Inyección de dependencias
Sistema de Gestión de Finanzas Personales

Frida Cano Falcón

Academia Java MTY
Agosto 2024

Viernes 16 de agosto de 2024

Introducción

La gestión de finanzas personales es una tarea fundamental que permite a los individuos tomar decisiones informadas sobre sus gastos, ingresos y presupuestos. Este proyecto se centra en desarrollar un sistema de gestión de finanzas personales en Java, utilizando el patrón de diseño de Inyección de Dependencias (Dependency Injection, DI) para promover un código modular, fácil de mantener y probar. El sistema incluirá funcionalidades como el registro de transacciones y la generación de un reporte básico.

Estructura del Sistema

El sistema se organiza en tres capas principales:

1. **Capa de Modelo**: Define las entidades básicas como *Transaccion*
2. **Capa de Servicio**:Contiene la lógica de negocio para manejar las transacciones y genera un reporte básico.

Explicación de la Inyección de Dependencias

La Inyección de Dependencias por Constructor es un patrón en el cual las dependencias requeridas por una clase se proporcionan a través de su constructor. Esto significa que cuando se crea una instancia de la clase, se inyectan todas las dependencias necesarias.

En este ejemplo, la clase *TransaccionServicioImpl* recibe una lista de transacciones (*List*<*Transaccion*>) como parámetro de su constructor. Esto:

- Desacopla la lógica de negocio de la implementación específica de la lista, permitiendo cambiar la implementación sin modificar la lógica.
- Facilita las pruebas unitarias, ya que se puede pasar una implementación de prueba de *List*<*Transaccion*> durante las pruebas.
- Promueve la inmutabilidad en la clase *TransaccionServicioImpl*, ya que las dependencias se establecen en el momento de la creación y no se pueden cambiar posteriormente.

Implementación en código

1. Capa de Modelo

La clase *Transaccion* representa una transacción financiera, que incluye una descripción y un monto. Esta es la entidad básica sobre la cual se construye el sistema.

src > com > curso > tarea > Transaccion.java > Transaccion > toString()

```java
package com.curso.tarea;

public class Transaccion {
    private String descripcion;
    private double monto;

    public Transaccion(String descripcion, double monto) {
        this.descripcion = descripcion;
        this.monto = monto;
    }

    @Override
    public String toString() {
        return "Transaccion{" +
                "descripcion='" + descripcion + '\'' +
                ", monto=" + monto +
                '}';
    }
}
```

Código clase Transacción

2. Capa de Servicio

La interfaz TransaccionServicio define los métodos para agregar transacciones y obtener un reporte de las transacciones registradas.

src > com > curso > tarea > TransaccionServicio.java > TransaccionServicio

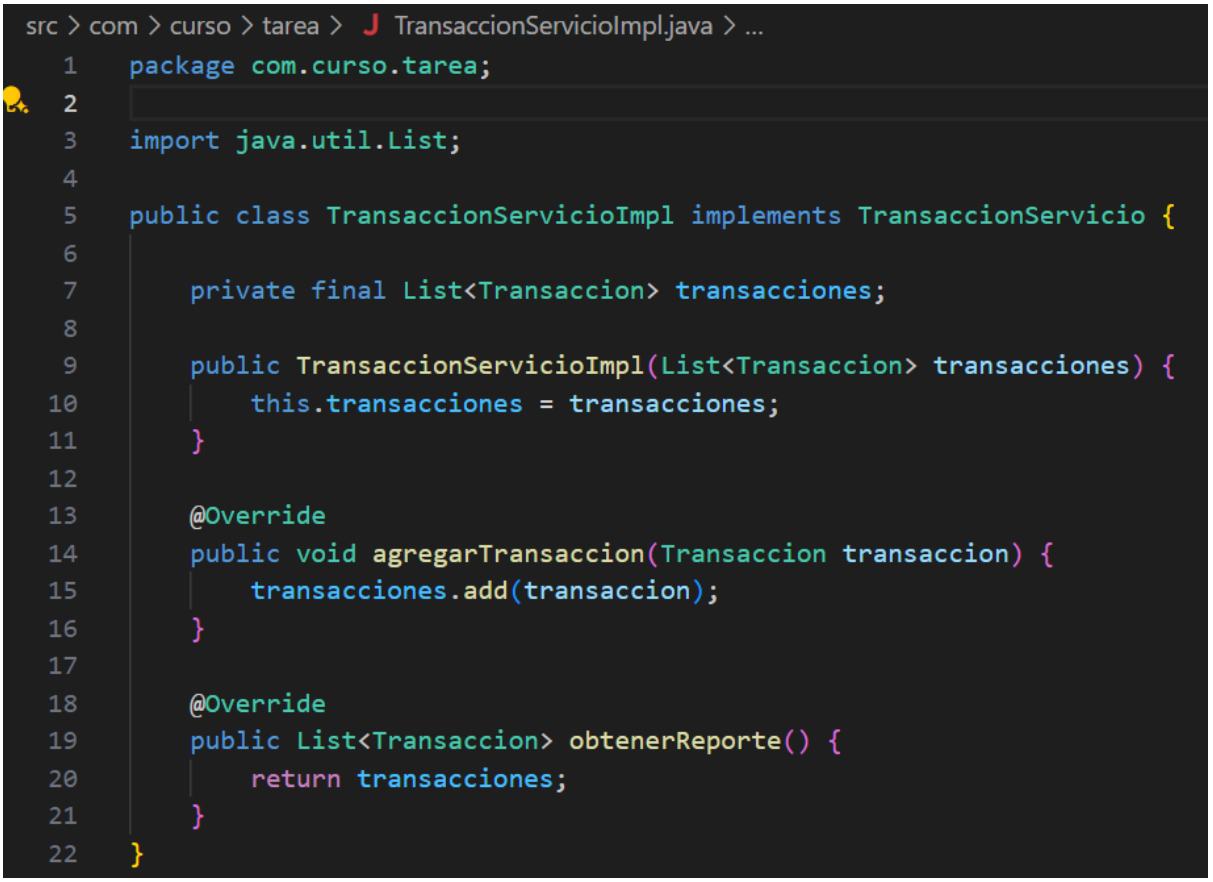
```java
package com.curso.tarea;
import java.util.List;

public interface TransaccionServicio {
    void agregarTransaccion(Transaccion transaccion);
    List<Transaccion> obtenerReporte();
}
```

