Laboratorio 5: Repaso para el Examen Final

Table of Contents

1. Selección Única

1. Una plataforma de aprendizaje en línea quiere permitir que los estudiantes trabajen en grupos donde cada grupo contiene estudiantes, tareas y subgrupos. ¿Qué patrón usaría para manejar esta estructura jerárquica?
    1. Decorator
    2. Composite
    3. Bridge
    4. Mediator

    Respuesta: b. Composite

2. Un sistema de mensajería interna debe permitir cambiar la forma de envío de notificaciones (correo, SMS, push) sin modificar el código principal del sistema. ¿Qué patrón aplicaría?
    1. Adapter
    2. Chain of Responsibility
    3. Strategy
    4. Proxy

    Respuesta: c. Strategy

3. En un entorno de trabajo colaborativo, múltiples usuarios editan el mismo documento. El sistema necesita permitir “deshacer” y “rehacer” los cambios. ¿Qué patrón de diseño aplica mejor?
    1. Command
    2. Memento
    3. Observer
    4. State

    Respuesta: b. Memento

4. En una arquitectura basada en eventos, ¿qué atributo de calidad se mejora al desacoplar emisores y receptores de mensajes?
    1. Seguridad
    2. Mantenibilidad
    3. Disponibilidad
    4. Usabilidad

    Respuesta: b. Mantenibilidad

5. El patrón Flyweight mejora principalmente:
    1. Usabilidad
    2. Escalabilidad horizontal
    3. Eficiencia en el uso de memoria
    4. Portabilidad del sistema

    Respuesta: c. Eficiencia en el uso de memoria

6. El siguiente código rompe un principio SOLID. ¿Cuál?

```
class ReportGenerator {
    public void generateReport(String data) {
        System.out.println("Generando reporte: " + data);
    }
    public void sendEmail(String data) {
        System.out.println("Enviando por correo: " + data);
    }
}
```

    1. SRP
    2. LSP
    3. DIP
    4. OCP

    Respuesta: a. Single Responsibility Principle (SRP)

7. El siguiente código es un ejemplo de mal diseño, ¿por qué?

```
class PaymentProcessor:
    def process(self, card_type):
        if card_type == "VISA":
            print("Procesando VISA")
        elif card_type == "MasterCard":
            print("Procesando MasterCard")
        else:
```

```
            print("Tipo no soportado")
```

1. Rompe el Principio de OCP
2. Rompe el Principio de SRP
3. Afecta la eficiencia del sistema
4. Afecta la seguridad del sistema

   Respuesta: a. Rompe el Open/Closed Principle (OCP)

8. ¿Qué atributo de calidad se vería afectado negativamente si un sistema tiene muchas dependencias circulares?
   1. Mantenibilidad
   2. Rendimiento
   3. Seguridad
   4. Escalabilidad

   Respuesta: a. Mantenibilidad

9. El siguiente código rompe la Inversión de Dependencias (DIP) porque:

```
class EmailService {
    public void send(String msg) {
        System.out.println("Email enviado");
    }
}
class Notification {
    private EmailService email = new EmailService();
    public void notify(String msg) { email.send(msg);}
}
```

   1. Notification depende de una abstracción
   2. EmailService implementa una interfaz
   3. Notification depende de una clase concreta
   4. EmailService es un Singleton

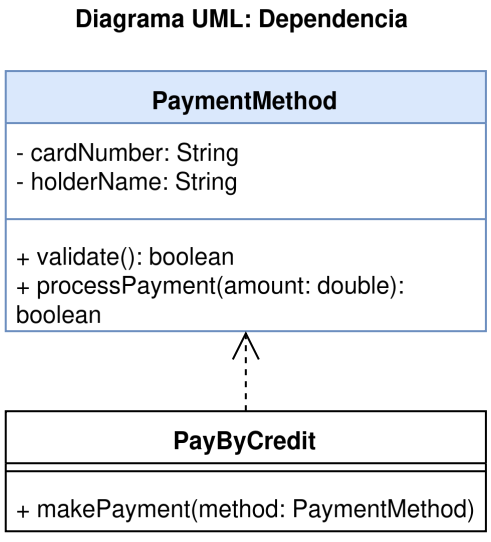
   Respuesta: c. Notification depende de una clase concreta

