State Pattern

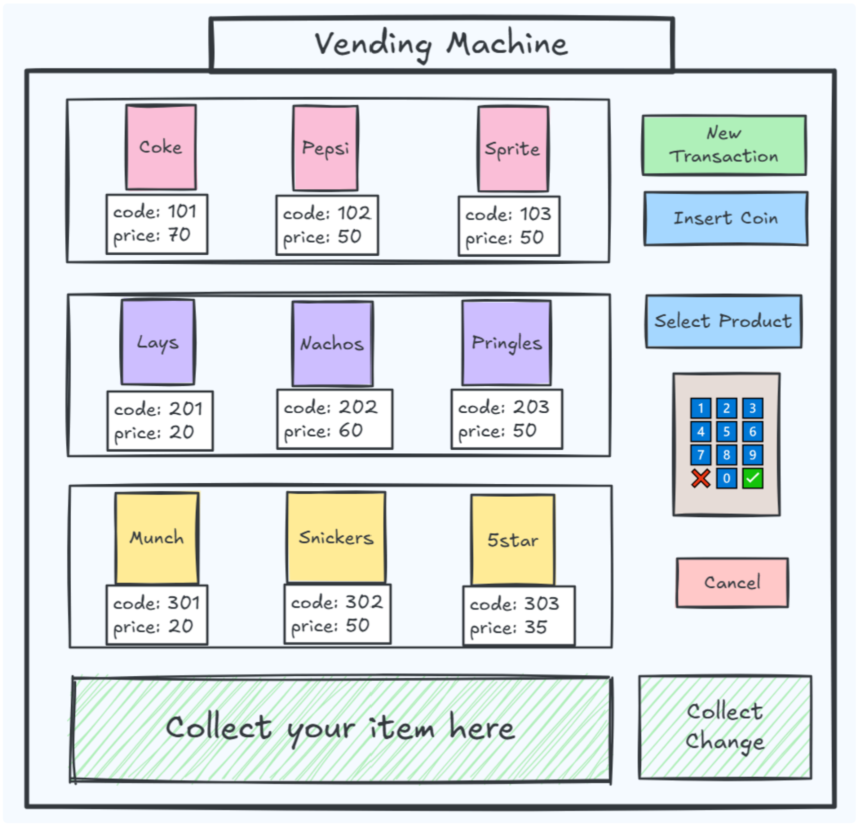
Resources

- Video → 41. All Behavioral Design Patterns | Strategy, Observer, State, Template, Command, Visitor, Memento
- Video → 16. Design Vending Machine (Hindi) | LLD of Vending Machine | State Design Pattern | LLD question

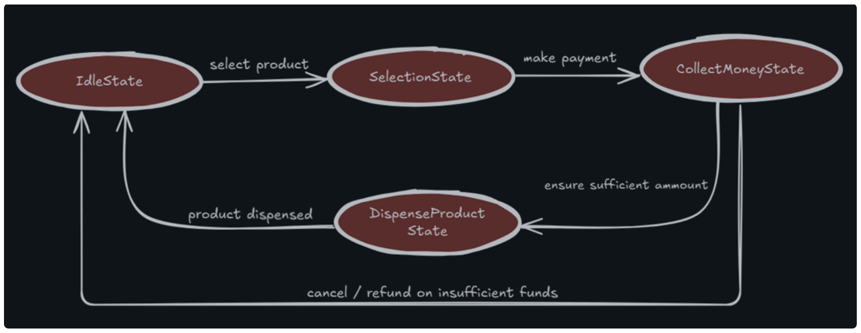
Definition

> **The State Pattern allows an object to change its behavior dynamically at runtime whenever there is a change in its internal state.**

