el ataque (`getAtaque`). Colócalo en la sección `// método de instancia`

```
public int atacar(Jugador j) {
   int ataque = dadoAtaque.lanzar();
   System.out.println(nombre + " (" + tipoJugador + ")
ataca a " + j.getNombre() + "! lazó un " + ataque);

   if(ataque >= 6) {
       j.recibirAtaque(getAtaque());
       System.out.println(j.getNombre() + " recibio
     daño!");
   } else {
       System.out.println(nombre + " falló!");
   }
   return ataque;
}
```

11. En la clase `Jugador` incluye el método `recibirAtaque`, el cual acepta como argumento un valor de tipo entero. En este método se debe restar de los puntos de vida el valor de ataque. Colócalo en la sección `// método de instancia`

```
public void recibirAtaque(int daño) {
    puntosVida -= daño;
    System.out.println(nombre + " recibe " + daño + "
puntos de daño!");
}
```

12. En la clase `Jugador` incluye el método `estaVivo`, el cual retorna verdadero si el jugador tiene puntos de vida. Colócalo en la sección `// método de instancia`

```
public boolean estaVivo() {
    return puntosVida > 0;
}
```

13. En la clase `Jugador` incluye el método `getAtaque`, el cual retorna un número entero en función del tipo de Jugador. Este método será privado y solo podrá ser llamado desde otros métodos. Colócalo en la sección `// método de instancia`

```
private int getAtaque() {
    switch(tipoJugador) {
        case FIGHTER:
            return 3;
        case ARCHER:
            return 2;
        case WIZARD:
            return 1;
        default:
            return 0;
    }
}
```

14. Agrega una clase denominada `Main` para realizar el juego. Agrega el método `main`. En el método main crea dos jugadores, Aragorn y Legolas, Aragorn será un *fighter* y Legolas un *archer*. Haz que los dos jugadores se ataquen, primero ataca el *fighter* y luego el *archer*. Los dos deben atacar hasta que uno fallezca. Establece el ganador.

```
public class Main {
    public static void main(String[] args) {
       Jugador f1 = new Jugador("Aragorn",
Jugador.Tipo.FIGHTER, 80);
       Jugador a1 = new Jugador("Legolas",
Jugador.Tipo.ARCHER, 60);

       while (f1.estaVivo() && a1.estaVivo()) {
           f1.atacar(a1);
              if (a1.estaVivo()) {
                  a1.atacar(f1);
            }
        }

        if (f1.estaVivo()) {
            System.out.println(f1.getNombre() + "
gana!");
        } else {
            System.out.println(a1.getNombre() + "
gana!");
        }
```

```
        }
    }
```

15. Modifica el método `main` para crear dos jugadores, tú y tu amigo. Escoge algún tipo para ti y para tu amigo. La vida de ambos debe ser 50. Tú y tu amigo se enfrentan a Aragorn y Legolas. Haz que tú ataques a Aragorn y tu amigo a Legolas. Si Aragorn o Legolas muere primero ustedes ganan, caso contrario ganarán Aragorn y Legolas.

16. Imprime la cantidad de dados que se crearon de la clase `Dado`.

17. Modifica la clase `Jugador` e incluye dos variables de clase `cantExitos`, `cantFallidos`. Ambas deben ser privadas. Incrementa el valor de `cantExitos` cada vez que un ataque sea exitoso (reduzca la vida del otro jugador) e incrementa el valor de `cantFallidos` cada vez que un ataque falle. Genera un getter para cada variable de clase. Imprime los valores de cantidad de ataques exitosos y fallidos. Define una variable en el `Main` que permita llevar la cuenta del número de ataques que se producen en una pelea y preséntalo.

4. Entregables

Para esta práctica debes realizar lo siguiente:

1. Completa el código solicitado.
2. Ubica la carpeta de los proyectos generadas, en caso de que hayas usado Eclipse o IntelliJ, elimina la carpeta `bin` o `out` y procede a comprimir las carpetas del proyecto. Este archivo comprimido (.ZIP), con el nombre cumpliendo el formato indicado en clase, es la que debes subir en el Práctica 04. En caso de que uses Replit usa un replit para cada proyecto; descarga los archivos `.java` de un replit, colócalos en una carpeta; repite este proceso con el otro replit y comprime ambas carpetas, y ese archivo comprímelo y cárgalo en la plataforma.

5. Realizado por:

David Mejía N.