Low-Level Design Document for Vending Machine

Contents

[Introduction 1](#_Toc185417617)

[Purpose 1](#_Toc185417618)

[Class Specifications 1](#_Toc185417619)

[VendingMachine 1](#_Toc185417620)

[InventoryManager 1](#_Toc185417621)

[MoneyManager 2](#_Toc185417622)

[Item 3](#_Toc185417623)

[IState 4](#_Toc185417624)

[IdleState 4](#_Toc185417625)

[AwaitingPaymentState 4](#_Toc185417626)

[Events 5](#_Toc185417627)

[VendingMachineEventArgs 5](#_Toc185417628)

[Derived Event Classes 6](#_Toc185417629)

[Sequence Diagrams 6](#_Toc185417630)

[Vending Machine Actions 6](#_Toc185417631)

[User Selects an Item 7](#_Toc185417632)

[User Inserts Cash 7](#_Toc185417633)

[User Gets a Refund or Change 7](#_Toc185417634)

[State Machine Transitions 8](#_Toc185417635)

[User Selects an Item 8](#_Toc185417636)

[User Cancels Transaction 8](#_Toc185417637)

[User Inserts Cash 9](#_Toc185417638)

[Conclusion 10](#_Toc185417639)

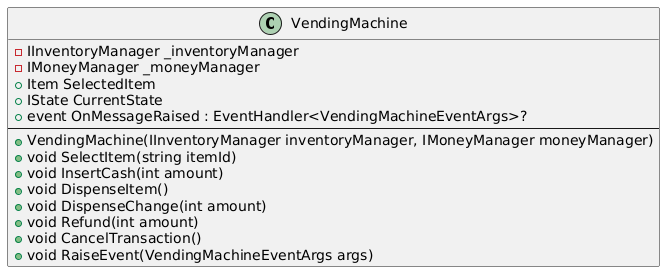
# Introduction

## Purpose

This document provides the low-level design for the Vending Machine System, detailing implementation specifics, class attributes, methods, interactions, and exception handling to enable the development of a robust and maintainable system.

# Class Specifications

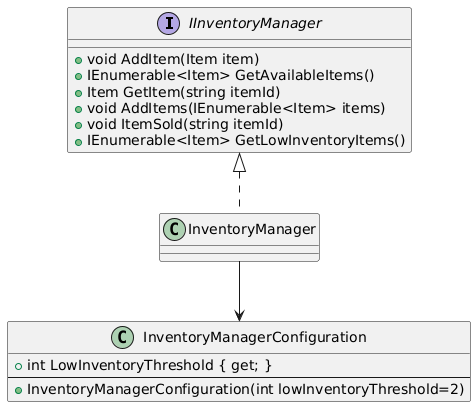
## VendingMachine



This class encapsulates the core behavior of a vending machine, including managing inventory, handling monetary transactions, and raising events for various states and actions. It operates in conjunction with state objects implementing the **IState** interface to handle specific behaviors based on the current state of the machine.

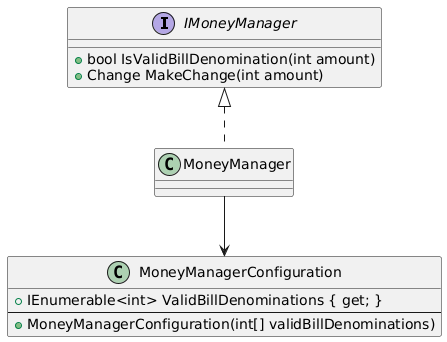
## InventoryManager

This class is responsible for adding, retrieving, and managing items in the inventory. It also handles operations such as tracking low stock, marking items as sold, and retrieving available items. **InventoryManager** is a concrete implementation of the **IInventoryManager** interface. **InventoryManagerConfiguration** is injected into **InventoryManager**. It provides a single configuration option - the threshold for low inventory levels, with a default of two items.



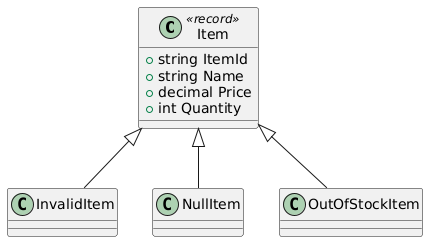
## MoneyManager

This class is responsible for managing monetary operations, and calculating change, within **VendingMachine. MoneyManager** is a concrete implementation of the **IMoneyManager** interface. **MoneyManagerConfiguration** is injected into **MoneyManager**. It provides a single configuration option – an array of valid bill denominations. Currently, the **VendingMachine** is configured to accept only one-dollar bills.



