**Examen Primer Corte**  
  
Jarvic Balanta Santacruz  
Universidad Cooperativa de Colombia  
Facultad de Ingeniería  
Patrones de Diseño Orientado a Objetos

# Resumen

Este examen evalúa el conocimiento de los patrones de diseño creacionales en la resolución de problemas del mundo real. Se incluyen preguntas conceptuales, identificación de patrones en historias de usuario y ejercicios de implementación en Java. A través de estos ejercicios, se busca aplicar los principios de diseño orientado a objetos para mejorar la modularidad y escalabilidad del código.

# Introducción

Los patrones de diseño creacionales proporcionan soluciones estructuradas para la creación eficiente de objetos. En este examen, se analizan patrones como **Factory Method, Abstract Factory, Builder y Prototype**, aplicándolos a escenarios del sector financiero, aeroportuario y de seguros. El objetivo es evaluar la capacidad se nosotros como estudiantes para identificar y aplicar estos patrones correctamente.

# Resultados

## Parte 1: Preguntas Conceptuales

1. **¿Cuál es la diferencia entre Factory Method y Abstract Factory?**

El patrón **Abstract** **Factory** proporciona una interfaz para crear familias de objetos relacionados sin especificar sus clases concretas, permitiendo generar múltiples objetos diseñados para trabajar juntos. En cambio, el patrón **Factory** **Method** define una interfaz para la creación de un único objeto, delegando la responsabilidad de instanciación a las subclases, que deciden qué clase concreta crear

1. **Explique en qué situaciones es recomendable usar el patrón Builder y por qué.**

El patrón **Builder** es recomendable cuando se necesita construir objetos complejos con múltiples configuraciones. Esto es útil en escenarios como la creación de documentos XML o JSON, donde la estructura puede variar, o en la construcción de una pizza con diferentes ingredientes.

1. **¿Cuál es la principal ventaja del patrón Prototype y en qué casos es más útil?**

La principal ventaja del patrón **Prototype** es que permite crear nuevos objetos de manera eficiente copiando o clonando un prototipo existente en lugar de instanciarlos desde cero. Es especialmente útil en casos donde la creación de objetos es costosa y requiere mucho tiempop.

1. **¿Por qué el patrón Singleton puede ser problemático en aplicaciones multihilo y cómo se puede solucionar?**

El patrón **Singleton** puede ser problemático en aplicaciones multihilo porque, si varios hilos intentan acceder simultáneamente a la instancia única, pueden crear múltiples instancias en lugar de una sola. Una solución puede ser el bloqueo sincronizado en el método de obtención de la instancia.

1. **Mencione un ejemplo de la vida real donde se puedan combinar Abstract Factory y Builder, y explique cómo se complementan.**

En la fabricación de autos, los patrones **Abstract** **Factory** y **Builder** se pueden complementar. **Abstract** **Factory** se encarga de crear familias de componentes, como motores, chasis, asegurando que cada tipo de automóvil sea eléctrico o gasolina tenga los elementos adecuados. Luego, el patrón **Builder** permite ensamblar el vehículo con configuraciones personalizadas, como asientos de cuero, sistema de sonido.

## Parte 2: Identificación del Patrón

**fHistoria 1**: Como gerente de una compañía de seguros, quiero permitir a los agentes crear pólizas de seguro personalizadas según el tipo de cliente (automóvil, hogar, vida), de manera que puedan gestionar las características específicas de cada póliza sin necesidad de cambiar el código de la aplicación.

**¿Cuál Patrón recomendarías y por qué?**

* *En este caso es ideal usar le patrón* ***Abstract******Factory*** *porque puede proporcionar una interfaz la cual puede crear familias de objetos. Los agentes pueden generar pólizas de manera flexible sin modificar el código de la aplicación.*

**Historia 2:** Como administrador de un sistema de reservas de hoteles, quiero que los usuarios puedan elegir entre diferentes configuraciones de habitaciones (lujo, estándar, económica) con opciones adicionales como desayuno, acceso al gimnasio y estacionamiento, para que la experiencia del cliente sea personalizada.

¿Cuál Patrón recomendarías y por qué?

* *Si se quiere que los usuarios puedan elegir combinaciones. Se debe utilizar el patrón* ***Builder****. Dado que necesitamos que los usuarios puedan personalizar su habitación eligiendo diferentes configuraciones y opciones adicionales. Entonces, sistema deberá ofrecer flexibilidad y modularidad, y así mejorar la experiencia del usuario.*

## Parte 3: Implementación en Java

* **Historia de Usuario 1: Sector Financiero.**

Como desarrollador de un banco digital, necesito un sistema que permita a los clientes solicitar diferentes tipos de préstamos (hipotecario, automotriz, personal) con diferentes tasas de interés y plazos. El sistema debe garantizar que las reglas de negocio sean aplicadas correctamente según el tipo de préstamo seleccionado.

