Contenido

[Temario: Aprendiendo POO en Java con Yu-Gi-Oh! 2](#_Toc142311011)

[Módulo 1: Introducción a la Programación Orientada a Objetos y Java 2](#_Toc142311012)

[Módulo 2: Objetos y Clases en el Mundo de Yu-Gi-Oh! 3](#_Toc142311013)

[Módulo 3: Encapsulamiento y Modificadores de Acceso en Duelos de Yu-Gi-Oh! 6](#_Toc142311014)

[Módulo 4: Herencia y Duelos entre Clases 6](#_Toc142311015)

[Módulo 5: Composición y Relaciones en el Campo de Batalla 6](#_Toc142311016)

[Módulo 6: Manejo de Excepciones y Trampas en el Código 6](#_Toc142311017)

[Módulo 7: Persistencia de Datos y Guardado de Partidas 7](#_Toc142311018)

[Módulo 8: Proyecto Final - Desarrollo de un Juego de Yu-Gi-Oh! Simplificado 7](#_Toc142311019)

¡Por supuesto! Aquí tienes un temario básico para aprender programación orientada a objetos (POO) en Java utilizando el tema de Yu-Gi-Oh! como contexto. La idea es combinar la enseñanza de los conceptos de POO con elementos del mundo de Yu-Gi-Oh! para hacer el aprendizaje más divertido e interesante.

# Temario: Aprendiendo POO en Java con Yu-Gi-Oh!

# Módulo 1: Introducción a la Programación Orientada a Objetos y Java

**1. ¿Qué es la Programación Orientada a Objetos? Conceptos básicos.**

La Programación Orientada a Objetos (POO) es un paradigma de programación que se basa en el concepto de "objetos", los cuales son unidades independientes y autónomas que contienen tanto datos (atributos) como comportamientos (métodos) relacionados. La POO busca modelar el mundo real de una manera más natural y estructurada, permitiendo una mayor organización, reutilización y comprensión del código.

Principios Fundamentales de la POO:

| **Principio** | **Descripción** |
| --- | --- |
| **Abstracción** | Representación de los aspectos esenciales de un objeto, ocultando detalles innecesarios. |
| **Encapsulamiento** | Ocultación de los detalles internos de un objeto, exponiendo solo la interfaz necesaria. |
| **Herencia** | Creación de nuevas clases basadas en clases existentes, heredando atributos y comportamientos. |
| **Polimorfismo** | Tratar objetos de diferentes clases de manera uniforme, permitiendo intercambiar tipos de manera coherente. |

**2. Introducción a Java: Historia y características principales.**

**3. Configuración del entorno de desarrollo: Instalación de JDK y configuración de IDE.**

# Módulo 2: Objetos y Clases en el Mundo de Yu-Gi-Oh!

**1. Clases y Objetos: Definición y diferencia.**

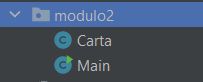
| Concepto | Descripción |
| --- | --- |
| Clases y Objetos | Una clase es una plantilla que define la estructura y el comportamiento de un tipo de objeto. Un objeto es una instancia concreta de una clase. |
| Atributos | Los atributos son variables que almacenan datos relacionados con un objeto. En Yu-Gi-Oh!, los atributos podrían representar características de las cartas. |
| Métodos | Los métodos son funciones que definen el comportamiento de un objeto. En Yu-Gi-Oh!, los métodos podrían permitir activar efectos o realizar acciones en el juego. |
| Constructores | Los constructores son métodos especiales utilizados para inicializar objetos al crear instancias de una clase. |
| Instancia | Una instancia es un objeto concreto creado a partir de una clase. |

