Vending Machine

A vending machine is an automated machine. A vending machine dispenses the product to the users based on the amount of money inserted & the selection of the product.

Expectation to Interview

→ what function do the vending machines perform? Alternatively, how many different states can the vending machine have?

→ After inserting money into the money, what does the system do?

→ who presses the vending machine button, & what happens after pressing it?

→ what does the dispense function do?

→ If the vending machine is in a dispense state, is it possible to insert money

Money handling

→ what should the system do if we pay less money than the product price?

→ what should system do if we pay more money than the product price

→ Can the credit card can be used to input money or can only cash be used?

Requirement Collection

R1: There a different products placed at different positions in the vending machine

R2: The Vending machine can be in one of the three states:

No money inserted : No money inserted

Money Inserted state : Money is inserted into the machine

Dispense state : The machine gives out product.

R3: There can be two actors in the system. One is the user & the other is the admin.

R4: The admin can add a product to the machine or remove a product from the machine

R5: The System should allow the users to select a product they went to purchase from the machine by specifying the rack number.

R6: The user can insert money into the machine in the form of cash

R.7: The system should be able to calculate the money inserted into the machine

R8: The System should check whether the user inserted the exact amount required for the specific product into the machine

R9: If the amount is greater than the product price, the system should change back the user & dispense the product.

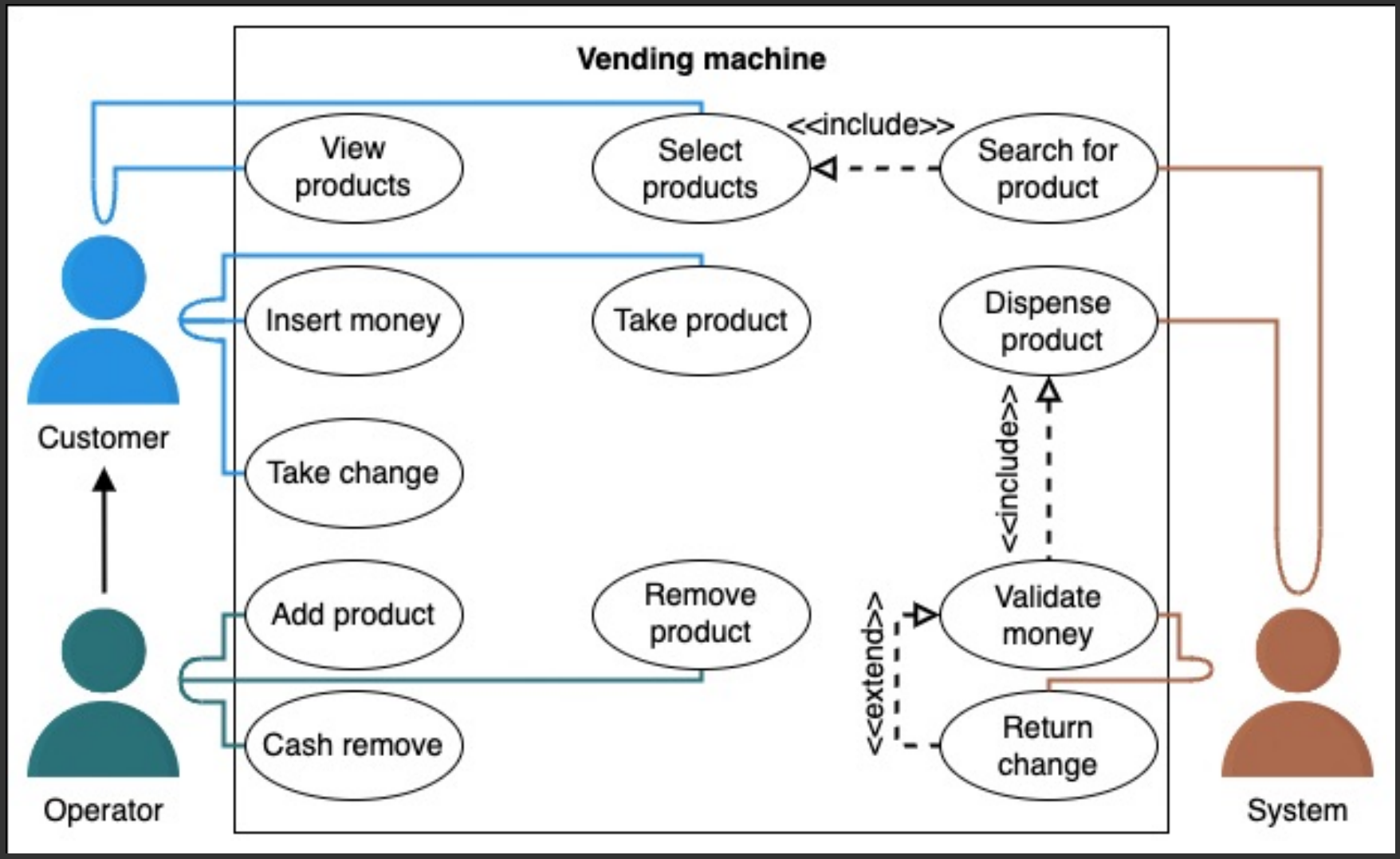
R10: If the amount less than the product price, the system should display an error message & return the money.