Real Life Example: Vending Machine



Understanding the working of a Vending Machine



Different States and Operations

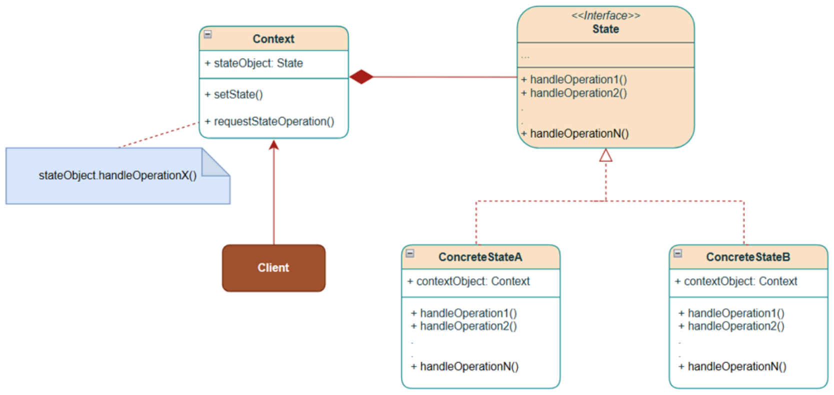
Example: Vending Machine

State	Operations
IdleState	• Insert Cash
SelectionState	• Choose the Product • Cancel/Refund • Return the Change
CollectMoneyState	• Insert Coin • Check for insufficient payment • Cancel/Refund
DispenseProductState	• Dispense Product

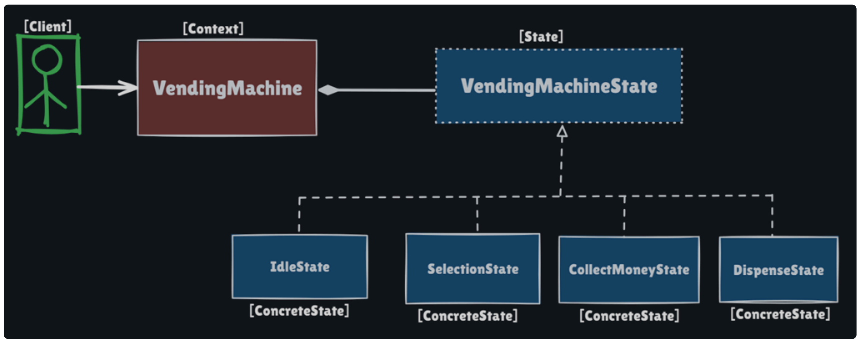
Example: TV

State	Operations
OFF	• Switch ON
ON	• Change Channel • Change Display Settings • Change Volume • Switch OFF

Class Diagram



Structure of State Pattern



1. **State Interface(e.g., `VendingMachineState`):** Declares common functions that all states must implement.
2. **Concrete States(e.g., `IdleState` , `SelectionState` , `CollectMoneyState` , `DispenseState`):** Each class implements the state interface behaviors(operations) differently depending on the current state of the vending machine, and an exception is thrown for operations that do not apply to the current state.
3. **Context Class** (e.g., `VendingMachine`): Maintains a reference to the current state. Holds all possible states as objects. Delegates call to the current state object.
4. **Client(`VendingMachineAppDemo`):** Interacts with Context Class (`VendingMachine`) and expects appropriate behavior as per changes in the state of the object.

Implementation(e.g., Vending Machine)

```java
// Step 1: Define the State interface(abstract class)
// All states will implement this interface
public abstract class VendingMachineState {

    public void beginTransaction(VendingMachine vendingMachine) throws Exception {
        throw new Exception("Transaction already in progress. Cancel to end the transaction.");
    }

    public void chooseProduct(VendingMachine vendingMachine, String productCode) throws Exception {
        throw new Exception("Product cannot be chosen in DispenseState. You need to begin transaction first.");
    }

    public void insertCoin(VendingMachine vendingMachine, Double amountPaid) throws Exception {
        throw new Exception("You cannot pay in DispenseState.  You need to begin transaction first.");
    }

    public void dispenseProduct(VendingMachine vendingMachine) throws Exception {
        throw new Exception("Product cannot be dispensed in CollectMoneyState. You need to pay first.");
    }

}
```