Código Interfaz TransacciónServicio

En esta implementación, *TransaccionServicioImpl* recibe una lista de transacciones a través de su constructor. Esto es un ejemplo de inyección de dependencias por constructor, donde la lista de transacciones se proporciona en el momento de la creación de la instancia. Esto

desacopla la creación de la dependencia (*List<Transaccion>*) de su uso, lo que mejora la flexibilidad y testabilidad del código.



Código clase TransacciónServicioImpl

3. Capa de Presentación - Principal

En la clase *Principal*, se crea una instancia de *List<Transaccion>* y se pasa como argumento al constructor de *TransaccionServicioImpl*. De esta manera, *Principal* no es responsable de la creación de la dependencia interna (*List<Transaccion>*), sino que delega esta responsabilidad a la clase que implementa el servicio. Este es un ejemplo claro de inyección de dependencias por constructor.

src > com > curso > tarea > Principal.java > Principal > main(String[])

```java
package com.curso.tarea;

import java.util.ArrayList;
import java.util.List;

public class Principal {
    Run | Debug
    public static void main(String[] args) {
        List<Transaccion> transacciones = new ArrayList<>();
        TransaccionServicio servicio = new TransaccionServicioImpl(transacciones);

        Transaccion t1 = new Transaccion(descripcion:"Almuerzo", monto:50.0);
        Transaccion t2 = new Transaccion(descripcion:"Café", monto:10.0);

        servicio.agregarTransaccion(t1);
        servicio.agregarTransaccion(t2);

        servicio.obtenerReporte().forEach(System.out::println);
    }
}
```

Código clase Principal

Resultados de la prueba Principal

```
PS C:\Users\HP\Documents\GitHub\EntregablesJavaAcademy2024\Semana 1\InyeccionDependencias>  c:; cd 'c:\Users\HP\Documents\GitHub\
EntregablesJavaAcademy2024\Semana 1\InyeccionDependencias'; & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsIn
ExceptionMessages' '-cp' 'C:\Users\HP\Documents\GitHub\EntregablesJavaAcademy2024\Semana 1\InyeccionDependencias\bin' 'com.curso.
tarea.Principal'
Transaccion{descripcion='Almuerzo', monto=50.0}
Transaccion{descripcion='Café', monto=10.0}
```

Terminal con los resultados

Conclusión

El uso de la Inyección de Dependencias por Constructor en este sistema de gestión de finanzas personales ha permitido crear un código más limpio y modular. Al inyectar las dependencias en el constructor, se garantiza que todas las clases estén adecuadamente desacopladas, lo que facilita la escalabilidad y el mantenimiento del código. Este enfoque también simplifica las pruebas unitarias, ya que las dependencias pueden ser fácilmente simuladas o reemplazadas durante las pruebas. Este proyecto, aunque simple, sienta las bases para desarrollar sistemas más complejos y robustos en el futuro.