* **¿Qué patrón aplicaría?**

El patron Factory Method.

**Resuelva la Implementación:**

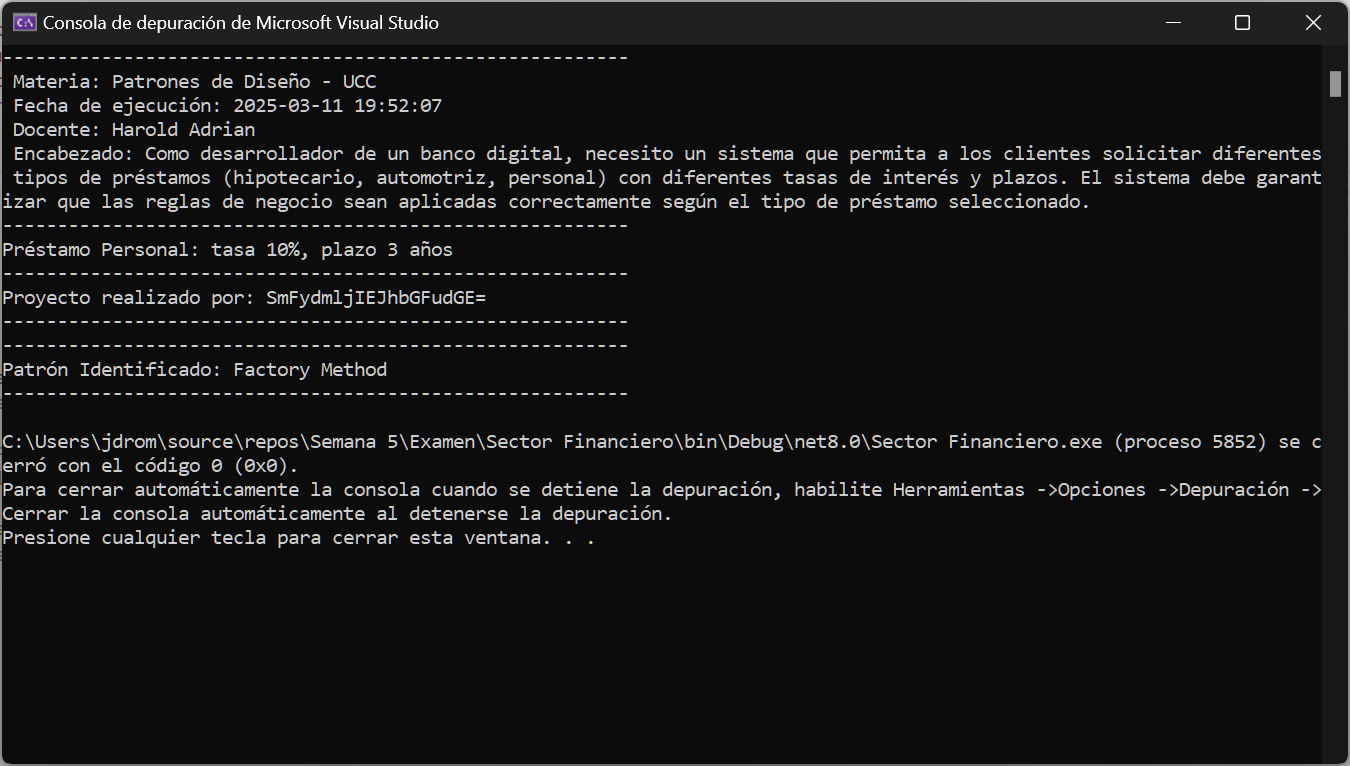
****

Ilustración 1. Historia de Usuario 1: Sector Financiero.

Repositorio: https://github.com/Jarvicf/ucc.patrones.dise-o.objetos/tree/e02d000ea7ed41f80d1ea159818d0a9c6ceed045/Semana%205%20-%20Examen/Examen/Sector%20Financiero

* **Historia de Usuario 2: Sector Aeroportuario.**

Como ingeniero de software en una aerolínea, necesito un sistema de gestión de reservas de vuelos que permita a los clientes personalizar su boleto eligiendo la clase del vuelo (económica, ejecutiva, primera clase) y los servicios adicionales (equipaje extra, selección de asiento, comida especial). El sistema debe facilitar la creación de boletos personalizados sin alterar la lógica principal.

