State

Patrones de Diseño

Problema

Se desea desarrollar el software para el cajero automático de un banco.

- El cajero comienza inactivo, esperando que se inserte una tarjeta.
- Una vez que el usuario inserta su tarjeta, solicita al usuario que ingrese su PIN.
- Después de que se ingrese un PIN válido, el cajero permite al usuario retirar dinero o verificar su saldo.
- Manejar errores.

Problema

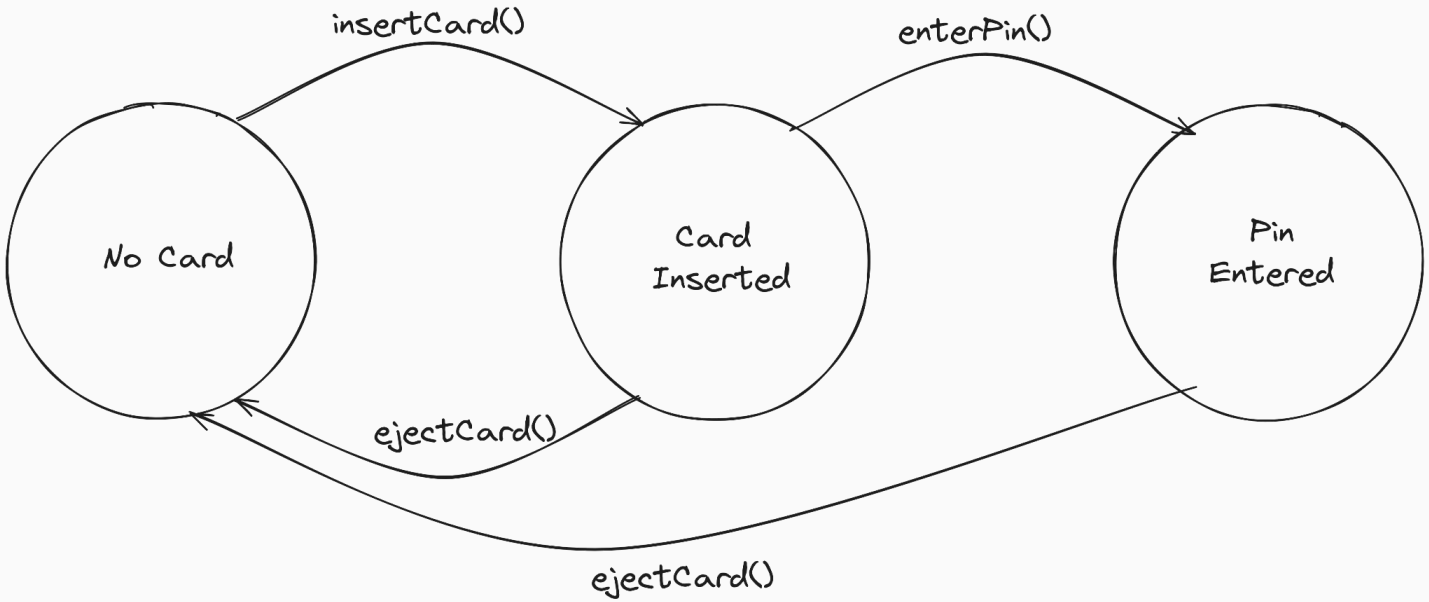
withoutState.java

```java
public void enterPin(int pin) {
    if (!cardInserted) {
      System.out.println("No card inserted.");
    } else if (pinEntered) {
      System.out.println("PIN already entered.");
    } else if (pin == 1234) {
      pinEntered = true;
      System.out.println("PIN accepted. You can now perform transactions.");
    } else {
      System.out.println("Incorrect PIN. Try again.");
    }
  }
```

Problema

Las **acciones** que se pueden realizar o los **errores** que puedan surgir dependen del **estado** del sistema.

Problema


insertCard()
enterPin()
No Card
Card Inserted
Pin Entered
ejectCard()
ejectCard()

Solución: State

SOLID

- Single Responsibility Principle
- Open/Closed Principle

vs Strategy

Son similares en la práctica. Las diferencias son sutiles.

- **Strategy**: Se enfoca en intercambiar diferentes algoritmos o métodos que logran el mismo objetivo.
- **State**: Gestiona diferentes comportamientos según el estado actual del objeto.
- En **State**, los estados casi siempre conocen a los otros estados y pueden iniciar transiciones. En **strategy** casi nunca.

Caso cercano…