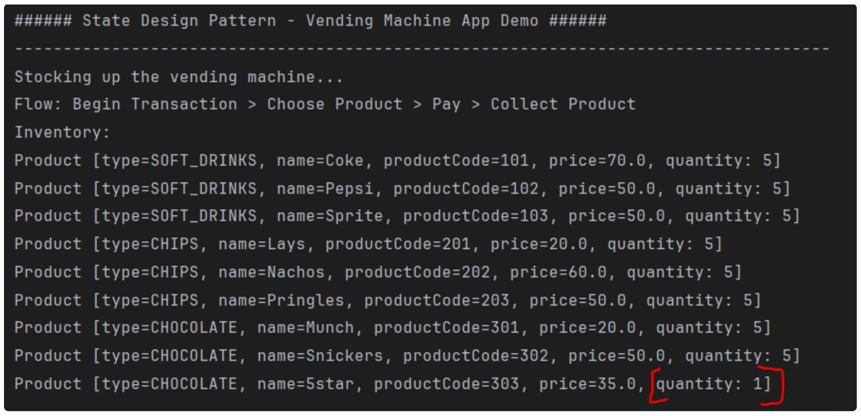
```java
// Step 2a: Concrete State - IdleState
// When machine has no coin inserted
public class IdleState extends VendingMachineState {

    @Override
    public void beginTransaction(VendingMachine vendingMachine) throws Exception {
        System.out.println("CurrentState: " + vendingMachine.getCurrentState().getClass().getSimpleName());
        System.out.println("A new Transaction has been started...");
        vendingMachine.setCurrentState(new SelectionState());
    }

}
// Step 2b: Concrete State - SelectionState
// When the customer is selecting a product
public class SelectionState extends VendingMachineState {

    @Override
    public void chooseProduct(VendingMachine vendingMachine, String productCode) throws Exception {
        System.out.println("CurrentState: " + vendingMachine.getCurrentState().getClass().getSimpleName());
        System.out.println("Product Selection in progress...");
        System.out.println("Product selected: " + productCode);
        Optional<Product> selectedProduct = vendingMachine.getInventory()
                .stream()
                .filter(product -> product.getProductCode().equals(productCode)).findFirst();
        if (selectedProduct.isEmpty()) {  // Wrong Product Code
            vendingMachine.setCurrentState(new IdleState());
            throw new Exception("WRONG PRODUCT CODE: The product code is invalid. Please enter the correct code.");
        }
        if (selectedProduct.get().getQuantity() == 0) { // Out of Stock
            vendingMachine.setCurrentState(new IdleState());
            throw new Exception("OUT OF STOCK: The product is out of stock. Please select another product.");
        }
        vendingMachine.setSelectedProduct(selectedProduct.get());
        vendingMachine.setCurrentState(new CollectMoneyState());
    }

}
// Step 2c: Concrete State - CollectMoneyState
// When the customer makes the payment for selected product
public class CollectMoneyState extends VendingMachineState {

    @Override
    public void insertCoin(VendingMachine vendingMachine, Double amountPaid) throws Exception {
        System.out.println("Current State: " + vendingMachine.getCurrentState().getClass().getSimpleName());
        System.out.println("You Paid: " + amountPaid);
        if (amountPaid < vendingMachine.getSelectedProduct().getPrice()) {
            vendingMachine.initiateRefund(amountPaid);
            vendingMachine.setCurrentState(new IdleState());
            throw new Exception("INSUFFICIENT AMOUNT: Amount paid is less than the product price. Amount Refunded.");
        }
        vendingMachine.setPaymentMade(amountPaid);
        vendingMachine.setCurrentState(new DispenseState());
    }

}
// Step 2d: Concrete State - DispenseState
// When the machine is dispensing the product
public class DispenseState extends VendingMachineState {

    @Override
    public void dispenseProduct(VendingMachine vendingMachine) throws Exception {
        System.out.println("Current State: " + vendingMachine.getCurrentState().getClass().getSimpleName());
        System.out.print("Product Dispensed: ");
        System.out.println(vendingMachine.getSelectedProduct().getName());
        System.out.println("Change Dispensed: " + vendingMachine.getChangeToReturn());
        vendingMachine.getInventory().stream()
                .filter(product -> product.getProductCode().equals(vendingMachine.getSelectedProduct().getProductCode()))
                .findFirst()
                .ifPresent(product -> product.setQuantity(product.getQuantity() - 1));
        vendingMachine.setCurrentState(new IdleState());