Actors

Primary Actors → Customer, Operator

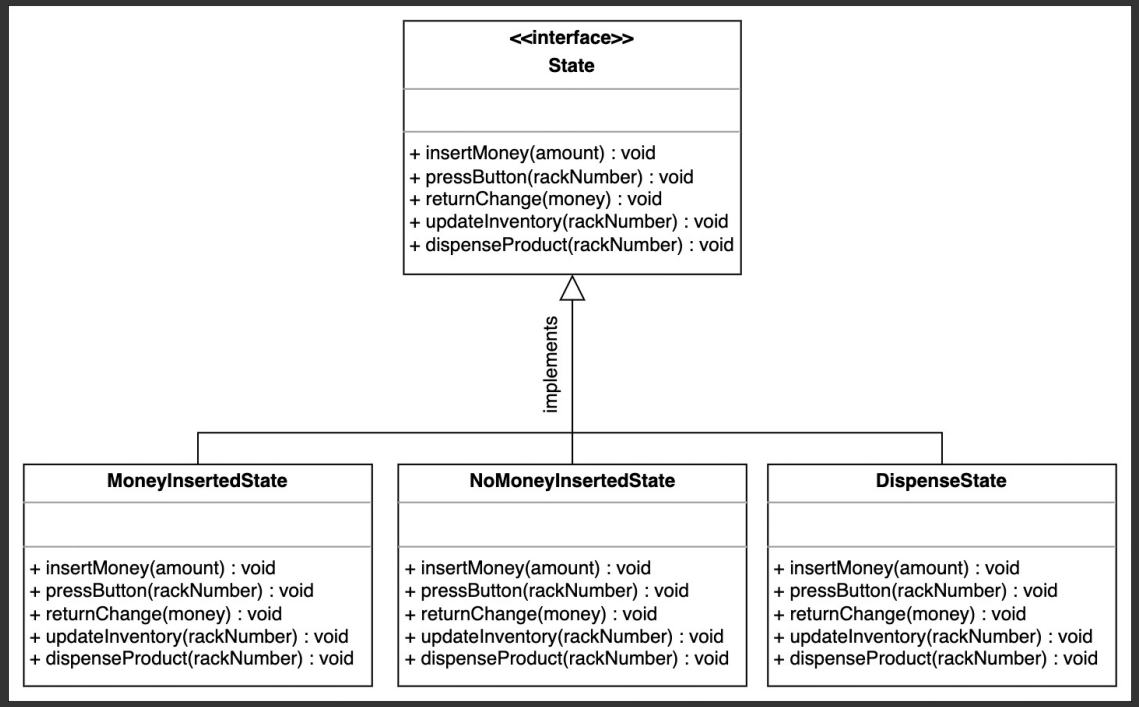
Secondary Actors → System

Customer	Operator	System
View products	Add product	Search product
Select products	Remove product	Dispense product
Insert money	Cash remove	Validate money
Take product	View products	Return change
Take change	Select products	
	Insert money	
	Take product	
	Take change	

