实物类面向对象设计

文泰来 老师

课程大纲

- 实物类OOD题型
- 实物类OOD解题思路
- Vending machine
- Coffee maker
- Kindle

实物类OOD题型

- Vending machine
- Jukebox
- CD Player
- Coffee maker
- Kindle

实物类OOD题型

- 频率：中高

实物类OOD题型

- 频率：中高
- 难度：中低

实物类OOD解题技巧

- 考虑对于实物的输入输出

实物类OOD解题技巧

- 考虑对于实物的输入输出

例子：Coffee maker

实物类OOD解题技巧

- 考虑对于实物的输入输出

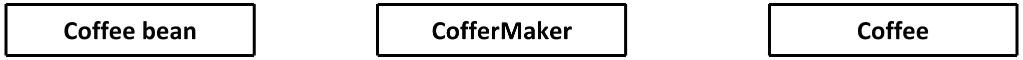
例子：Coffee maker

CofferMaker

实物类OOD解题技巧

- 考虑对于实物的输入输出

例子：Coffee maker



实物类OOD解题技巧

- Design pattern的运用

实物类OOD解题技巧

- Design pattern的运用

State pattern
Decorate pattern
Factory pattern

Vending Machine

- Can you design a vending machine?

Clarify

- What
- How

Clarify

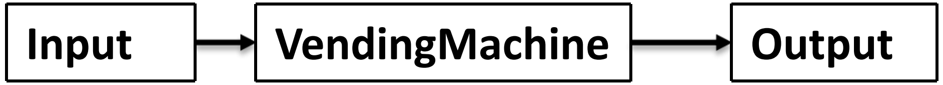
- What

关键字： Vending machine

Clarify

- What

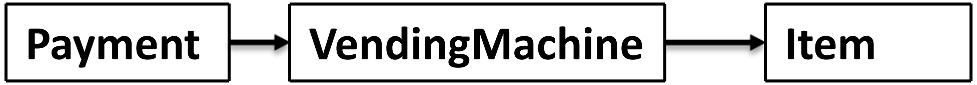
关键字： Vending machine



Clarify

- What

关键字： Vending machine



Clarify

- What

关键字： Vending machine, Payment, Item

Clarify

- 关键字： Vending machine

Clarify

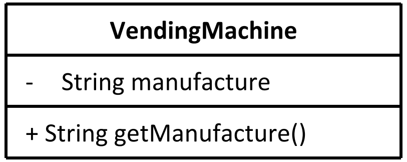
- 关键字： Vending machine

厂家，重量，颜色…

Clarify

- 关键字： Vending machine

厂家，重量，颜色…



VendingMachine

- String manufacture

+ String getManufacture()

Clarify

- 关键字： Vending machine

大小： Vending machine的大小是否有限制？

Clarify

- 关键字： Item

Clarify

- 关键字： Item

What items does this vending machine sell?

Clarify

- 关键字： Item

What items does this vending machine sell?

Naïve design approach: each item matches a class

Clarify

- 关键字： Item

What to do when an item sold out?

Clarify

- 关键字： Item

What to do when an item sold out?

Design: Might need to support refill use case

Clarify

- 关键字： Payment

Clarify

- 关键字： Payment

What are the supported payment methods?

Clarify

- Payment

- Coin

- Paper money

- Credit card

Clarify

- Payment

- Coin/Paper money : 知道当前收了多少钱，找零

- Credit card: 直接当前Item的价格

Clarify

- 对于本题：

- 假设Vending machine的大小没有限制
- 假设目前只卖三种产品：Coke, Sprite和Mountain Dew
- 假设目前只接受硬币

Clarify

- How

Clarify

- How

Clarify

- How to select item to purchase?

Design: selectItem(?)

Clarify

- 对于本题：

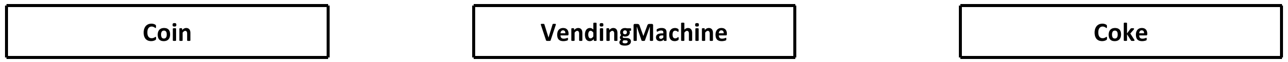
- 假设输入一个input代表一种Item (e.g. A1 -> Coke)