    }

}
```

```java
// Step 3: Context class - VendingMachine
// Holds reference to current state of the vending machine
public class VendingMachine {
    public ArrayList<Product> inventory;
    private VendingMachineState currentState;
    private Product selectedProduct;
    private double paymentMade;
    private double changeToReturn;

        public VendingMachine() {
        this.setCurrentState(new IdleState()); // Initial state
        this.setInventory(stockUpVendingMachine()); // Load the vending machine with products
    }

    public VendingMachineState getCurrentState() {
        return this.currentState;
    }

    public void setCurrentState(VendingMachineState state) {
        this.currentState = state;
    }

    public Product getSelectedProduct() {
        return this.selectedProduct;
    }

    public void setSelectedProduct(Product selectedProduct) {
        this.selectedProduct = selectedProduct;
    }

    public double getPaymentMade() {
        return this.paymentMade;
    }

    public void setPaymentMade(double paymentMade) {
        this.paymentMade = paymentMade;
        this.setChangeToReturn(paymentMade - selectedProduct.getPrice());
    }

    public double getChangeToReturn() {
        return this.changeToReturn;
    }

    public void setChangeToReturn(double changeToReturn) {
        this.changeToReturn = changeToReturn;
    }

    public ArrayList<Product> getInventory() {
        return this.inventory;
    }

    public void setInventory(ArrayList<Product> productList) {
        this.inventory = productList;
    }

    public void displayInventory() {
        System.out.println("Inventory:");
        for (Product product : inventory) {
            System.out.println(product.toString());
        }
    }

    // State methods
    public void beginTransaction() throws Exception {
        currentState.beginTransaction(this);
    }

    public void chooseProduct(String productCode) throws Exception {
        currentState.chooseProduct(this, productCode);
    }

    public void insertCoin(Double amountPaid) throws Exception {
        currentState.insertCoin(this, amountPaid);
    }

    public void dispenseProduct() throws Exception {
        currentState.dispenseProduct(this);
    }

    public void initiateRefund(double changeToReturn) {
        System.out.println("Refunded Amount: " + changeToReturn);
        this.changeToReturn = 0.00;
    }

    private ArrayList<Product> stockUpVendingMachine() {
        System.out.println("------------------------------------------------------------------------------------");
        System.out.println("Stocking up the vending machine...");
        ArrayList<Product> products = new ArrayList<>();
        // Shelf 1 - Soft Drinks
        products.add(new Product(ProductType.SOFT_DRINKS, "Coke", "101", 70.00, 5));
        products.add(new Product(ProductType.SOFT_DRINKS, "Pepsi", "102", 50.00, 5));
        products.add(new Product(ProductType.SOFT_DRINKS, "Sprite", "103", 50.00, 5));
        // Shelf 2 - Chips
        products.add(new Product(ProductType.CHIPS, "Lays", "201", 20.00, 5));
        products.add(new Product(ProductType.CHIPS, "Nachos", "202", 60.00, 5));
        products.add(new Product(ProductType.CHIPS, "Pringles", "203", 50.00, 5));
        // Shelf 3 - Chocolates
        products.add(new Product(ProductType.CHOCOLATE, "Munch", "301", 20.00, 5));
        products.add(new Product(ProductType.CHOCOLATE, "Snickers", "302", 50.00, 5));
        products.add(new Product(ProductType.CHOCOLATE, "5star", "303", 35.00, 1));

        return products;
    }
}
```

