Dominando la Programación Orientada a Objetos: Tu Proyecto de Software Real

Ingeniería | Ingeniería de sistemas | Aprendizaje Basado en Proyectos

Descripción

Este plan de clase está diseñado para estudiantes universitarios de Ingeniería de Sistemas con el propósito de introducirlos y profundizar en los conceptos clave de Programación Orientada a Objetos (POO) a través de la metodología de Aprendizaje Basado en Proyectos. A lo largo de seis sesiones, los estudiantes desarrollarán un proyecto tangible que resuelve un problema real, permitiéndoles aplicar principios como clases, objetos, encapsulamiento, herencia y polimorfismo en un contexto práctico y colaborativo.

El aprendizaje de POO es fundamental para el diseño y desarrollo de software moderno, ya que facilita la creación de sistemas flexibles, reutilizables y mantenibles. Esta experiencia conecta directamente con sus futuras responsabilidades profesionales y proyectos de software reales, dotándolos de habilidades técnicas y competencias de trabajo en equipo y autonomía. Además, el enfoque práctico y activo motiva a los estudiantes a internalizar los conceptos y a desarrollar un pensamiento crítico para solucionar problemas complejos en ingeniería de software.

Objetivos de Aprendizaje

- Analizar los principios fundamentales de la Programación Orientada a Objetos y su aplicación en el desarrollo de software.
- Diseñar y construir un proyecto de software utilizando clases, objetos y relaciones de herencia.
- Implementar técnicas de encapsulamiento y polimorfismo para mejorar la modularidad y flexibilidad del código.
- Evaluar críticamente y depurar el código orientado a objetos en equipos colaborativos.
- Comunicar efectivamente los resultados y el proceso de desarrollo del proyecto a través de presentaciones y documentación técnica.

Recursos Necesarios

- Computadoras con entorno de desarrollo integrado (IDE) para programación Java o C# (1 por estudiante o pareja)
- Acceso a internet para consulta de documentación y tutoriales (mínimo 1 por grupo)
- Proyector y pantalla para presentaciones
- Documentación impresa con conceptos básicos de POO (diagramas UML, definiciones)
- Software para gestión colaborativa (GitHub o similar)
- Plantillas para planificación y diseño de proyecto (en formato digital o impreso)
- Pizarra blanca y marcadores

Requisitos Previos

- Conocimientos básicos de programación estructurada (variables, estructuras de control, funciones).
- Familiaridad con algún lenguaje de programación imperativo (como C, Python o Java básico).
- Habilidades básicas en trabajo colaborativo y manejo de herramientas informáticas.
- Comprensión previa de conceptos de algoritmos y lógica de programación.

Actividades

Sesión 1: Introducción y Fundamentos de la Programación Orientada a Objetos

Fase de Inicio

Tiempo estimado: 20 minutos

Propósito de la sesión:

Contextualizar a los estudiantes sobre la importancia de la Programación Orientada a Objetos y presentar el proyecto que desarrollarán. Activar conocimientos previos para conectar con conceptos básicos de programación.

Activación de conocimientos previos:

- **Docente:** Presenta la pregunta: "¿Qué diferencias existen entre un programa que usa solo funciones y variables y uno que utiliza objetos?"
- **Estudiantes:** Forman parejas para discutir durante 5 minutos y luego comparten ideas en plenaria.

Motivación y enganche:

Docente: Muestra un video corto (3 minutos) sobre cómo la POO es utilizada en aplicaciones populares actuales como videojuegos, redes sociales o sistemas bancarios, enfatizando la importancia de entender estos conceptos para crear software robusto.

Contextualización:

Docente: Explica cómo la POO facilita el desarrollo de sistemas complejos y cómo este aprendizaje se aplicará en un proyecto real que desarrollarán en equipo.

Estudiantes: Escuchan, preguntan dudas iniciales y reflexionan sobre sus experiencias previas.

Fase de Desarrollo

Tiempo estimado: 90 minutos

Presentación del contenido:

Docente: Introduce los conceptos de clases y objetos con ejemplos sencillos usando diagramas UML y fragmentos de código. Explica el vocabulario clave y los fundamentos teóricos brevemente, invitando a los estudiantes a explorar en sus IDEs.