## Item

This class represents an item in the inventory with its unique identifier, name, price, and quantity.



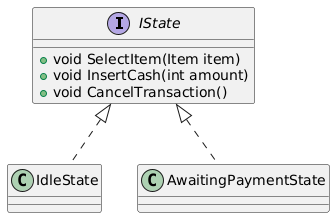
There are three classes derived from **Item:**

* **NullItem:** Used to signify the absence of a valid item.
* **OutOfStockItem:** Used to indicate that a specific item exists in the inventory but currently has a quantity of zero.
* **InvalidItem:** Used to indicate that an item does not exist in the inventory.

These classes are an example of the Null Object Pattern. This pattern helps to avoid null checks and provide more expressiveness and context to the state of the item.

## IState

This interface defines the contract for a state in the vending machine's state management system. Defines the contract for a state in the vending machine's state management system.



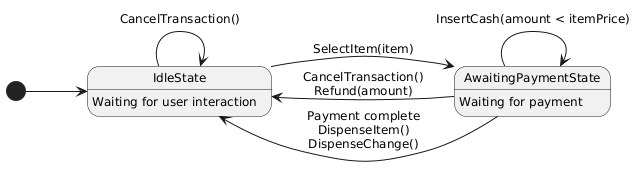
## IdleState

This class represents the idle state of the vending machine, where no item is selected, and the machine is waiting for user interaction. In this state, the vending machine allows the user to select an item. Attempting to insert cash or perform other actions without selecting an item first will result in appropriate feedback.

## AwaitingPaymentState

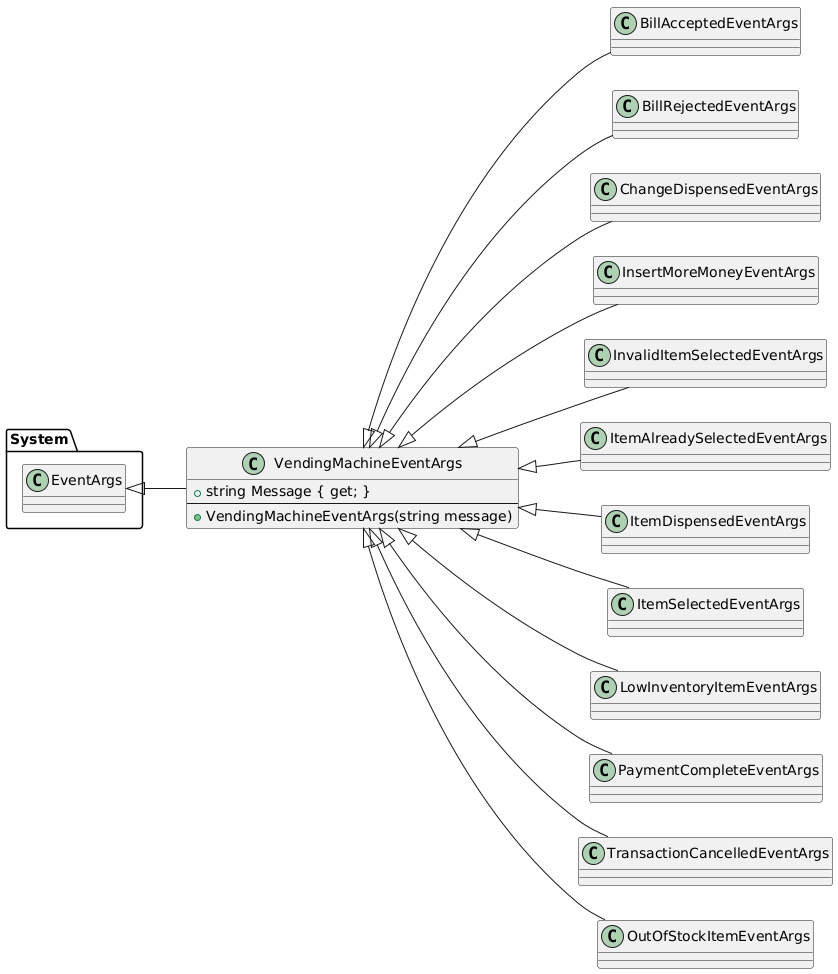
This class represents the state of a vending machine when it is awaiting payment for a selected item. This state handles the insertion of cash, cancellation of the transaction, and transitions to other states based on the payment status. It ensures that the vending machine processes payments correctly and dispenses items and change as needed.