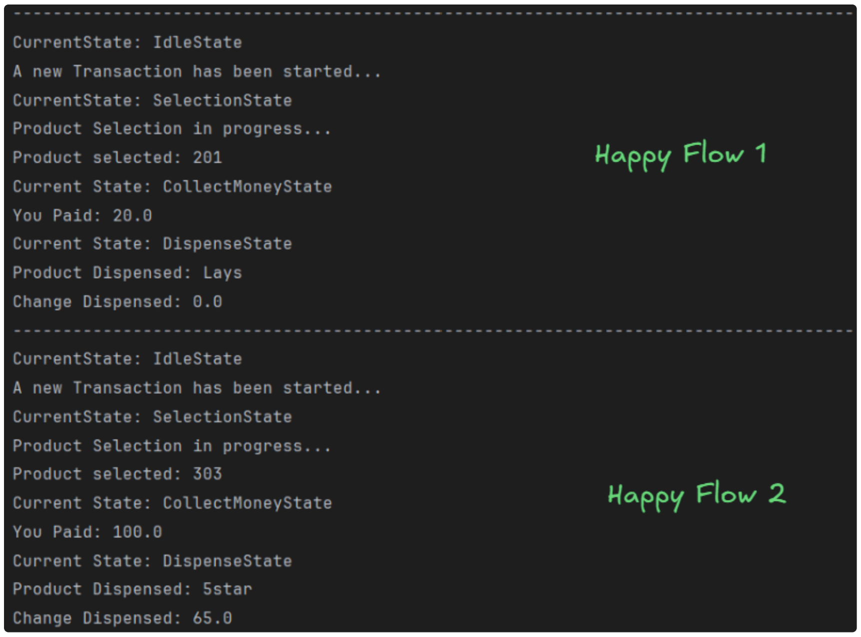
```java
enum ProductType {
    CHOCOLATE,
    CHIPS,
    SOFT_DRINKS
}

public class Product {
    ProductType type;
    String name;
    String productCode;
    Double price;

     public Product(ProductType type, String name, String productCode, Double price, int quantity) {
        this.type = type;
        this.name = name;
        this.productCode = productCode;
        this.price = price;
        this.quantity = quantity;
    }

    public ProductType getType() {
        return type;
    }

    public void setType(ProductType type) {
        this.type = type;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getProductCode() {
        return productCode;
    }

    public void setProductCode(String productCode) {
        this.productCode = productCode;
    }

    public Double getPrice() {
        return price;
    }

    public void setPrice(Double price) {
        this.price = price;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    @Override
    public String toString() {
        return "Product [type=" + type + ", name=" + name +
                ", productCode=" + productCode + ", price=" + price +
                ", quantity: " + quantity +
                "]";
    }
}