Actividades de aprendizaje activo:

- **Actividad 1: Modelado de clases básicas**
  - Objetivo: Analizar y modelar clases y objetos.
  - Instrucciones:
    - En grupos de 3-4, analizarán un caso sencillo: modelar una clase "Estudiante" con atributos y métodos básicos.
    - Diseñarán un diagrama UML en papel o digital.
    - Discutirán qué atributos y métodos son esenciales y por qué.
  - Organización: Grupos de 3-4 estudiantes
  - Producto: Diagrama UML de la clase "Estudiante" con atributos y métodos definidos.
  - Tiempo estimado: 35 minutos
  - Rol docente: Circular entre grupos, hacer preguntas guía como "¿Cómo relacionarías esta clase con otras en un sistema académico?" o "¿Qué métodos podrías agregar para mejorar la funcionalidad?"
- **Actividad 2: Programando el primer objeto**
  - Objetivo: Implementar clases y crear objetos en código.
  - Instrucciones:
    - Individualmente, programar la clase "Estudiante" definida en la actividad anterior en su IDE.
    - Crear al menos dos objetos con diferentes valores y probar métodos simples (como mostrar información).
  - Organización: Individual
  - Producto: Código fuente funcional y capturas de pantalla de la ejecución.
  - Tiempo estimado: 40 minutos
  - Rol docente: Apoyar con dudas técnicas, verificar comprensión y sugerir mejoras en el código.
- **Actividad 3: Presentación rápida y retroalimentación**
  - Objetivo: Comunicar el diseño y resultados iniciales.
  - Instrucciones:
    - Cada grupo presenta su diagrama UML y código a otro grupo para recibir retroalimentación.
    - Se realiza una breve discusión sobre dificultades y aprendizajes.
  - Organización: Grupos emparejados
  - Producto: Feedback escrito en notas adhesivas o digital.
  - Tiempo estimado: 15 minutos

- Rol docente: Facilitar la dinámica y asegurar que el feedback sea constructivo.

Diferenciación:

Para estudiantes que terminan antes: Se les propone extender la clase "Estudiante" con nuevos métodos que simulen acciones (por ejemplo, registrar calificaciones) y pensar en una clase relacionada para modelar.

Para estudiantes que requieran apoyo: Se ofrece material complementario con ejemplos paso a paso y tutorías breves personalizadas para aclarar dudas de sintaxis y conceptos.

Transiciones:

Al cerrar la sesión, el docente conecta el modelado básico con la próxima sesión donde se abordará la herencia, invitando a pensar qué clases podrían derivar de "Estudiante".

Fase de Cierre

Tiempo estimado: 10 minutos

Síntesis:

- Los estudiantes completan un organizador gráfico digital o en papel con los conceptos aprendidos: definición de clase, objeto, atributos y métodos.

Reflexión metacognitiva:

- ¿Cómo te ayudó el modelar y programar la clase "Estudiante" a entender mejor los conceptos de POO?
- ¿Qué dificultades encontraste al traducir el diagrama UML a código?
- ¿Cómo crees que la POO facilita la organización del software en comparación con la programación tradicional?

Retroalimentación:

Docente: Recoge los organizadores y reflexiones para comentar aspectos destacados y áreas a mejorar en la próxima sesión.

Transferencia:

Se anticipa que en la siguiente sesión se trabajará con herencia y polimorfismo, conceptos que ampliarán la funcionalidad de las clases creadas.

Tarea o reto:

Investigar ejemplos reales de software que utilicen herencia y traer al menos un ejemplo para compartir en la próxima sesión.

Sesión 2: Herencia y Polimorfismo en Acción

Fase de Inicio

Tiempo estimado: 15 minutos

Propósito de la sesión:

Revisar la tarea, conectar con la sesión anterior y presentar los conceptos de herencia y polimorfismo para ampliar el proyecto.

Activación de conocimientos previos:

- **Docente:** Pregunta inicial: "¿Qué ejemplos encontraron de herencia en sistemas reales? ¿Cómo creen que se representa en código?"
- **Estudiantes:** Comparten ejemplos y discuten brevemente en plenaria.

Motivación y enganche:

Docente: Presenta un mini reto: "Modifiquen su proyecto para crear una clase derivada de 'Estudiante' llamada 'EstudianteBecado' que tenga atributos específicos y comportamientos diferentes."