Core Object

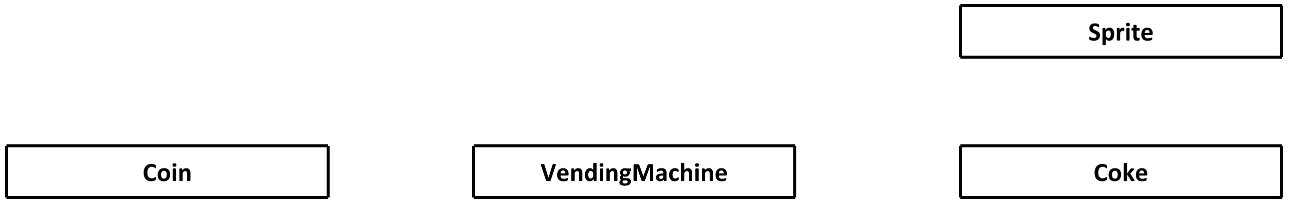
Core Object

Coin
VendingMachine

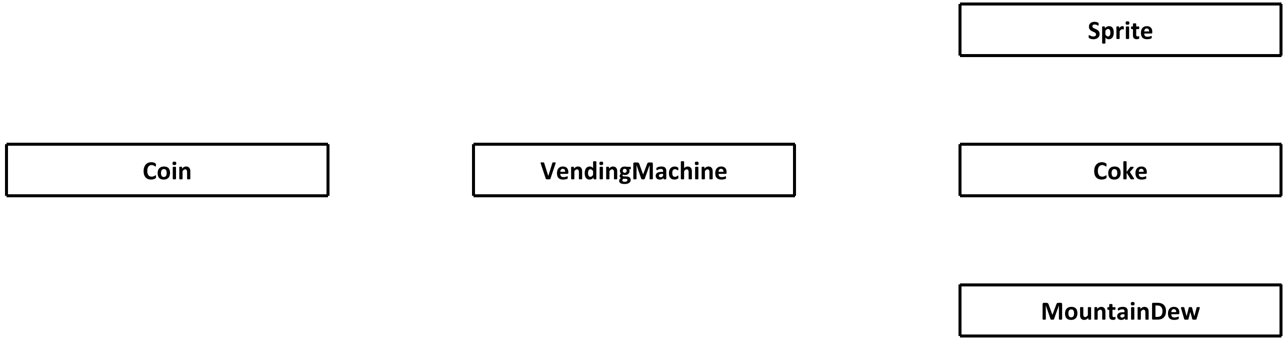
Core Object



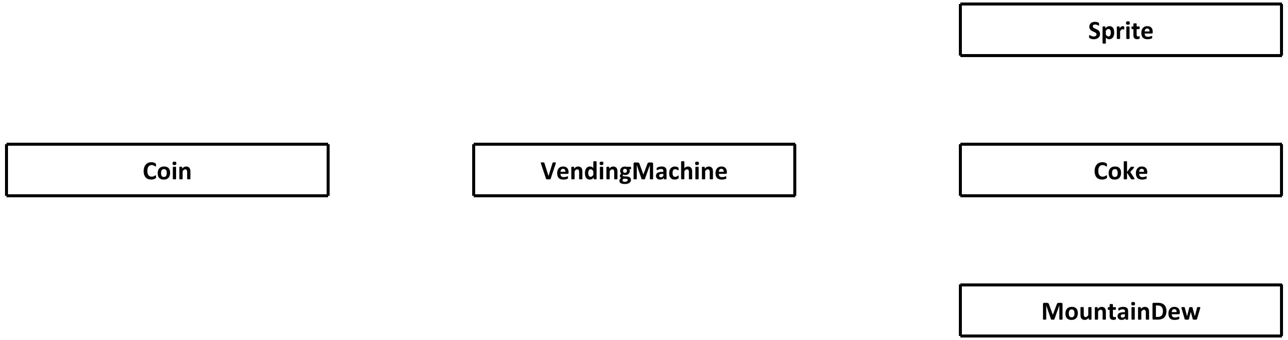
Core Object



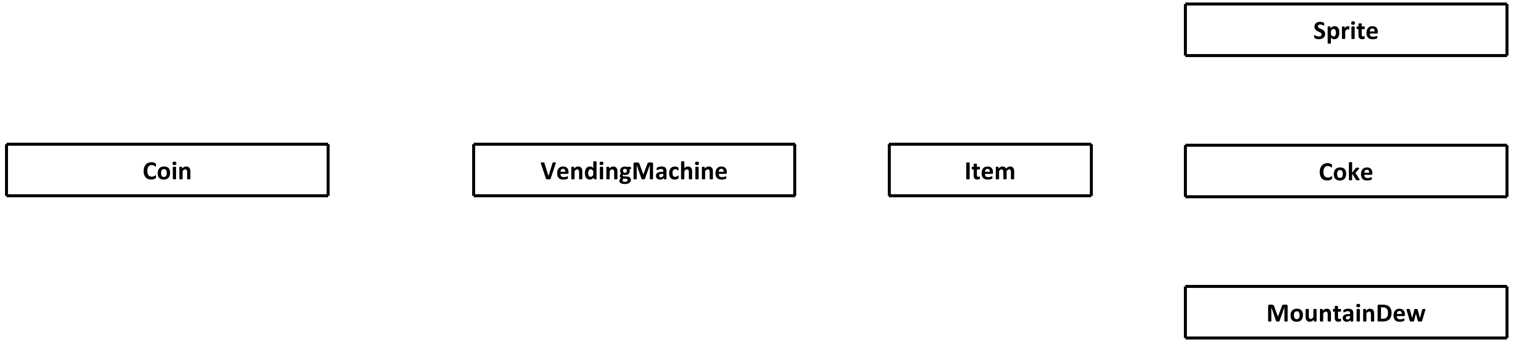
Core Object