```

```java
// Client code - Interacts with Context Class (VendingMachine)
public class VendingMachineAppDemo {
    public static void main(String[] args) {
        System.out.println("###### State Design Pattern - Vending Machine App Demo ######");

        VendingMachine vendingMachine = new VendingMachine(); // Stock up the vending machine
        System.out.println("Flow: Begin Transaction > Choose Product > Pay > Collect Product");
        vendingMachine.displayInventory();
        try {
            // Happy Flow 1: User Buys Lays
            System.out.println("------------------------------------------------------------------------------------");
            vendingMachine.beginTransaction();
            vendingMachine.chooseProduct("201"); // Lays - 20 rupees
            vendingMachine.insertCoin(20.00);
            vendingMachine.dispenseProduct();

            // Happy Flow 2: User Buys Snickers
            System.out.println("------------------------------------------------------------------------------------");
            vendingMachine.beginTransaction();
            vendingMachine.chooseProduct("303"); // Snickers - 50 rupees
            vendingMachine.insertCoin(100.00); // Change to be returned: 50 rupees
            vendingMachine.dispenseProduct();

            //Negative Flow 1: User buys out of stock product
            // 5Star Quantity is 1
            System.out.println("------------------------------------------------------------------------------------");
            vendingMachine.beginTransaction();
            vendingMachine.chooseProduct("303"); // 5star - 50 rupees
            vendingMachine.insertCoin(35.00); // Change to be returned: 15 rupees
            vendingMachine.dispenseProduct();
            // 5Star Quantity is now 0
            System.out.println("------------------------------------------------------------------------------------");
            vendingMachine.beginTransaction();
            vendingMachine.chooseProduct("303"); // 5star - 50 rupees
            vendingMachine.insertCoin(35.00); // OUT OF STOCK exception: Refund 35 rupees
            vendingMachine.dispenseProduct(); // This line will not execute

            // Negative Flow 2: User pays insufficient amount
            /*System.out.println("------------------------------------------------------------------------------------");
            vendingMachine.beginTransaction();
            vendingMachine.chooseProduct("103"); // Sprite - 50 rupees
            vendingMachine.insertCoin(20.00); // throws exception - INSUFFICIENT PAYMENT exception
            vendingMachine.dispenseProduct(); // This line will not execute*/

            // Negative Flow 3: User enters wrong product code
            /*System.out.println("-----------------------------------------------------------------------------------");
            vendingMachine.beginTransaction();
            vendingMachine.chooseProduct("999"); // WRONG PRODUCT CODE exception
            vendingMachine.insertCoin(50.00); // this line will not execute
            vendingMachine.dispenseProduct(); // this line will not execute*/

            // Negative Flow 4: User tries to buy product without beginning a transaction
            /*System.out.println("-----------------------------------------------------------------------------------");
            vendingMachine.chooseProduct("201"); // throws exception
            // vendingMachine.insertCoin(50.00); // throws exception
            // vendingMachine.dispenseProduct(); // throws exception*/

        } catch (Exception e) {
            System.out.println(e.getMessage());
        } finally {
            System.out.println("-----------------------------------------------------------------------------------");
            System.out.println("Flow: New Transaction > Choose Product > Pay > Collect Product");
            vendingMachine.displayInventory();
        }
    }
}
```