Contextualización:

Se explica cómo la herencia y polimorfismo permiten evitar duplicación de código y manejar diferentes tipos de objetos en sistemas reales.

Fase de Desarrollo

Tiempo estimado: 95 minutos

- **Actividad 1: Diseño de herencia**
  - Objetivo: Diseñar jerarquías de clases que reflejen herencia.
  - Instrucciones:
    - En grupos, ampliarán el diagrama UML para incluir la clase derivada 'EstudianteBecado'.
    - Definirán atributos y métodos adicionales, y cómo sobreescribirán métodos de la clase base.
  - Organización: Grupos de 3-4
  - Producto: Diagrama UML actualizado con herencia y detalles de polimorfismo.
  - Tiempo estimado: 40 minutos
  - Rol docente: Facilitar preguntas como "¿Qué métodos necesitan ser modificados?", "¿Cómo se comportará un objeto de la clase derivada frente a uno base?"
- **Actividad 2: Implementación y pruebas**
  - Objetivo: Implementar herencia y polimorfismo en código.
  - Instrucciones:
    - Cada estudiante modifica su código para agregar la clase derivada y probar métodos polimórficos.

        - Realizan pruebas con objetos de ambas clases y documentan resultados.
    - Organización: Individual
    - Producto: Código funcional con herencia y polimorfismo, pruebas y documentos.
    - Tiempo estimado: 40 minutos
    - Rol docente: Supervisar, resolver dudas y sugerir mejoras en la implementación.
- **Actividad 3: Debate sobre ventajas y desafíos**
    - Objetivo: Evaluar críticamente la utilidad de herencia y polimorfismo.
    - Instrucciones:
        - En plenaria, discutir las ventajas y posibles problemas encontrados al implementar herencia.
        - Registrar ideas en pizarra para referencia futura.
    - Organización: Plenaria
    - Producto: Lista colectiva de ventajas y desafíos.
    - Tiempo estimado: 15 minutos
    - Rol docente: Moderar discusión y destacar puntos clave.

Diferenciación:

Estudiantes avanzados pueden explorar interfaces o clases abstractas para profundizar el concepto de polimorfismo.

Estudiantes con dificultad reciben ejemplos guiados y apoyo para entender la relación entre clases base y derivadas.

Transiciones:

Al final, el docente conecta la herencia con la siguiente sesión que abordará encapsulamiento y abstracción para robustecer el diseño.

Fase de Cierre

Tiempo estimado: 10 minutos

Síntesis:

- Realizar un mapa mental colectivo sobre herencia y polimorfismo en el aula o digital.

Reflexión metacognitiva:

- ¿Cómo mejoró la organización del código con herencia?
- ¿Qué dificultades encontraste al implementar polimorfismo?
- ¿Cómo usarías estos conceptos en un sistema real?

Retroalimentación:

Docente comenta sobre los mapas y reflexiones, enfatizando buenas prácticas.

Transferencia:

Se anticipa la importancia del encapsulamiento para proteger datos y métodos.

Tarea o reto:

Diseñar una clase adicional que herede de 'EstudianteBecado' y describir qué funcionalidad específica tendría.

Sesión 3: Encapsulamiento y Abstracción para un Código Seguro y Claro

Fase de Inicio

Tiempo estimado: 15 minutos

Propósito de la sesión:

Introducir conceptos de encapsulamiento y abstracción para mejorar la seguridad y claridad en programación orientada a objetos.

Activación de conocimientos previos:

- **Docente:** Pregunta: "¿Por qué creen que es importante proteger los datos dentro de una clase? ¿Cómo podrían hacerlo?"
- **Estudiantes:** Discuten en parejas y comparten ideas.

Motivación y enganche:

Se presenta un caso real de un error en software por acceso no controlado a variables internas, generando un fallo crítico.

Contextualización:

Se conecta el concepto de encapsulamiento con la necesidad de proteger datos y definir interfaces claras para sus objetos.

Fase de Desarrollo

Tiempo estimado: 90 minutos

- **Actividad 1: Modificando el código para encapsulamiento**
  - Objetivo: Aplicar encapsulamiento usando modificadores de acceso y métodos getter/setter.
  - Instrucciones:
    - En parejas, modifican su proyecto para hacer privados los atributos de las clases y crear métodos públicos para acceso controlado.
    - Prueban que el código funcione correctamente y documentan los cambios.