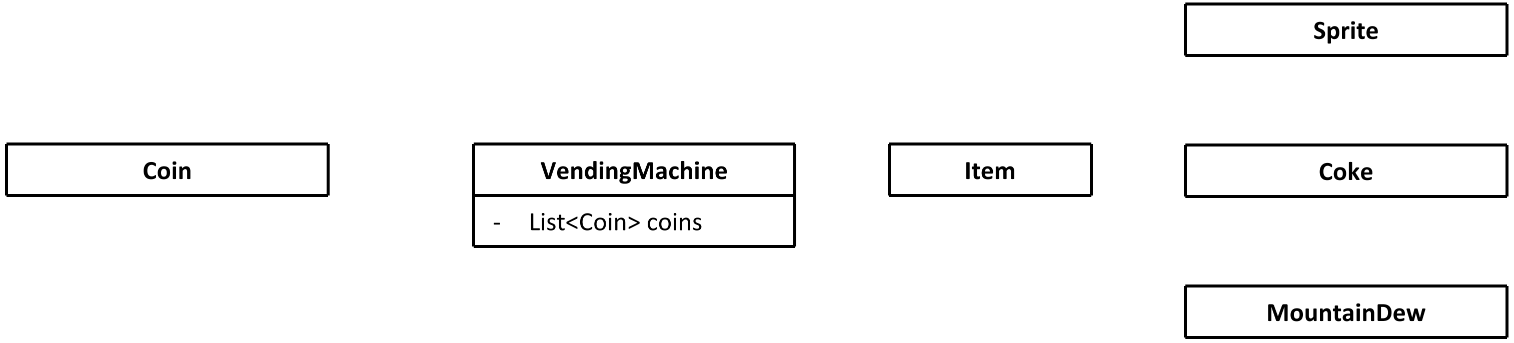
Core Object



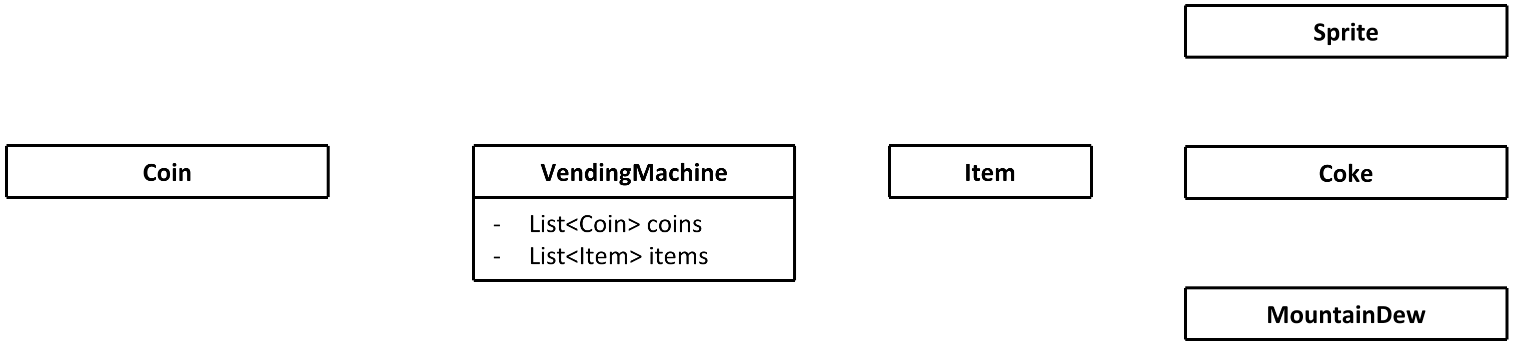
Core Object



Core Object


Sprite
Coin
VendingMachine
- List<Coin> coins
Item
Coke
MountainDew

Core Object


Sprite
Coin
VendingMachine
- List<Coin> coins
- List<Item> items
Item
Coke
MountainDew