* **¿Qué patrón aplicaría?**

El patrón Builder.

**Resuelva la Implementación:**

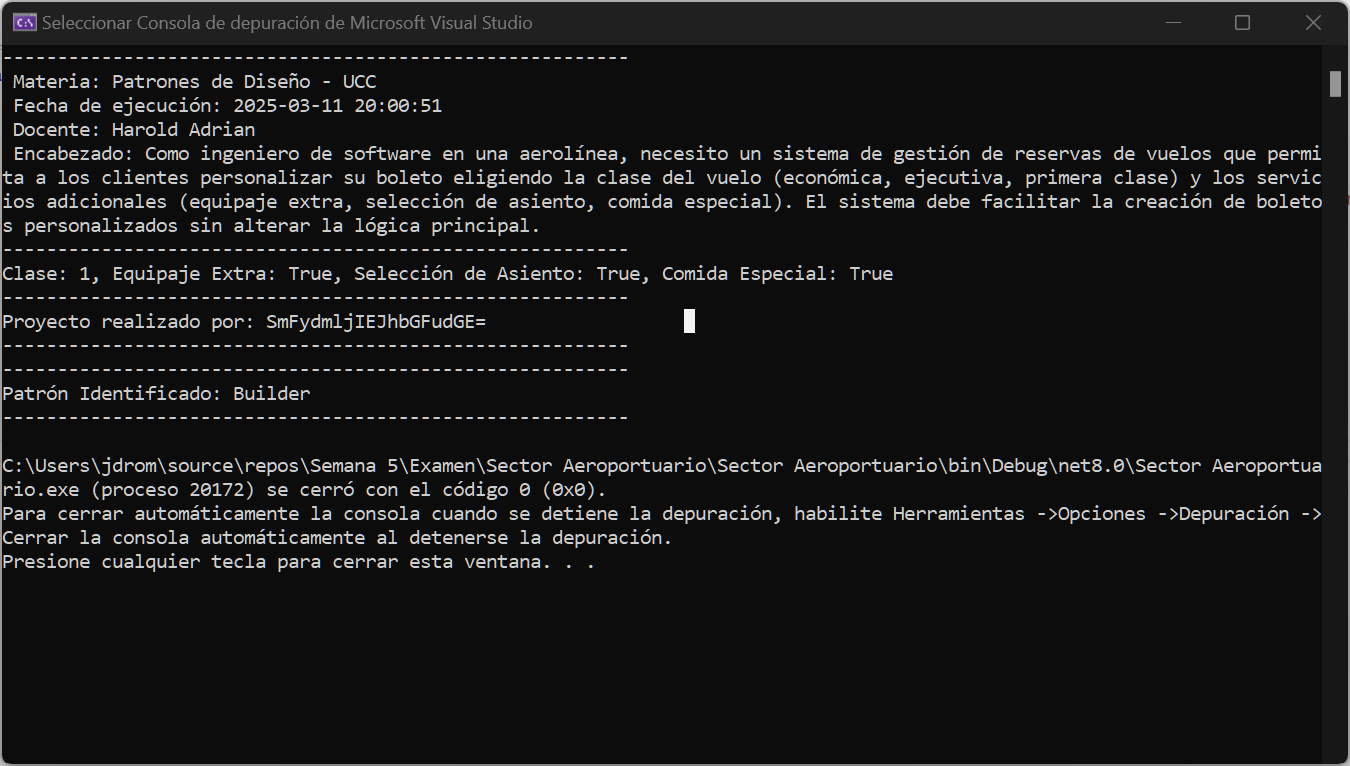
****

Ilustración 2.Historia de Usuario 2: Sector Aeroportuario.

Repositorio.:

https://github.com/Jarvicf/ucc.patrones.dise-o.objetos/tree/e02d000ea7ed41f80d1ea159818d0a9c6ceed045/Semana%205%20-%20Examen/Examen/Sector%20Aeroportuario/Sector%20Aeroportuario

# Conclusiones

Los patrones de diseño creacionales brindan soluciones estructuradas para la creación de objetos, mejorando la modularidad y reutilización del código.

* Factory Method y Abstract Factory permiten crear objetos de manera flexible sin acoplamiento directo.
* Builder simplifica la construcción de objetos con múltiples configuraciones.
* Prototype facilita la clonación eficiente de objetos, evitando la sobrecarga de instanciación.

Aplicar correctamente estos patrones en el desarrollo de software mejora la escalabilidad y mantenibilidad de los sistemas.

# Referencias

Bolaños Rodríguez, H. A. (2025). Curso Patrones de Diseño OO - Semana 6. Universidad Cooperativa de Colombia.