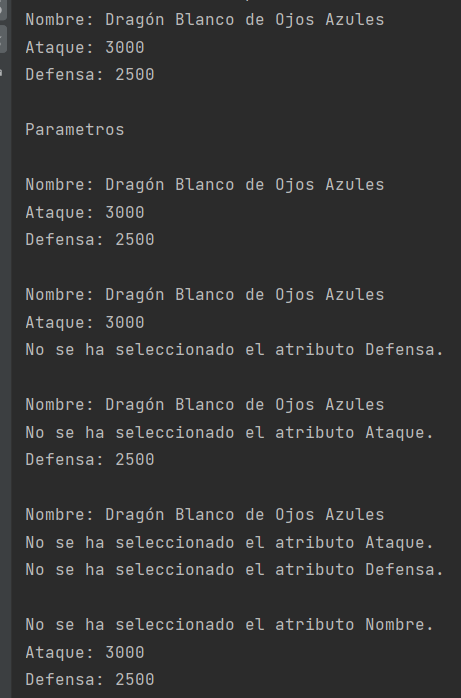
**2. Creación de clases y objetos basados en cartas de Yu-Gi-Oh!.**

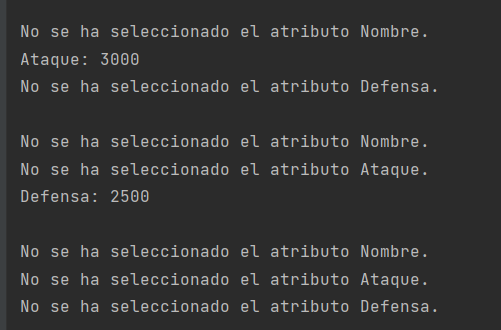
**3. Atributos y métodos: Asociación con características de las cartas.**

package modulo2;  
  
public class Carta {  
  
 //ATRBUTOS  
 String nombre;  
 int ataque;  
 int defensa;  
  
 //CONSTRUCTOR  
 public Carta(String nombre, int ataque, int defensa) {  
 this.nombre = nombre;  
 this.ataque = ataque;  
 this.defensa = defensa;  
 }  
  
 //METODOS  
 public void mostrarInformacion() {  
 System.*out*.println("Nombre: " + nombre);  
 System.*out*.println("Ataque: " + ataque);  
 System.*out*.println("Defensa: " + defensa);  
 }  
  
 public void mostrarInformacionParametros(boolean name, boolean atk, boolean def) {  
 System.*out*.println("");  
 if (name) {  
 System.*out*.println("Nombre: " + nombre);  
 } else {  
 System.*out*.println("No se ha seleccionado el atributo Nombre.");  
 }  
  
 if (atk) {  
 System.*out*.println("Ataque: " + ataque);  
 } else {  
 System.*out*.println("No se ha seleccionado el atributo Ataque.");  
 }  
  
 if (def) {  
 System.*out*.println("Defensa: " + defensa);  
 } else {  
 System.*out*.println("No se ha seleccionado el atributo Defensa.");  
 }  
  
 }  
  
}

package modulo2;  
  
public class Main {  
 public static void main(String[] args) {  
  
 //Creacion de una instancia (objeto) de la clase carta  
 Carta miCarta = new Carta("Dragón Blanco de Ojos Azules", 3000, 2500);  
  
 miCarta.mostrarInformacion();  
  
 System.*out*.println("\nParametros");  
 // Combinaciones de parámetros para mostrarInformacionParametros  
 miCarta.mostrarInformacionParametros(true, true, true);  
 miCarta.mostrarInformacionParametros(true, true, false);  
 miCarta.mostrarInformacionParametros(true, false, true);  
 miCarta.mostrarInformacionParametros(true, false, false);  
 miCarta.mostrarInformacionParametros(false, true, true);  
 miCarta.mostrarInformacionParametros(false, true, false);  
 miCarta.mostrarInformacionParametros(false, false, true);  
 miCarta.mostrarInformacionParametros(false, false, false);  
  
 }//MAIN  
}//CLASS







# Módulo 3: Encapsulamiento y Modificadores de Acceso en Duelos de Yu-Gi-Oh!

**1. Encapsulamiento: Definición y ventajas.**

**Concepto de Encapsulamiento:** El encapsulamiento es uno de los principios fundamentales de la Programación Orientada a Objetos (POO). En el contexto de Yu-Gi-Oh!, el encapsulamiento se refiere a la práctica de ocultar los detalles internos de una carta o jugador y proporcionar una interfaz controlada para interactuar con ellos. Esto se logra mediante la restricción del acceso directo a los atributos y métodos internos y utilizando métodos públicos para acceder y modificar estos valores.

