SOFTWARE PARA ADMINISTRACIÓN DE ALQUILER DE VESTUARIO

Etapa de Transferencia

Nataly Quintana Bernal

Ingeniería de Software

Mat. Patrones de Diseño

Fundación Universitaria Compensar

Introducción

Tomando como base los parámetros brindados para la ejecución de la etapa de transferencia y dando solución a la actividad propuesta, en este informe se continuará con la revisión de diagrama de clases y diagrama secuencial, así como la identificación de patrones de comportamiento que permitan dar una mejora al proyecto previo, y se realizará la implementación de la aplicación en Java utilizando NetBeans, verificando su correcto funcionamiento mediante pruebas y generando la documentación de las mismas.

Caso de estudio:

"Los Atuendos" es un negocio de alquiler de vestidos para dama, caballero y disfraces. De cada prenda que el negocio tiene para alquiler se lleva un registro con esta información: referencia, color, marca, talla, valor del alquiler. Cuando se trata de un vestido para dama, se tiene además la indicación de si tiene pedrería, si es largo o corto y la cantidad de piezas que lo componen. Si es un traje para caballero se discrimina el tipo: convencional, frac, sacoleva, otro; además se indica si el traje lleva corbata, corbatín o plastrón. Por el lado de los disfraces, cada uno tiene un nombre. Cuando una persona acude al negocio para solicitar un servicio, se toma su número de identificación, nombre, dirección, teléfono y correo electrónico. Una persona puede alquilar la cantidad de trajes o vestidos que necesite. Cada servicio de alquiler queda registrado con número, fecha de solicitud, fecha de alquiler, empleado que tomó el pedido, cliente que toma el servicio. Nombre, número de identificación, dirección, teléfono y cargo registrado de cada empleado del negocio. Cuando se devuelven las prendas, se registran en un listado para enviar a lavado, según el orden en que van llegando al negocio; pero cuando la prenda llega manchada, es delicada o el administrador considera necesario, se le da prelación para enviarlo a lavado. Por restricciones logísticas, las prendas se envían a lavado en tandas, según la disponibilidad del día.

Contexto de la implementación inicial

1. **Requisitos identificados**:
- Funcionales: registro de entidades, registro de alquileres, consultas

- No funcionales: base de datos relacional, integridad, manejo de excepciones

2. **Patrones de diseño aplicados**

Creación**:**
- *Factor Method* (para instanciar clientes, empleados, prendas)
- *Singleton* (para conectar con bd)

Estructurales:
- *Adapter* (integración de lavandería)
- *Composite* (prendas individuales y compuestas)
- *Facade* (operaciones del alquiler)

Análisis de patrones de comportamiento

Teniendo en cuenta que estos patrones ayudan a gestionar mejor las interacciones entre objetos, haciendo que el sistema sea más flexible, mantenible y escalable, se analizaron de la siguiente manera:

- **Iterator:** Podría ser útil al requerir recorrer colecciones sin exponer su implementación interna
  - Recorrer la lista de prendas disponibles
  - Navegar por listados de prendas enviadas a la lavanderia

  Su beneficio, se centraria en la separación entre la lógica de recorrido vs la estructura de los datos; generando mas flexibilidad al consultar información

- **Command**: Podría ser útil al encapsular acciones, permitiendo ejecutar, deshacer o almacenar solicitudes
  - Registrar un alquiler

- Registrar una devolución
- Envio de prendas a la lavanderia

Su beneficio, seria que permitiría tener un historial de operaciones y facilitaría la implementación de funcionalidades como deshacer ante algún error

- **Observer**: Podría ser de utilidad, cuando un cambio de objeto deba notificarse a otros interesados
  - Notificar al área de lavandería
  - Notificar ante un nuevo registro

  Su beneficio, seria la integración entre módulos generando flexibilidad en su información

- **Strategy**: Podría ser de utilidad, si se requieren diferentes algoritmos para una misma actividad y se desea algo más dinámico
  - Cálculo de valores
  - Estrategia de priorización

  Su beneficio, seria la flexibilidad para ajustar reglas sin modificar la lógica

Sin embargo, para la implementación y mejora del proyecto se tomaron los siguientes patrones de comportamiento:

Patron a utilizar	¿Por qué contribuye a mejorar el diseño?
Command	Encapsula cada operación (registrar alquiler, enviar a lavandería) como un objeto independiente, lo que permite auditar acciones, implementar un historial de operaciones y agregar fácilmente funciones de deshacer/rehacer.
Observer	Permite que, cuando ocurre un evento (ej. registro de un alquiler), se notifique automáticamente a otros módulos interesados

Diagrama de Clases (con patrones de comportamiento)

Principales componentes

1. Command
   - ICommand > interfaz base con execute ()
   - RegistrarAlquilerCommand,RegistrarDevolucionCommand, EnviarLavanderiaCommand > son comandos concretos que encapsulan operaciones
   - Ivoker > ejecuta el comando que se asigno
   - AlquilerReceiver > es la lógica del proyecto (**ej**. *Cambiar estado, enviar a lavar*)

2. Observer
   - ISujeto > interfaz para objetos (*prendas*)
   - IObservador > interfaz
   - PrendaComponent > implementa ISujeto y notifica cuando el estado cambia
   - AdministradorObservador, EmpleadoObservador > observadores especifico que reaccionan al evento

Importante:

El *Invoker* conoce un ICommand, pero desconoce qué hace internamente; cada Command llama métodos del AlquilerReceiver, el cual interactúa con el modelo (*ServicioAlquiler, PrendaComponent*). Cuando una prenda cambia de estado, el Observer notifica a todos los observadores, como el Administrador y el Empleado.