10. ¿Cuál es el principal tradeoff al aumentar la disponibilidad mediante redundancia?
    1. Reduce la cohesión del sistema
    2. Aumenta la latencia
    3. Aumenta costos de mantenimiento
    4. Reduce la interoperabilidad

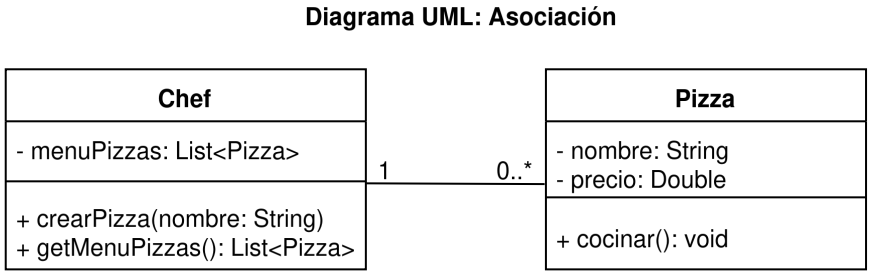
Respuesta: c. Aumenta costos de mantenimiento

2. Desarrollo

1. ¿Qué significa que un sistema tenga alta cohesión y bajo acoplamiento? (5%)
   - **Alta cohesión:** Cuando los elementos dentro de un módulo o clase están estrechamente relacionados y trbajan juntos para cumplir una única responsabilidad o función. Esto facilita la **comprensión**, el **mantenimiento** y la **reutilización** del código.
   - **Bajo acoplamiento:** Cuando los módulos o clases tienen pocas dependencias entre sí, lo que permite que los cambios en un módulo no afecten significativamente a otros. Esto mejora la **flexibilidad** y la **mantenibilidad** del sistema.
   - Juntos, estos principios contribuyen a un diseño de software más robusto, fácil de entender y extender, flexible, mantenible y reutilizable.
2. ¿La siguiente historia cumple con INVEST? Justifique cada letra de INVEST. (5%) Como estudiante, quiero acceder a mis notas para poder verificar mi progreso en los cursos.
   - **I (Independent):** Puede ser desarrollada y entregada sin depender de otras historias.
   - **N (Negotiable):** Es negociable porque los detalles específicos sobre cómo se accede a las notas pueden discutirse y ajustarse según las necesidades del usuario.
   - **V (Valuable):** Proporciona valor al usuario al permitirle verificar su progreso.
   - **E (Estimable):** Es estimable porque se puede evaluar los recursos y el tiempo necesarios para implementarla.
   - **S (Small):** Es lo suficientemente pequeña para ser completada en una historia de usuario.
   - **T (Testable):** Es testeable porque se pueden definir criterios de verificación de que el usuario puede acceder a sus notas correctamente.
3. En UML, explique la diferencia entre dependencia y asociación. Dibuje un ejemplo para cada relación. (10%)
   1. **Dependencia:** Es una relación donde un cambio en una clase puede afectar a otra clase que *depende* de ella.



2. **Asociación:** Es una relación donde una clase tiene una *referencia* a otra clase. Puede ser bidireccional o unidirecciona.



Pregunta 4 y 5:

Una empresa de transporte público desea desarrollar un sistema para calcular rutas y tarifas. El sistema debe cumplir con los siguientes requisitos:

- Existen distintos medios de transporte (bus, tren, ferry).
- Cada medio calcula su tarifa y tiempo estimado de forma diferente.
- El sistema debe permitir añadir nuevos medios de transporte sin afectar las clases existentes.
- Los cálculos deben poder cambiarse dinámicamente (por ejemplo, según hora pico o clima).
- Existen reglas de negocio dinámicas que pueden otorgar distintos tipos de descuentos o tarifas a pagar. Por ejemplo: descuentos por estudiante, tarifas extra por hora pico, por circular cuando la placa cuenta con restricción, descuentos verdes (por transporte eléctrico) y convenios especiales con municipalidades.
- A futuro, se quiere integrar un algoritmo de optimización de rutas basado en IA que pueda inyectarse sin afectar la arquitectura actual.