**2. Modificadores de acceso: public, private, protected y default.**

**Modificadores de Acceso:** Los modificadores de acceso son palabras clave utilizadas en la declaración de atributos y métodos para controlar el nivel de acceso que otros componentes del programa tienen a ellos. En Yu-Gi-Oh!, estos modificadores ayudan a determinar qué información y acciones están disponibles para los jugadores y cómo interactúan con las cartas.

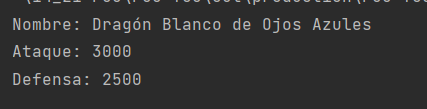
Tipos de Modificadores de Acceso:

* **public**: El atributo o método es accesible desde cualquier parte del programa, incluyendo otras clases.
* **private**: El atributo o método solo es accesible desde dentro de la misma clase. Otros componentes no pueden acceder directamente a ellos.
* **protected**: El atributo o método es accesible desde la misma clase y también desde subclases (herederos).
* **default** (sin modificador): El atributo o método es accesible solo dentro del mismo paquete.

**3. Aplicación de encapsulamiento y modificadores en la representación de cartas y jugadores**.

package modulo3;  
  
public class Carta {  
  
 //ATRBUTOS  
 private String nombre;  
 private int ataque;  
 private int defensa;  
  
 //CONSTRUCTOR  
  
 //METODOS  
 public void mostrarInformacion() {  
 System.*out*.println("Nombre: " + nombre);  
 System.*out*.println("Ataque: " + ataque);  
 System.*out*.println("Defensa: " + defensa);  
 }  
  
 //GET AND SET  
 public String getNombre() {  
 return nombre;  
 }  
  
 public void setNombre(String nombre) {  
 this.nombre = nombre;  
 }  
  
 public int getAtaque() {  
 return ataque;  
 }  
  
 public void setAtaque(int ataque) {  
 this.ataque = ataque;  
 }  
  
 public int getDefensa() {  
 return defensa;  
 }  
  
 public void setDefensa(int defensa) {  
 this.defensa = defensa;  
 }  
  
}

package modulo3;  
  
public class Main {  
 public static void main(String[] args) {  
  
 Carta miCarta = new Carta();  
  
 miCarta.setNombre("Dragón Blanco de Ojos Azules");  
 miCarta.setAtaque(3000);  
 miCarta.setDefensa(2500);  
  
 miCarta.mostrarInformacion();  
  
  
 }//MAIN  
}//CLASS



# Módulo 4: Herencia y Duelos entre Clases

1. Herencia: Concepto y jerarquías de clases.

2. Clases base y clases derivadas: Relación en el contexto de Yu-Gi-Oh!.

3. Polimorfismo: Utilización de interfaces y clases abstractas para representar diferentes tipos de cartas.

# Módulo 5: Composición y Relaciones en el Campo de Batalla

1. Composición: Creación de objetos complejos a partir de otros objetos.

2. Relaciones entre clases: Asociación, agregación y composición en los duelos de Yu-Gi-Oh!.

3. Implementación de relaciones entre cartas, mazos y jugadores.

# Módulo 6: Manejo de Excepciones y Trampas en el Código

1. Excepciones en Java: Tipos y manejo.

2. Implementación de manejo de excepciones en el juego de Yu-Gi-Oh!.

3. Creación de trampas (clases de manejo de excepciones personalizadas) para situaciones específicas en el juego.

# Módulo 7: Persistencia de Datos y Guardado de Partidas

1. Serialización: Guardado y carga de objetos en archivos.

2. Implementación de la persistencia de datos para guardar y cargar partidas de Yu-Gi-Oh!.

3. Uso de archivos para mantener un registro de las cartas, mazos y resultados de los duelos.

# Módulo 8: Proyecto Final - Desarrollo de un Juego de Yu-Gi-Oh! Simplificado

1. Aplicación de todos los conceptos aprendidos en la creación de un juego básico de Yu-Gi-Oh! en consola.

2. Implementación de duelos, cartas, jugadores, reglas básicas y persistencia de datos.

3. Pruebas y depuración del juego finalizado.

Recuerda que este temario es una guía general y puedes adaptarlo según tus necesidades y nivel de conocimiento. ¡Diviértete aprendiendo POO en Java con el emocionante mundo de Yu-Gi-Oh! como contexto!