**Título del Curso:** Dominando la Programación Orientada a Objetos con IntelliJ IDEA: De la Teoría a la Práctica (con Videos Explicativos)

**Objetivo General:** Al finalizar este curso, serás capaz de comprender los principios fundamentales de la Programación Orientada a Objetos y aplicarlos de manera práctica utilizando el entorno de desarrollo IntelliJ IDEA para construir aplicaciones robustas y bien estructuradas, reforzando tu aprendizaje con videos explicativos en español para cada concepto.

**Metodología:** Cada módulo constará de sesiones teóricas donde se explicarán los conceptos clave con ejemplos claros y concisos, complementadas con un video explicativo en español. Inmediatamente después de cada bloque teórico y la visualización del video, realizarás ejercicios prácticos diseñados para reforzar lo aprendido, utilizando IntelliJ IDEA para escribir, ejecutar y depurar tu código.

**Estructura del Curso:**

**Módulo 1: Fundamentos de la Programación Orientada a Objetos**

* **Sesión Teórica 1: Introducción a la POO (¿Qué es, por qué es importante?)**
  + ¿Qué es la Programación Orientada a Objetos?
  + Paradigmas de programación: Imperativa vs. Orientada a Objetos.
  + Beneficios de la POO: Modularidad, reutilización, mantenibilidad, extensibilidad.
  + Conceptos clave: Clases, objetos, atributos, métodos.
  + **Video Explicativo (Ejemplo):** [Buscaré un video en YouTube como "Programación Orientada a Objetos Explicación Fácil" o similar]
  + **Ejercicio Práctico 1 en IntelliJ IDEA:**
    - Configuración inicial de un proyecto Java en IntelliJ IDEA.
    - Creación de una clase simple (ej. Coche) con algunos atributos (ej. marca, modelo, color).
    - Creación de objetos (instancias) de la clase Coche y asignación de valores a sus atributos.
    - Impresión de los atributos de los objetos en la consola.
* **Sesión Teórica 2: Clases y Objetos (Definición, instanciación, atributos, métodos)**
  + Definición y sintaxis de una clase en Java.
  + Declaración de atributos (variables de instancia).
  + Definición de métodos (funciones dentro de una clase).
  + El constructor: Inicialización de objetos.
  + La palabra clave this.
  + **Video Explicativo (Ejemplo):** [Buscaré un video en YouTube como "Clases y Objetos en Java Tutorial" o similar]
  + **Ejercicio Práctico 2 en IntelliJ IDEA:**
    - Añadir métodos a la clase Coche (ej. acelerar(), frenar()).
    - Llamar a los métodos de los objetos Coche.
    - Crear múltiples constructores con diferentes parámetros.

**Módulo 2: Los Pilares de la POO - Encapsulamiento**

* **Sesión Teórica 3: Encapsulamiento (Modificadores de acceso, ocultación de datos)**
  + ¿Qué es el encapsulamiento y por qué es importante?
  + Modificadores de acceso: public, private, protected, (y el modificador por defecto).
  + Ocultación de datos: Hacer los atributos private.
  + Métodos getter y setter para acceder y modificar atributos privados.
  + Beneficios del encapsulamiento: Control de acceso, integridad de los datos, flexibilidad.
  + **Video Explicativo (Ejemplo):** [Buscaré un video en YouTube como "Encapsulamiento en POO Ejemplo" o similar]
  + **Ejercicio Práctico 3 en IntelliJ IDEA:**
    - Modificar la clase Coche para hacer sus atributos private.
    - Implementar métodos getter y setter para los atributos de Coche.
    - Intentar acceder directamente a los atributos privados desde fuera de la clase (para observar el error).
    - Utilizar los getter y setter para interactuar con los atributos.

**Módulo 3: Los Pilares de la POO - Herencia**

* **Sesión Teórica 4: Herencia (Clases padre e hijas, reutilización de código)**
  + ¿Qué es la herencia y cómo funciona?
  + Clases padre (superclases) y clases hijas (subclases).
  + La palabra clave extends.
  + Reutilización de código a través de la herencia.
  + El constructor en las clases hijas y la llamada a super().
  + **Video Explicativo (Ejemplo):** [Buscaré un video en YouTube como "Herencia en Java Explicación" o similar]
  + **Ejercicio Práctico 4 en IntelliJ IDEA:**
    - Crear una clase padre Vehiculo con atributos y métodos comunes (ej. velocidadMaxima, arrancar()).
    - Crear clases hijas Coche y Moto que hereden de Vehiculo.
    - Añadir atributos y métodos específicos a las clases hijas.
    - Crear objetos de las clases hijas y acceder a los atributos y métodos heredados y propios.