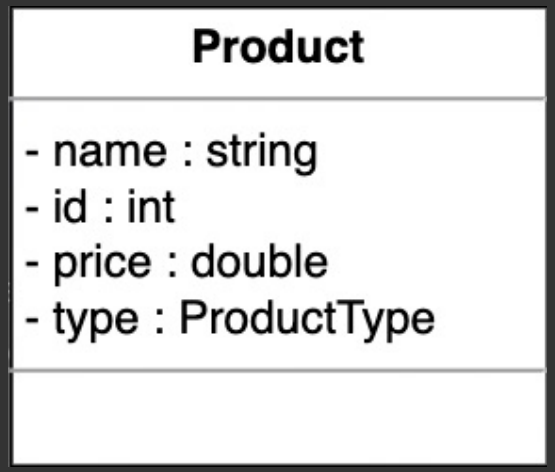

Vending machine
View products
Select products
<<include>>
Search for product
Insert money
Take product
Dispense product
Customer
Take change
<<include>>
Add product
Remove product
Validate money
<<extend>>
Cash remove
Return change
Operator
System

Class Diagram

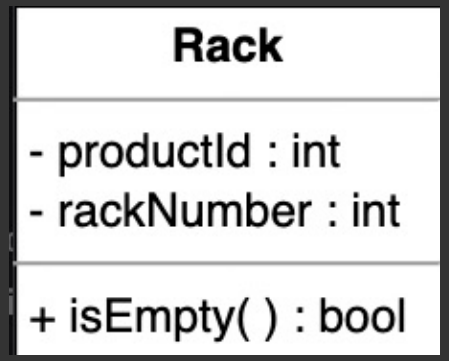
State -



Product



Rack



Vending Machine

VendingMachine

- currentState : State
- amount : double
- noOfRacks : int
- racks : List<Rack>
- availableRacks : List<int>

+ insertMoney(amount) : void
+ pressButton(rackNumber) : void
+ returnChange(money) : void
+ dispenseProduct(rackNumber) : void
+ updateInventory(rackNumber) : void
+ getProductIdAtRack(rackId) : int