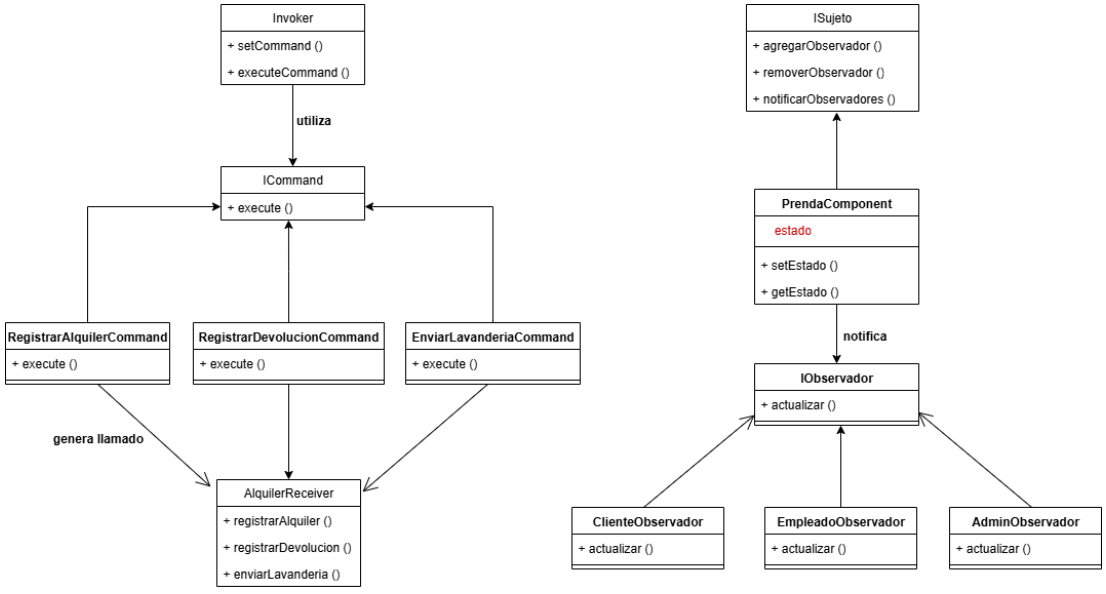
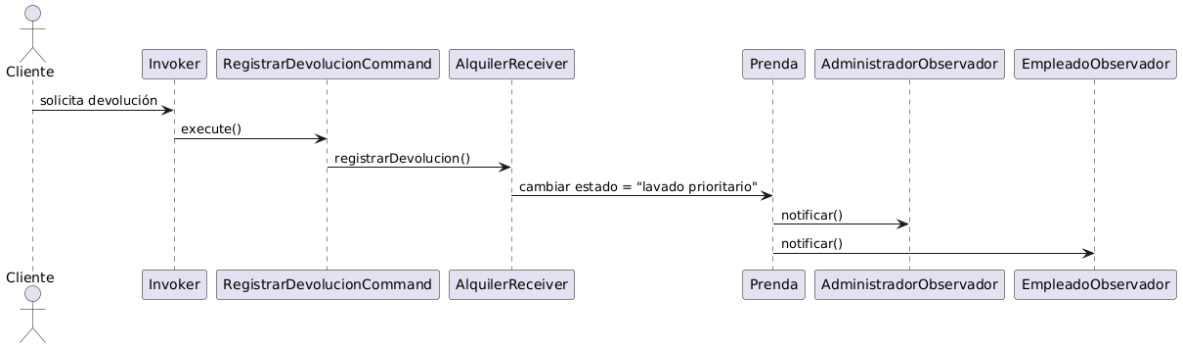


Figura 1. *Diagrama de clases*

Diagramas de Secuencia

Actores y objetos

- Usuario > da clic en Registrar devolución
- AlquilerFacade > recibe la solicitud
- Invoker > ejecuta el comando
- Registrar DevoluciónCommand > almacena la acción
- AlquilerReceiver > realiza la logica y cambia el estado de la prenda
- PrendaComponent > actualiza el estado
- Observadores (Admin/Empleado) > son notificados del cambio



Segundo momento de actividad:

El desarrollo de la mejora del software fue subido a github y se puede acceder a la URL: NatalyQB/Patrones_Dise-o . Adicionalmente fue importado a la carpeta compartida en drive de Ucompensar Patrones de Diseño Software

En la implementación del proyecto se generó la creación de:

- Carpeta com.mycompany.command
  - ICommand: interfaz con el método execute()
  - RegistrarAlquilerCommand: encapsula la lógica de registrar un alquiler en el sistema
  - RegistrarDevolucionCommand: encapsula la lógica de registrar la devolución de una prenda, notificando el estado
  - EnviarLavanderiaCommand: encapsula el envío de prendas a lavandería (*con prioridad si corresponde*)
  - Invoker: clase que recibe un comando y lo ejecuta
  - AlquilerReceiver: contiene la lógica real de negocio, como cambiar estado de prendas o actualizar alquileres.

- Carpeta com.mycompany.observer
  - ISujeto: define operaciones para agregar, eliminar y notificar observadores
  - IObservador: define el método actualizar(*ISujeto sujeto, String evento*)
  - AdministradorObservador: observa prendas y genera órdenes de lavandería prioritarias
  - EmpleadoObservador: observa prendas y genera notificaciones para empleados

Adicionalmente, se genero la modificación de los archivos:

- PrendaComponent.java
- AlquilerFacade.java

Se intento generar la implementación de la interfaz pero al ejecutar el proyecto se generaron errores que no logre solucionar, por ello se creo un archivo MainDemo.java para poder ejecutar en consola y ver el resultado esperado.