**Módulo 4: Los Pilares de la POO - Polimorfismo**

* **Sesión Teórica 5: Polimorfismo (Sobrecarga y sobreescritura de métodos, interfaces)**
  + ¿Qué es el polimorfismo?
  + Sobrecarga de métodos (overloading): Métodos con el mismo nombre pero diferentes parámetros.
  + Sobrescritura de métodos (overriding): Métodos en la clase hija con la misma firma que en la clase padre.
  + La palabra clave @Override.
  + Polimorfismo en tiempo de ejecución (upcasting y downcasting - introducción).
  + **Video Explicativo (Ejemplo):** [Buscaré un video en YouTube como "Polimorfismo en Java Tutorial" o similar]
  + **Ejercicio Práctico 5 en IntelliJ IDEA:**
    - Sobrecargar el constructor de la clase Coche.
    - Sobrescribir el método arrancar() en las clases Coche y Moto para que tengan un comportamiento específico.
    - Crear un método en la clase padre Vehiculo que pueda aceptar objetos de sus clases hijas y llamar al método arrancar() de forma polimórfica.
* **Sesión Teórica 6: Interfaces**
  + ¿Qué es una interfaz y para qué sirve?
  + Definición de interfaces en Java.
  + Implementación de interfaces con la palabra clave implements.
  + Múltiple herencia de tipos a través de interfaces.
  + **Video Explicativo (Ejemplo):** [Buscaré un video en YouTube como "Interfaces en Java Explicación" o similar]
  + **Ejercicio Práctico 6 en IntelliJ IDEA:**
    - Crear una interfaz Conducible con un método conducir().
    - Hacer que las clases Coche y Moto implementen la interfaz Conducible.
    - Crear una lista de objetos que implementan la interfaz Conducible y llamar al método conducir() de cada uno.

**Módulo 5: Abstracción y Principios SOLID (Introducción)**

* **Sesión Teórica 7: Abstracción (Clases abstractas, interfaces)**
  + ¿Qué es la abstracción y cómo se logra en POO?
  + Clases abstractas: Definición y uso.
  + Métodos abstractos.
  + Diferencias entre clases abstractas e interfaces.
  + **Video Explicativo (Ejemplo):** [Buscaré un video en YouTube como "Clases Abstractas en Java Tutorial" o similar]
  + **Ejercicio Práctico 7 en IntelliJ IDEA:**
    - Crear una clase abstracta Figura con un método abstracto calcularArea().
    - Crear clases concretas Circulo y Rectangulo que hereden de Figura e implementen el método calcularArea().
* **Sesión Teórica 8: Principios SOLID (Single Responsibility, Open/Closed, Liskov Substitution - Introducción)**
  + Introducción a los principios SOLID como guía para el buen diseño orientado a objetos.
  + **Single Responsibility Principle (SRP):** Una clase debe tener una y solo una razón para cambiar.
  + **Open/Closed Principle (OCP):** Las entidades de software deben estar abiertas para la extensión, pero cerradas para la modificación.
  + **Liskov Substitution Principle (LSP):** Las subtipos deben ser sustituibles por sus tipos base.
  + **Video Explicativo (Ejemplo):** [Buscaré un video en YouTube como "Principios SOLID Explicación Sencilla" o similar]
  + **Ejercicio Práctico 8 en IntelliJ IDEA:**
    - Analizar ejemplos de código que violan y cumplen el SRP.
    - Refactorizar un ejemplo simple para aplicar el SRP.

**Módulo 6: Manejo de Excepciones y Colecciones (Relación con POO)**

* **Sesión Teórica 9: Manejo de Excepciones (try-catch-finally)**
  + ¿Qué son las excepciones y por qué son importantes?
  + Bloques try, catch y finally.
  + Tipos de excepciones (checked vs. unchecked).
  + Lanzamiento de excepciones personalizadas (throw new Exception(...)).
  + **Video Explicativo (Ejemplo):** [Buscaré un video en YouTube como "Manejo de Excepciones en Java Tutorial" o similar]
  + **Ejercicio Práctico 9 en IntelliJ IDEA:**
    - Crear un método que pueda lanzar una excepción (ej. división por cero).
    - Utilizar bloques try-catch para manejar la excepción.
    - Implementar un bloque finally para asegurar que cierto código se ejecute siempre.
* **Sesión Teórica 10: Colecciones (Listas, conjuntos, mapas - su relación con la POO)**
  + Introducción a las colecciones en Java (List, Set, Map).
  + Cómo las colecciones se utilizan para trabajar con objetos.
  + Ejemplos de almacenamiento y manipulación de objetos utilizando colecciones.
  + **Video Explicativo (Ejemplo):** [Buscaré un video en YouTube como "Colecciones en Java Explicación" o similar]
  + **Ejercicio Práctico 10 en IntelliJ IDEA:**
    - Crear una lista de objetos de la clase Coche.
    - Iterar sobre la lista y acceder a los atributos y métodos de cada objeto.
    - Utilizar un Map para almacenar coches utilizando su matrícula como clave.

A medida que avancemos en cada sesión, te proporcionaré los enlaces específicos a los videos que encuentre más adecuados para complementar el material teórico. ¿Qué te parece esta adición de videos al curso?