Con base en lo anterior, responda:

1. ¿Qué patrón de diseño usaría para representar los diferentes medios de transporte y sus cálculos? (10%) Patrón: Strategy Explicación: Este algoritmo permite definir un conjunto de algoritmos (calculo de tarifa y tiempo), los encapsula y los hace intercambiables. Cada medio de transporte puede implementar su propia estrategia de cálculo, permitiendo añadir nuevos medios de transporte sin modificar las clases existentes.
2. ¿Qué patrón de diseño usaría para permitir la integración de algún servicio de IA posteriormente? (10%) Patrón: Decorator Explicación: Este patrón permite añadir responsabilidades adicionales a un objeto de forma dinámica. Se puede crear un decorador que envuelva el cálculo de rutas existente y añada la funcionalidad de optimización basada en IA sin modificar el código original.

3. Código

Realice un POC (proof of concept) en código para explicar cómo implementaría los posibles distintos descuentos y tarifas que cambian dinámicamente.

Un ejemplo sobre como puede verse el resultado de su código es:

```
Calculando viaje combinado: Bus + Tren + Bicicleta
Tarifa base total: 2450 colones
Aplicando descuento de estudiante (-15%)
Aplicando descuento verde (-5%)
Tarifa final: 1970 colones
```

Respuesta:

- Patrón seleccionado: Decorator
- Explicación: Utilizaría el patrón decorator para aplicar los distintos descuentos y tarifas de forma dinámica.

Código completo en: Repositorio de Diseño de Software - II Semestre 2025

POC en Python:

Se establece un decorador base `tarifaDecorator` y luego se crean decoradores específicos para cada tipo de descuento, como `studentDecorator` y `greenDecorator`. Estos decoradores envuelven una instancia de `tarifa` y modifican el cálculo de la tarifa según las reglas de negocio.

```
class tarifa:
    def calcular_tarifa(self):
        pass

class tarifaDecorator(tarifa):
    def __init__(self, tarifa_decorada):
        self._tarifa_decorada = tarifa_decorada

    def calcular_tarifa(self):
        return self._tarifa_decorada.calcular_tarifa()

class basicTarifa(tarifa):
    def __init__(self, tarifa_base):
        self._tarifa_base = tarifa_base

    def calcular_tarifa(self):
        return self._tarifa_base

class studentDecorator(tarifaDecorator):
    def __init__(self, tarifa_decorada):
        super().__init__(tarifa_decorada)

    def calcular_tarifa(self):
        tarifa_original = self._tarifa_decorada.calcular_tarifa()
        descuento_estudiante = tarifa_original * 0.15
        return tarifa_original - descuento_estudiante

class greenDecorator(tarifaDecorator):
    def __init__(self, tarifa_decorada):
```

```
        super().__init__(tarifa_decorada)

    def calcular_tarifa(self):
        tarifa_original = self._tarifa_decorada.calcular_tarifa()
        descuento_verde = tarifa_original * 0.05
        return tarifa_original - descuento_verde
```

Ejemplo de uso:

```
def main():
    print("=== Simulación de sistema de transporte ===\n")

    # Tarifa base
    tarifa_base = basicTarifa(2450)
    print(f"Tarifa base total: {tarifa_base.calcular_tarifa()} colones\n")

    # Aplicar decoradores
    tarifa_estudiante = studentDecorator(tarifa_base)
    print("Aplicando descuento de estudiante (-15%)")

    tarifa_final = greenDecorator(tarifa_estudiante)
    print("Aplicando descuento verde (-5%)")

    # Resultado final (redondeando a entero como en ejemplo)
    print(f"Tarifa final total: {tarifa_final.calcular_tarifa():.0f} colones\n")
```