- Organización: Parejas
- Producto: Código modificado y documentación de diseño.
- Tiempo estimado: 50 minutos
- Rol docente: Supervisar, corregir malos usos y sugerir mejoras en diseño.

- **Actividad 2: Diseño de clases abstractas**
  - Objetivo: Comprender y aplicar el concepto de abstracción mediante clases abstractas o interfaces.
  - Instrucciones:
    - En grupos, diseñan una clase abstracta “Persona” que reúna atributos comunes y defina métodos abstractos para ser implementados por “Estudiante” y “EstudianteBecado”.
    - Discuten ventajas de este diseño para su proyecto.
  - Organización: Grupos de 3-4
  - Producto: Diagrama UML actualizado y boceto de código.
  - Tiempo estimado: 40 minutos
  - Rol docente: Facilitar comprensión, responder preguntas y guiar el diseño.

Diferenciación:

Para estudiantes con mayor facilidad, se propone implementar interfaces y explorar polimorfismo avanzado.

Para quienes requieren apoyo, se brindan ejemplos de código con explicación detallada y tutorías personalizadas.

Transiciones:

Se conecta la importancia de estos conceptos con la próxima sesión que abordará la integración y documentación del proyecto.

Fase de Cierre

Tiempo estimado: 15 minutos

Síntesis:

- Completar un resumen en 3 ideas sobre encapsulamiento y abstracción en grupos pequeños.

Reflexión metacognitiva:

- ¿Cómo afectó el encapsulamiento la seguridad de los datos en su programa?
- ¿Qué ventajas ofrece la abstracción en el diseño de software?
- ¿Qué retos enfrentaron al implementar estas técnicas?

Retroalimentación:

El docente evalúa resúmenes y reflexiones, destacando los avances conceptuales.

Transferencia:

Se anticipa la importancia de documentar y presentar el proyecto para garantizar su comprensión y mantenimiento.

Tarea o reto:

Preparar un breve informe escrito sobre la importancia del encapsulamiento y la abstracción en proyectos reales.

Sesión 4: Diseño Colaborativo y Documentación del Proyecto

Fase de Inicio

Tiempo estimado: 15 minutos

Propósito de la sesión:

Preparar a los estudiantes para trabajar colaborativamente en la integración y documentación de su proyecto orientado a objetos.

Activación de conocimientos previos:

- **Docente:** Pregunta: "¿Qué elementos creen que no pueden faltar en la documentación de un proyecto de software?"
- **Estudiantes:** En grupos pequeños comparten ideas y luego se discuten en plenaria.

Motivación y enganche:

Se presenta un ejemplo de mala documentación y sus consecuencias negativas para el mantenimiento del software.

Contextualización:

Se explica la importancia de la documentación técnica y cómo será parte del producto final que desarrollarán.

Fase de Desarrollo

Tiempo estimado: 90 minutos

- **Actividad 1: Planificación y asignación de roles**
  - Objetivo: Organizar el trabajo colaborativo para integrar el proyecto.
  - Instrucciones:
    - En equipos, definirán roles (programador, documentador, tester) para el desarrollo del proyecto.
    - Elaborarán un cronograma simple para las tareas siguientes.
  - Organización: Grupos de 4-5 estudiantes
  - Producto: Plan de trabajo y asignación de roles.
  - Tiempo estimado: 30 minutos

- Rol docente: Supervisar, sugerir ajustes y asegurar compromiso.

- **Actividad 2: Integración del código y documentación**
  - Objetivo: Integrar las partes de código y preparar documentación técnica básica.
  - Instrucciones:
    - Cada grupo integra el código de sus miembros en un repositorio común (GitHub o similar).
    - Documentan las clases y métodos con comentarios y preparan un manual de usuario sencillo.
  - Organización: Grupos
  - Producto: Repositorio con código integrado y documentación básica.
  - Tiempo estimado: 50 minutos
  - Rol docente: Asesorar en manejo de repositorios, revisión de documentación y control de versiones.

Diferenciación:

Estudiantes avanzados pueden explorar herramientas avanzadas de documentación como Javadoc o Doxygen.