Inventory

Inventory

- noOfProducts : int
- products : List<Product>

+ addProduct(productId, rackId) : void
+ removeProduct(productId, rackId) : void

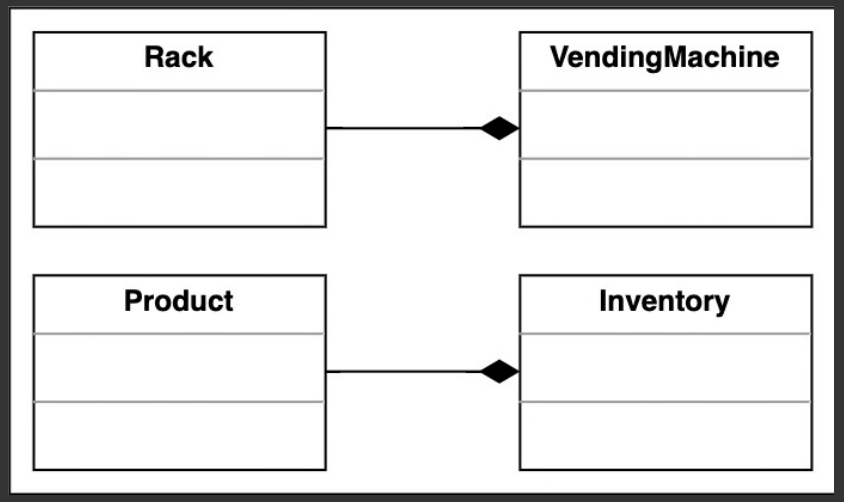
Enumeration

<<enumeration>>

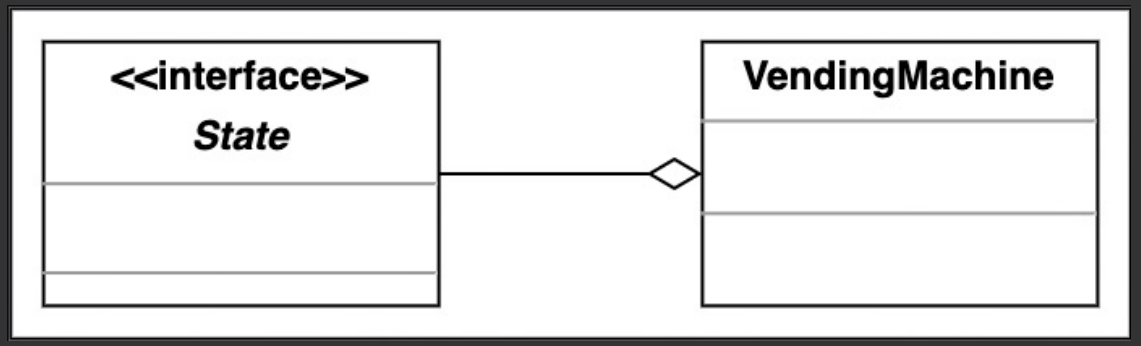
ProductType

Chocolate
Snack
Beverage
Other

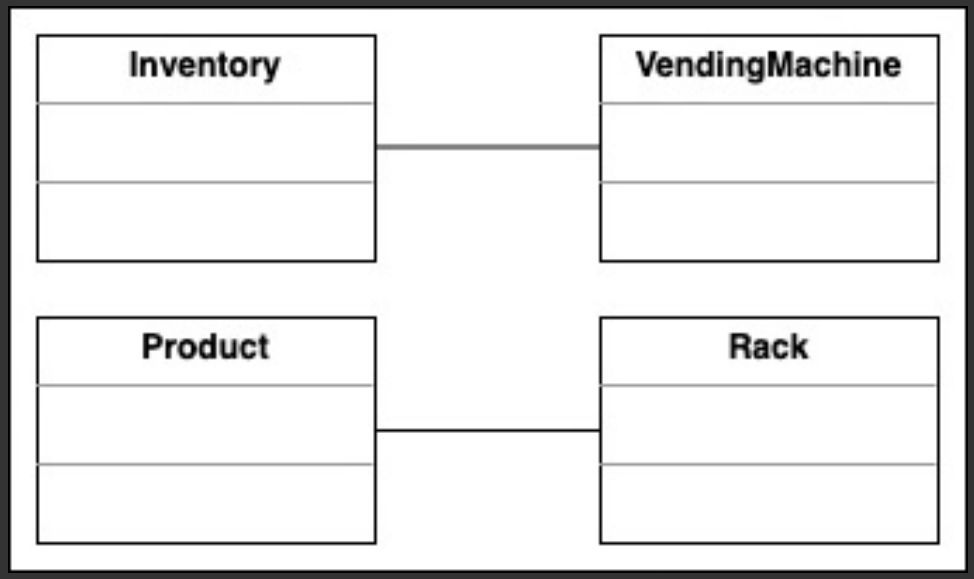
Composition



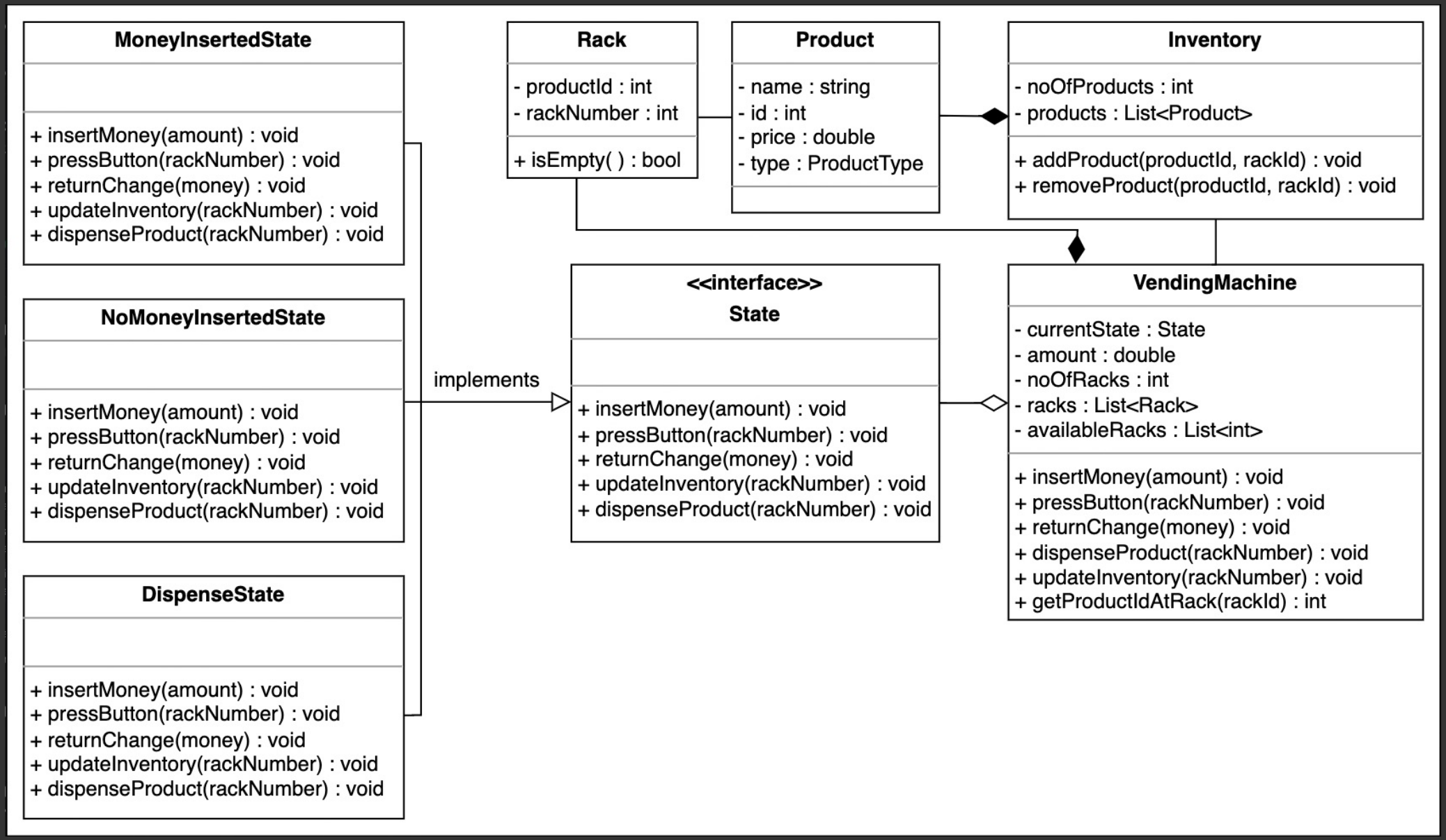
Aggregation



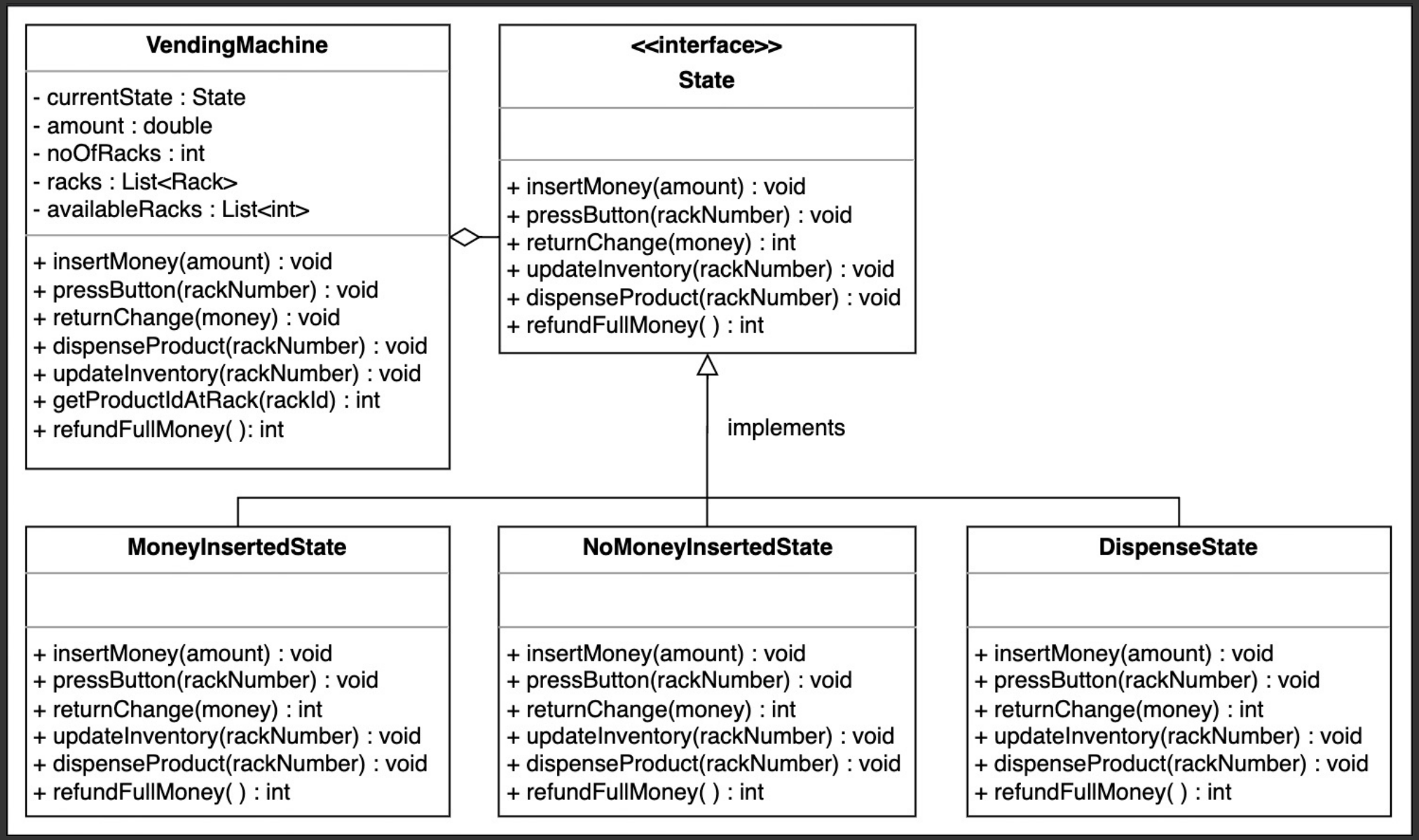
Association



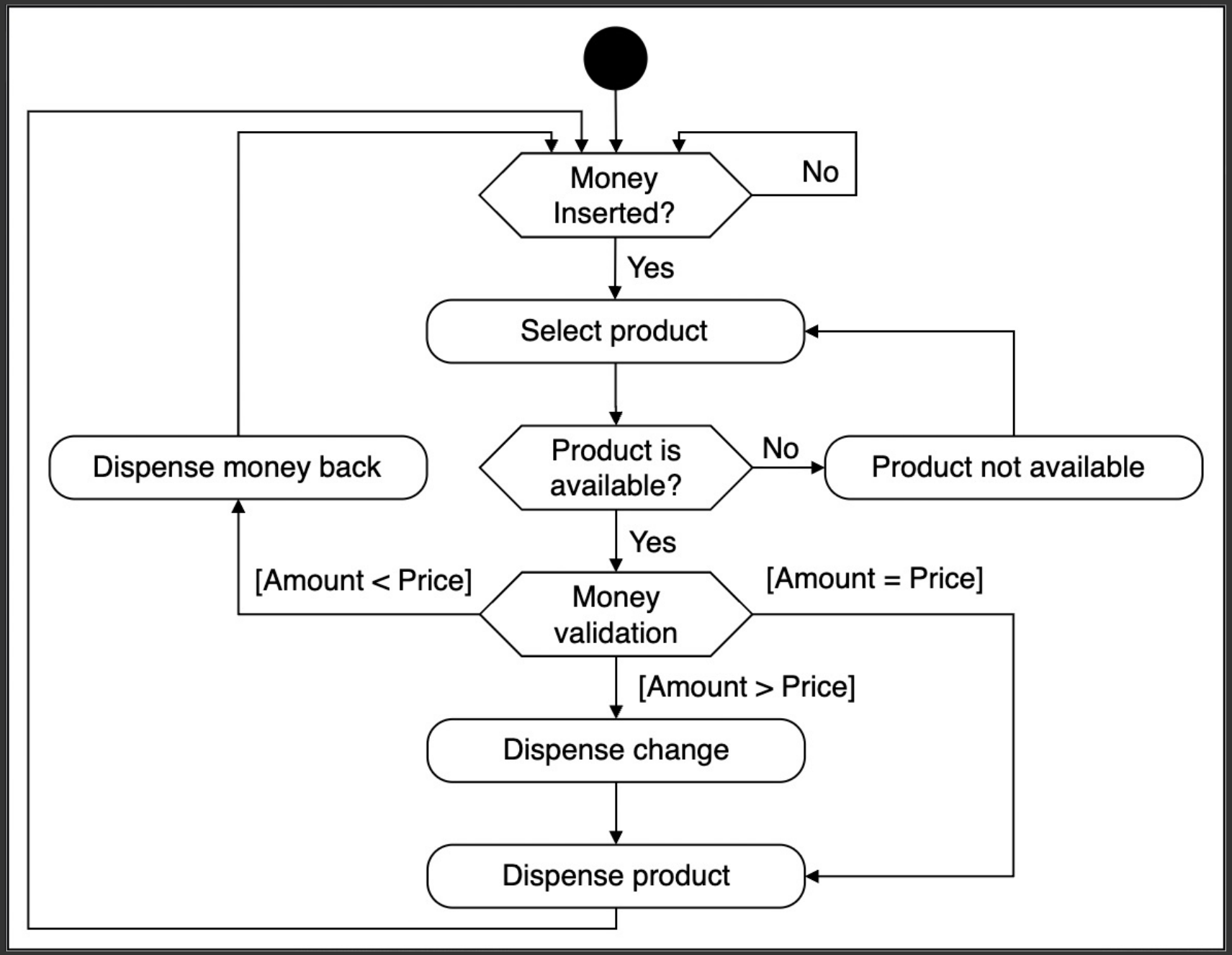
Class Diagram



follow up → Vending Machine Should have option of cancel the operation. Customer will get Refund



Action → Activity Diagram



Code For the Vending Machine

enumeration

```
// Enumerations
enum ProductType {
  CHOCOLATE,
  SNACK,
  BEVERAGE,
  OTHER
}
```

State

```
// State is an interface
public interface State {
    // Interface method (does not have a body)
    public void insertMoney(VendingMachine machine, double amount);
    public void pressButton(VendingMachine machine, int rackNumber);