The vending machine state diagram is shown below. The state transitions indicate the action methods invoked, along with conditions (if needed), to transition to the next state.



# Events

Coordination and communication between **VendingMachine** and the various states happen with events. This design provides a clear separation of concerns by allowing **VendingMachine** to notify subscribers about key actions or changes. The **Console Application** that serves as the user interface for the vending machine, utilizes some of these events for flow control and relay important messages to the user.



## VendingMachineEventArgs

This class serves as the base class for event arguments related to vending machine events. This class provides a common structure for all vending machine-related event arguments, encapsulating a descriptive message about the event.

## Derived Event Classes

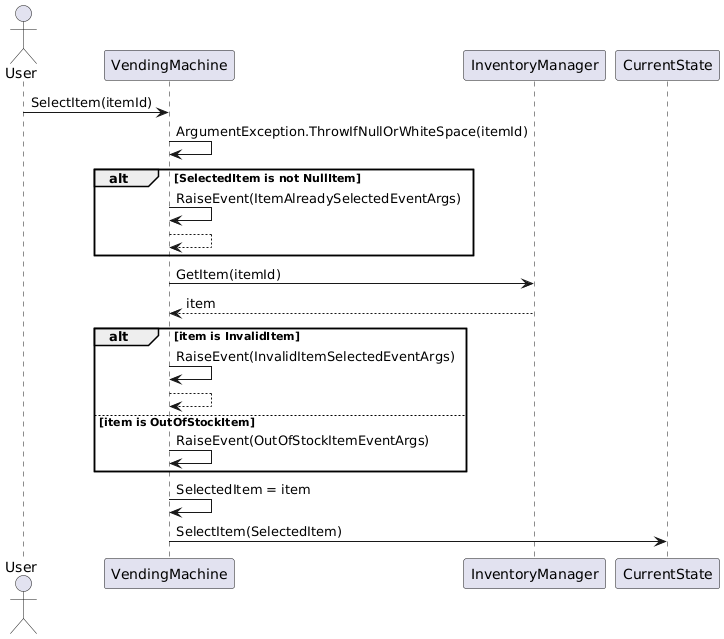
* **BillAcceptedEventArgs**: Represents event arguments for when a bill is accepted by the vending machine.
* **BillRejectedEventArgs**: Represents event arguments for when a bill is rejected by the vending machine.
* **ItemSelectedEventArgs**: Represents event arguments for when an item is selected from the vending machine.
* **OutOfStockItemEventArgs**: Represents event arguments for when a selected item is out of stock in the vending machine.
* **PaymentCompleteEventArgs**: Represents event arguments for when a payment transaction is completed in the vending machine.
* **TransactionCancelledEventArgs**: Represents event arguments for when a transaction is cancelled in the vending machine.
* **ChangeDispensedEventArgs**: Represents event arguments for when change is dispensed by the vending machine.
* **InsertMoreMoneyEventArgs**: Represents event arguments for when more money needs to be inserted into the vending machine.
* **InvalidItemSelectedEventArgs**: Represents event arguments for when an invalid item is selected from the vending machine.
* **ItemAlreadySelectedEventArgs**: Represents event arguments for when an item that is already selected is selected again.
* **ItemDispensedEventArgs**: Represents event arguments for when an item is dispensed from the vending machine.
* **LowInventoryItemEventArgs**: Represents event arguments for when an item in the vending machine has low inventory.
* **OutOfStockItemEventArgs**: Represents event arguments for when a selected item is out of stock in the vending machine.

# Sequence Diagrams

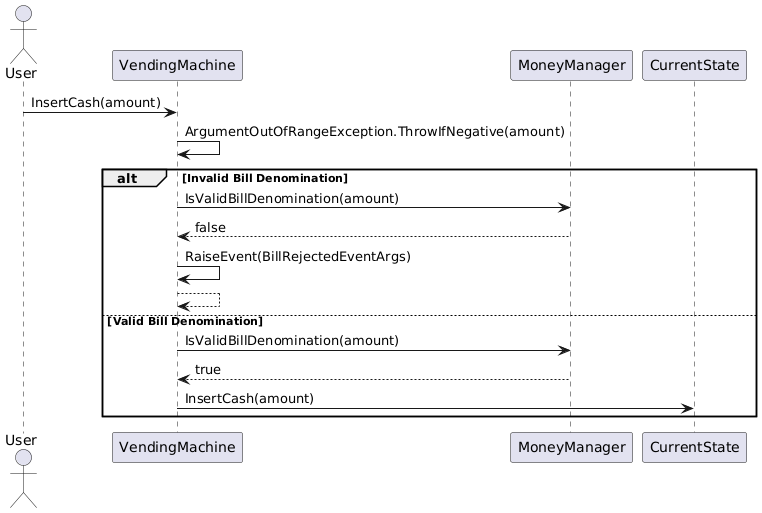
## Vending Machine Actions

This section shows sequence diagrams for key actions. This helps visualize the various events that are raised.