Estudiantes con dificultades pueden recibir apoyo para manejo básico de repositorios y plantillas para documentación.

Transiciones:

Se anticipa que la siguiente sesión se enfocará en pruebas, depuración y mejora del proyecto.

Fase de Cierre

Tiempo estimado: 15 minutos

Síntesis:

- Realizar un cuadro comparativo entre documentación buena y mala.

Reflexión metacognitiva:

- ¿Qué ventajas ofrece trabajar en equipo con roles definidos?
- ¿Cómo ayuda la documentación en el desarrollo y mantenimiento del software?
- ¿Qué dificultades enfrentaron y cómo las superaron?

Retroalimentación:

Docente brinda comentarios sobre organización y documentación, y reconoce avances colaborativos.

Transferencia:

Se vincula esta experiencia con prácticas profesionales y trabajo en empresas de software.

Tarea o reto:

Investigar pruebas unitarias y su importancia para la calidad del software.

Sesión 5: Pruebas, Depuración y Mejora Continua

Fase de Inicio

Tiempo estimado: 15 minutos

Propósito de la sesión:

Introducir conceptos y prácticas de pruebas y depuración orientadas a objetos para mejorar la calidad del proyecto.

Activación de conocimientos previos:

- **Docente:** Pregunta: "¿Cómo identifican y corrigen errores en sus programas? ¿Qué técnicas conocen?"
- **Estudiantes:** Discuten y comparten experiencias.

Motivación y enganche:

Se presenta el impacto de errores no detectados en proyectos reales y la necesidad de pruebas rigurosas.

Contextualización:

Se explica cómo las pruebas unitarias y la depuración mejoran la confiabilidad del software orientado a objetos.

Fase de Desarrollo

Tiempo estimado: 90 minutos

- **Actividad 1: Creación de pruebas unitarias**
  - Objetivo: Diseñar y ejecutar pruebas unitarias para clases y métodos.
  - Instrucciones:
    - En parejas, escribirán pruebas unitarias para las clases desarrolladas usando frameworks como JUnit o NUnit.
    - Ejecutarán las pruebas y documentarán resultados.
  - Organización: Parejas
  - Producto: Código de pruebas y reporte de resultados.
  - Tiempo estimado: 50 minutos
  - Rol docente: Asistir con configuración y análisis de resultados.
- **Actividad 2: Sesión de depuración colaborativa**
  - Objetivo: Identificar y corregir errores en el código.
  - Instrucciones:
    - En grupos, presentan errores encontrados y colaboran para depurarlos usando herramientas del IDE.
    - Ejercen técnicas de revisión de código y documentación de correcciones.
  - Organización: Grupos

- Producto: Código corregido, lista de errores y soluciones.
- Tiempo estimado: 40 minutos
- Rol docente: Facilitar discusión, sugerir buenas prácticas y validar correcciones.

Diferenciación:

Estudiantes avanzados pueden investigar pruebas de integración o automatización.

Quienes necesiten apoyo reciben ejemplos guiados y tutorías para interpretar resultados de pruebas.

Transiciones:

Se conecta la mejora continua con la preparación para la presentación final del proyecto en la siguiente sesión.

Fase de Cierre

Tiempo estimado: 15 minutos

Síntesis:

- Completar un cuadro de errores comunes y cómo se resolvieron.

Reflexión metacognitiva:

- ¿Qué aprendiste sobre la importancia de las pruebas?
- ¿Cómo la depuración colaborativa mejoró tu código?
- ¿Qué técnicas te parecen más útiles para asegurar calidad?

Retroalimentación:

Docente comenta los cuadros y reflexiones, resaltando buenas prácticas.

Transferencia:

Se anticipa la presentación final como oportunidad para mostrar el trabajo y recibir retroalimentación.

Tarea o reto:

Preparar una presentación breve sobre los aprendizajes y mejoras realizadas.

Sesión 6: Presentación Final y Reflexión del Proyecto de POO

Fase de Inicio

Tiempo estimado: 15 minutos

Propósito de la sesión:

Preparar a los estudiantes para presentar su proyecto y reflexionar sobre el proceso de aprendizaje.

Activación de conocimientos previos:

- **Docente:** Solicita a cada grupo compartir brevemente lo que consideran el logro más importante de su proyecto.
- **Estudiantes:** Compartir ideas y expectativas para la presentación.