    public void returnChange(double amount);
    public void updateInventory(VendingMachine machine, int rackNumber);
    public void dispenseProduct(VendingMachine machine, int rackNumber);
}

public class NoMoneyInsertedState implements State {
    @override
    public void insertMoney(VendingMachine machine, double amount) {
      // changes state to MonenInsertedState
    }
    public void pressButton(VendingMachine machine, int rackNumber) {}
    public void returnChange(double amount) {}
    public void updateInventory(VendingMachine machine, int rackNumber) {}
    public void dispenseProduct(VendingMachine machine, int rackNumber) {}
}

public class MoneyInsertedState implements State {
    @override
    public void insertMoney(VendingMachine machine, double amount) {}
    public void pressButton(VendingMachine machine, int rackNumber) {
        // check if product item is available
        // validate money
        // change state to DispenseState
    }
    public void returnChange(double amount) {}
    public void updateInventory(VendingMachine machine, int rackNumber) {}
    public void dispenseProduct(VendingMachine machine, int rackNumber) {}
}

public class DispenseState implements State {
    @override
    public void insertMoney(VendingMachine machine, double amount) {}
    public void pressButton(VendingMachine machine, int rackNumber) {}
    public void returnChange(double amount){}
    public void updateInventory(VendingMachine machine, int rackNumber) {}
    public void dispenseProduct(VendingMachine machine, int rackNumber) {
        // dispense product
        // change state to NoMoneyInsertedState
    }
}
```

Product, rack & inventory

```
public class Product {
  private String name;
  private int id;
  private double price;
  private ProductType type;
}

public class Rack {
  private int productId;
  private int rackNumber;

  public boolean isEmpty();
}

public class Inventory {
  private int noOfProducts;
  private List<Product> products;

  public void addProduct(int productId, int rackId);
  public void removeProduct(int productId, int rackId);
}
```

Vending Machine

```
public class VendingMachine {
    private State currentState;
    private double amount;
    private int noOfRacks;
    private List<Rack> racks;
    private List<int> availableRacks;

    // The VendingMachine is a Singleton class that ensures it will have
only one active instance at a time
    private static VendingMachine vendingMachine = null;

    // Created a private constructor to add a restriction (due to
Singleton)
    private VendingMachine();

    // Created a static method to access the singleton instance of
VendingMachine
    public static VendingMachine getInstance() {
        if (vendingMachine == null) {
            vendingMachine = new VendingMachine();
        }
        return vendingMachine;
    }

    public void insertMoney(double amount) {}
    public void pressButton(int rackNumber) {}
    public void returnChange(double amount) {}
    public void updateInventory(int rackNumber) {}
    public void dispenseProduct(int rackNumber) {}
    public int getProductIdAtRack(int rackNumber) {}
}
```