GUÍA Nº 9 – Patrones de Diseño

FACULTAD	CURSO	AMBIENTE
INGENIERÍA	DISEÑO Y ARQUITECTURA DE SOFTWARE	LABORATORIO DE DISEÑO Y ARQUITECTURA DE SOFTWARE 77C0206

ELABORADO POR	GONZALO GARCIA MARTINEZ	**APROBADO POR**	
VERSIÓN	001	**FECHA DE APROBACIÓN**	

1. LOGRO GENERAL DE UNIDAD DE APRENDIZAJE

- Identificar y comprender los elementos fundamentales del diseño de la arquitectura de software.
- Aplicar diferentes mecanismos de diseño para abordar requisitos y restricciones específicas del sistema.
- Utilizar patrones de diseño comunes para resolver problemas recurrentes en el diseño de software.

2. OBJETIVOS ESPECÍFICOS DE LA PRÁCTICA

- Habilidades y herramientas necesarias para diseñar arquitecturas de software robustas, flexibles y fáciles de mantener, utilizando principios y prácticas de diseño efectivas.

3. MATERIALES Y EQUIPOS

- Computadoras Personales.
- Sistema Operativo Windows.
- Pizarra
- Plumón
- Mota

4. PAUTAS DE SEGURIDAD

Las computadoras y laptops deben de estar prendidas mientras se usan. Pero al terminar el laboratorio estas deben dejarse apagadas.

- En el laboratorio debe estar prendido el aire acondicionado para evitar sobrecalentamientos y averías, especialmente en épocas de altas temperaturas.
- Los estudiantes no pueden llevar alimentos que puedan derramar sobre los computadores.

- Computadoras, router, switch, puntos de acceso (caídas).
- Eléctricos, por contacto directo o indirecto, electricidad estática y por fenómeno térmico. Puede producir: electrocuciones y quemaduras.
- Procedimiento ante Corte de Energía Eléctrica
- No tocar el equipo eléctrico en el que se encuentra trabajando, puede que retorne la energía.
- Comunicarse con el Asistente de Operaciones de turno quien se comunicará con el Técnico.

5. FUNDAMENTO

La asignatura de Diseño y Arquitectura de Software es de carácter teórico-práctico y tiene el propósito de potenciar en el estudiante sus habilidades para analizar y diseñar una arquitectura de software. Se desarrolla los siguientes contenidos: Introducción a la arquitectura de software, vistas y estilos de la arquitectura, requisitos de calidad de un software, diagramación UML orientada al diseño arquitectónico de software, patrones de arquitectura, arquitectura orientada a servicios (SOA), Arquitecturas en Cloud Computing, Arquitecturas para software en dispositivos móviles y documentación de una arquitectura de software.

6. INTRODUCCIÓN (MARCO TEÓRICO)

Bienvenidos a la sesión dedicada a los Patrones de Diseño, un conjunto de soluciones probadas para problemas comunes en el diseño de software. En esta sesión exploraremos los diferentes tipos de patrones de diseño, incluyendo los patrones de creación, estructura y comportamiento, y cómo se aplican en la práctica para mejorar la calidad, la flexibilidad y la mantenibilidad del software.

El objetivo de esta sesión es introducir a los estudiantes en el mundo de los Patrones de Diseño, proporcionándoles una comprensión sólida de los conceptos fundamentales y cómo se aplican en el diseño de software. Al finalizar la sesión, los estudiantes estarán familiarizados con varios patrones de diseño y serán capaces de identificar y aplicar los patrones adecuados para resolver problemas específicos en sus proyectos.

Contenido:
Introducción a los Patrones de Diseño
Patrones de Creación: Singleton, Factory Method, Abstract Factory, Builder, Prototype.
Patrones de Estructura: Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy.
Patrones de Comportamiento: Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor.

7. PROCEDIMIENTO (DESARROLLO DE LA PRÁCTICA)

7.1. Cuestionario

- ¿Qué son los Patrones de Creación en el contexto del diseño de software?
- Enumera al menos tres ejemplos de Patrones de Creación y describe brevemente cómo se aplican.
- ¿Cuál es la diferencia entre el patrón Singleton y el patrón Factory Method?
- ¿Por qué es importante utilizar el patrón Abstract Factory en lugar de crear objetos directamente utilizando el constructor?

- ¿Cuál es el propósito del patrón Builder y en qué situaciones se utiliza típicamente?
- ¿Qué ventajas ofrece el patrón Prototype en comparación con la creación tradicional de objetos?
- Explica el concepto de "clase Singleton" y por qué puede ser útil en ciertos contextos.
- ¿Cuáles son las principales características del patrón Factory Method?
- ¿Puedes dar un ejemplo de una situación en la que sería apropiado utilizar el patrón Prototype?
- ¿Qué problemas resuelve el patrón Builder y cómo lo hace?

7.2. Ejercicios

Patrón Singleton:

- Implementa un Singleton para una clase Logger que registra mensajes de registro para una aplicación.
- Crea una clase DatabaseConnection que siga el patrón Singleton para gestionar las conexiones a la base de datos en una aplicación Java.

Patrón Factory Method:

- Diseña una interfaz Shape con métodos como draw() y area(). Implementa las clases Circle, Square y Triangle que implementen esta interfaz y proporciona un método de fábrica en una clase ShapeFactory para crear instancias de objetos Shape según el tipo de forma solicitado.
- Implementa un Factory Method para crear instancias de objetos Employee en una aplicación de gestión de recursos humanos.

Patrón Abstract Factory:

- Diseña una interfaz AbstractFactory con métodos para crear instancias de objetos Button y Checkbox. Implementa las clases WindowsFactory y MacFactory que implementen esta interfaz para crear botones y casillas de verificación específicos para cada plataforma.
- Crea un Abstract Factory para la creación de componentes de interfaz de usuario (UI) como botones, campos de texto y menús en una aplicación Java.

Patrón Builder:

- Diseña una clase Pizza con varios atributos como size, crust, y toppings. Implementa un Builder para construir diferentes tipos de pizzas con diferentes combinaciones de atributos.
- Crea un Builder para construir objetos User con atributos como name, email, y age en una aplicación de gestión de usuarios.

Patrón Prototype:

- Diseña una clase Shape con métodos para clonar formas geométricas. Implementa clases como Circle, Square y Triangle que implementen este patrón para crear copias de formas existentes.
- Implementa un prototipo para clonar objetos Employee en una aplicación de gestión de recursos humanos.

8. ENTREGABLES

- Cuestionario resuelto, y ejercicios de los patrones en Java.

9. FUENTES DE INFORMACIÓN COMPLEMENTARIA

- "Design Patterns: Elements of Reusable Object-Oriented Software" by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.
- "Head First Design Patterns" by Eric Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra.
- "Patterns of Enterprise Application Architecture" by Martin Fowler.
- "Refactoring: Improving the Design of Existing Code" by Martin Fowler.
- "Clean Code: A Handbook of Agile Software Craftsmanship" by Robert C. Martin.