Motivación y enganche:

Se enfatiza la importancia de comunicar resultados y recibir retroalimentación constructiva.

Contextualización:

Se conecta esta presentación con habilidades profesionales de comunicación en ingeniería.

Fase de Desarrollo

Tiempo estimado: 90 minutos

- **Actividad 1: Presentación del proyecto**
  - Objetivo: Comunicar de manera clara y estructurada el proyecto desarrollado.
  - Instrucciones:
    - Cada grupo presenta su proyecto (máximo 15 minutos) mostrando diseño, código, pruebas y documentación.
    - Se responde a preguntas del docente y compañeros.
  - Organización: Grupos en plenaria
  - Producto: Presentación oral y soporte visual (diapositivas, código, documentos).
  - Tiempo estimado: 70 minutos (10 minutos por grupo aprox.)
  - Rol docente: Moderar, evaluar y facilitar preguntas.
- **Actividad 2: Evaluación y retroalimentación colectiva**
  - Objetivo: Reflexionar sobre fortalezas y áreas de mejora.
  - Instrucciones:
    - Tras cada presentación, docentes y estudiantes ofrecen retroalimentación constructiva basada en criterios previamente definidos.
  - Organización: Plenaria
  - Producto: Lista de retroalimentación para cada grupo.
  - Tiempo estimado: 20 minutos
  - Rol docente: Guiar retroalimentación, asegurar clima respetuoso y constructivo.

Diferenciación:

Se ofrece a estudiantes con ansiedad o dificultad para hablar en público la opción de preparar videos o presentaciones escritas.

Transiciones:

Se invita a reflexionar sobre cómo aplicar lo aprendido en cursos futuros y en la profesión.

Fase de Cierre

Tiempo estimado: 15 minutos

Síntesis:

- Completar un ticket de salida con 3 aprendizajes claves y 2 preguntas abiertas sobre POO y trabajo en equipo.

Reflexión metacognitiva:

- ¿Qué competencias técnicas y blandas desarrollaste en este proyecto?
- ¿Cómo cambiarías tu enfoque en un futuro proyecto de POO?
- ¿Qué aspectos del proyecto consideras más relevantes para tu formación profesional?

Retroalimentación:

Docente realiza cierre general, reconoce logros y orienta sobre próximos pasos académicos.

Transferencia:

Se conecta el proyecto realizado con la importancia de la POO en la industria y el aprendizaje continuo.

Tarea o reto:

Invitación voluntaria a continuar desarrollando el proyecto o crear uno nuevo aplicando los conceptos aprendidos.

Evaluación

Tipo de evaluación:

- **Diagnóstica:** Sesión 1, activación de conocimientos previos y discusión inicial para conocer ideas previas sobre POO.
- **Formativa:** Durante todas las sesiones en actividades prácticas, revisiones de código, debates y retroalimentaciones constantes.
- **Sumativa:** Sesión 6, evaluación final del proyecto presentado y la calidad de la documentación y pruebas realizadas.

Criterios de evaluación:

- Comprensión y aplicación correcta de conceptos básicos de POO (clases, objetos, herencia, polimorfismo, encapsulamiento).
- Capacidad para diseñar y desarrollar un proyecto funcional y coherente orientado a objetos.
- Calidad y claridad de la documentación técnica y presentación del proyecto.
- Trabajo colaborativo efectivo y comunicación dentro del equipo.

- Implementación de pruebas y habilidades de depuración para mejorar el código.

Instrumentos sugeridos:

- Rúbrica detallada para evaluar código, documentación y presentación.
- Lista de cotejo para seguimiento de actividades y trabajo en equipo.
- Observación directa durante actividades prácticas y debates.
- Portafolio digital con evidencias de código, diagramas, documentación y pruebas.
- Autoevaluación y coevaluación para reflexión personal y grupal.

Evidencias de aprendizaje:

- Código fuente desarrollado y funcional con implementación de POO.
- Diagramas UML y documentación técnica generados durante el proyecto.
- Resultados de pruebas unitarias y registros de depuración.
- Presentación final y retroalimentación recibida.
- Reflexiones escritas y autoevaluaciones de los estudiantes.

Generado con EdutekaLab — edutekalab.co