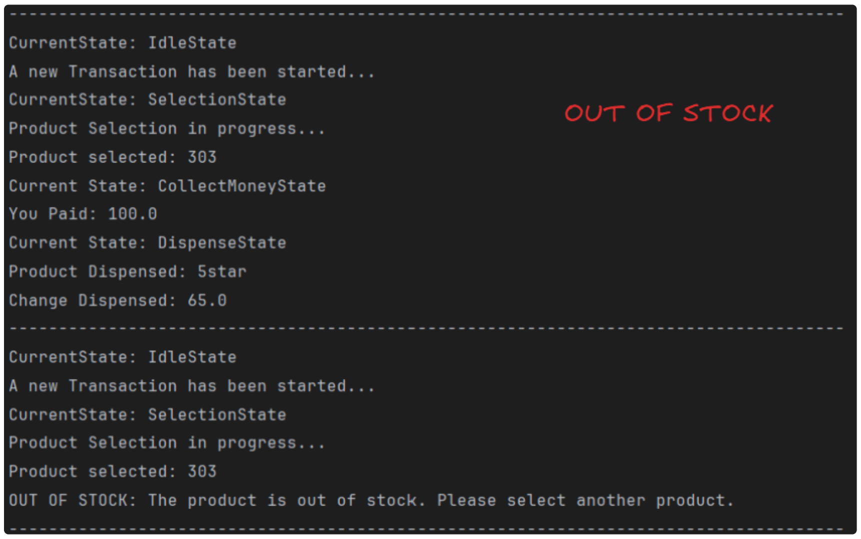
Output



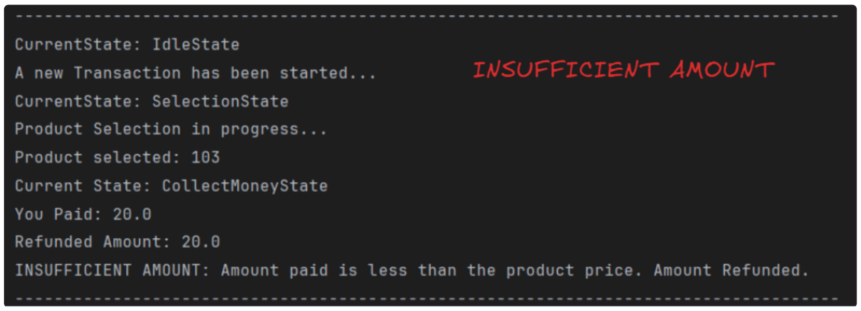
```
###### State Design Pattern - Vending Machine App Demo ######
-----------------------------------------------------------------------------------
Stocking up the vending machine...
Flow: Begin Transaction > Choose Product > Pay > Collect Product
Inventory:
Product [type=SOFT_DRINKS, name=Coke, productCode=101, price=70.0, quantity: 5]
Product [type=SOFT_DRINKS, name=Pepsi, productCode=102, price=50.0, quantity: 5]
Product [type=SOFT_DRINKS, name=Sprite, productCode=103, price=50.0, quantity: 5]
Product [type=CHIPS, name=Lays, productCode=201, price=20.0, quantity: 5]
Product [type=CHIPS, name=Nachos, productCode=202, price=60.0, quantity: 5]
Product [type=CHIPS, name=Pringles, productCode=203, price=50.0, quantity: 5]
Product [type=CHOCOLATE, name=Munch, productCode=301, price=20.0, quantity: 5]
Product [type=CHOCOLATE, name=Snickers, productCode=302, price=50.0, quantity: 5]
Product [type=CHOCOLATE, name=5star, productCode=303, price=35.0, quantity: 1]
```



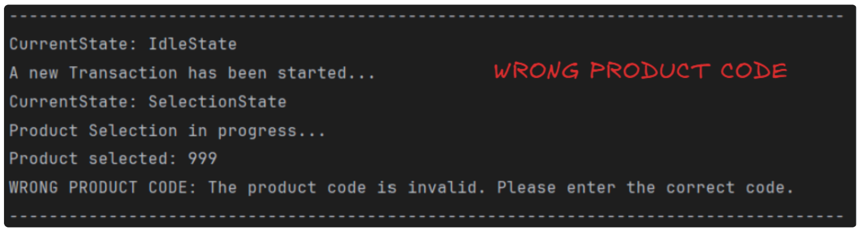
```
-----------------------------------------------------------------------------------
CurrentState: IdleState
A new Transaction has been started...
CurrentState: SelectionState
Product Selection in progress...
Product selected: 201
Current State: CollectMoneyState
You Paid: 20.0
Current State: DispenseState
Product Dispensed: Lays
Change Dispensed: 0.0
-----------------------------------------------------------------------------------
CurrentState: IdleState
A new Transaction has been started...
CurrentState: SelectionState
Product Selection in progress...
Product selected: 303
Current State: CollectMoneyState
You Paid: 100.0
Current State: DispenseState
Product Dispensed: 5star
Change Dispensed: 65.0
```



```
-----------------------------------------------------------------------------------
CurrentState: IdleState
A new Transaction has been started...
CurrentState: SelectionState
Product Selection in progress...
Product selected: 303
Current State: CollectMoneyState
You Paid: 100.0
Current State: DispenseState
Product Dispensed: 5star
Change Dispensed: 65.0
-----------------------------------------------------------------------------------
CurrentState: IdleState
A new Transaction has been started...
CurrentState: SelectionState
Product Selection in progress...
Product selected: 303
OUT OF STOCK: The product is out of stock. Please select another product.
-----------------------------------------------------------------------------------
```



```
-----------------------------------------------------------------------------------
CurrentState: IdleState
A new Transaction has been started...
CurrentState: SelectionState
Product Selection in progress...
Product selected: 103
Current State: CollectMoneyState
You Paid: 20.0
Refunded Amount: 20.0
INSUFFICIENT AMOUNT: Amount paid is less than the product price. Amount Refunded.
-----------------------------------------------------------------------------------
```



```
-----------------------------------------------------------------------------------
CurrentState: IdleState
A new Transaction has been started...
CurrentState: SelectionState
Product Selection in progress...
Product selected: 999
WRONG PRODUCT CODE: The product code is invalid. Please enter the correct code.
-----------------------------------------------------------------------------------
```