### User Selects an Item



### User Inserts Cash



### User Gets a Refund or Change

Both a refund and change returned is money returned to the user, hence the same sequence is used for both use cases.

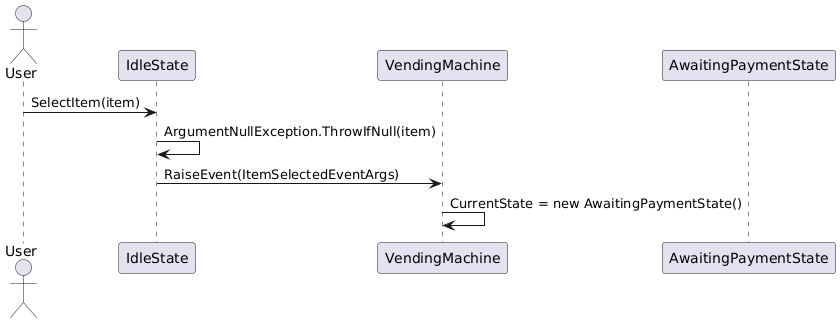


## State Machine Transitions

The diagrams below show the sequence diagrams for key state transitions.

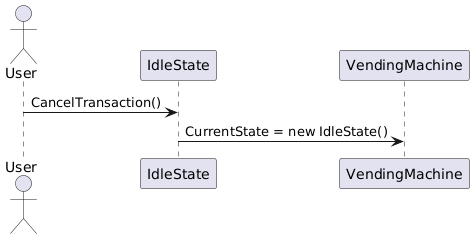
### User Selects an Item

The transition is from Idle to AwaitingPayment.



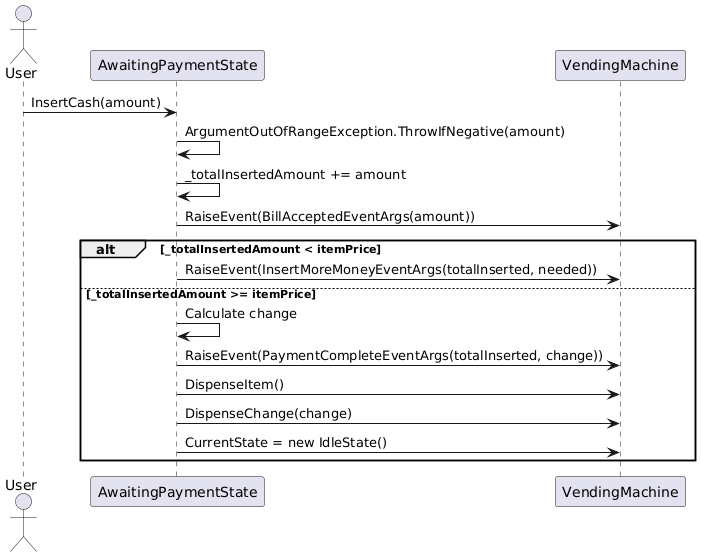
### User Cancels Transaction

The machine stays in the Idle state.



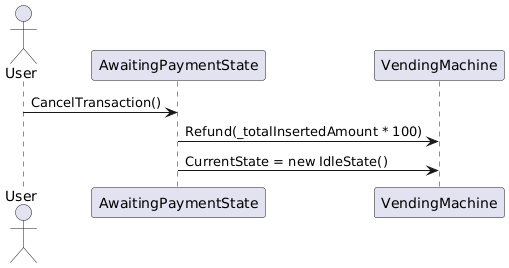
### User Inserts Cash

After the item is dispensed, the machine transitions to the Idle state from AwaitingPayment.



User Cancels Transaction

If the machine is in the AwaitingPayment state and the user cancels the transaction prior to completing payment, a refund is issued in coins. The machine then transitions to the Idle state.



# Conclusion

This low-level design provides a granular blueprint for implementing the Vending Machine system. Each component is modular and testable, ensuring the system is extensible, robust, and easy to maintain.