Use cases

Vending machine

Use cases

Vending machine:

- Select item

Use cases

Vending machine:

- Select item
- Insert coin

Use cases

Vending machine:

- Select item
- Insert coin
- Execute transaction

Use cases

Vending machine:

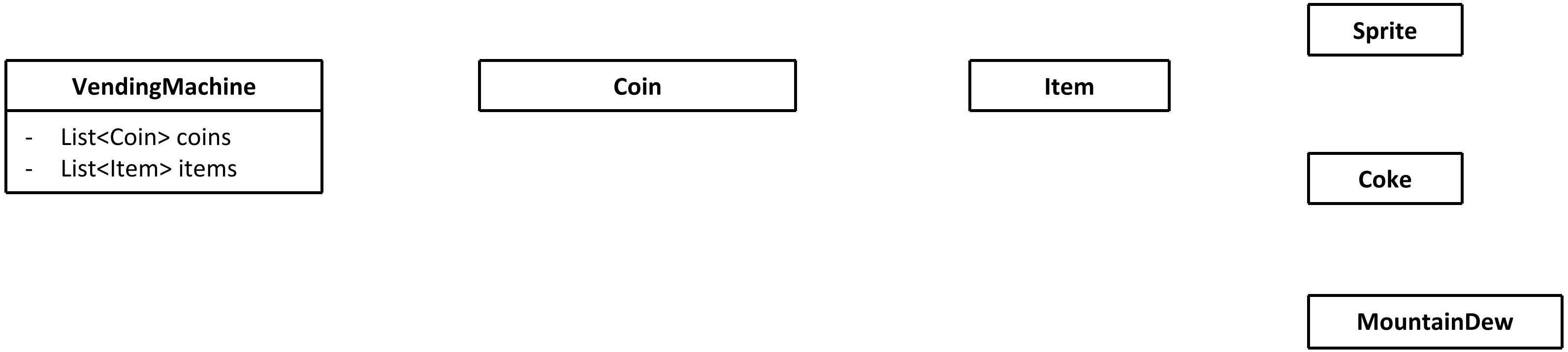
- Select item
- Insert coin
- Execute transaction
- Cancel transaction

Use cases

Vending machine:

- Select item
- Insert coin
- Execute transaction
- Cancel transaction
- Refill items

Classes



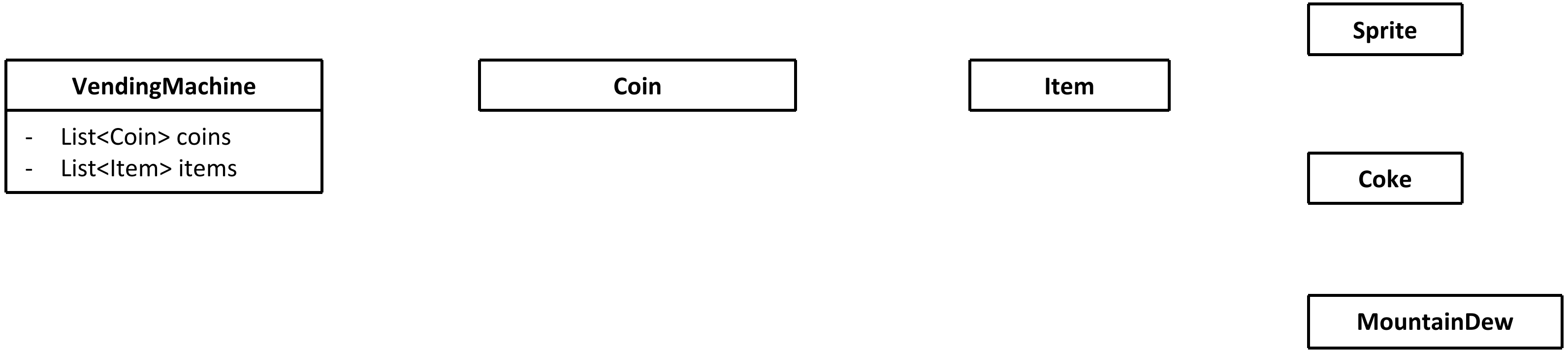
Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes

Use case: Select item

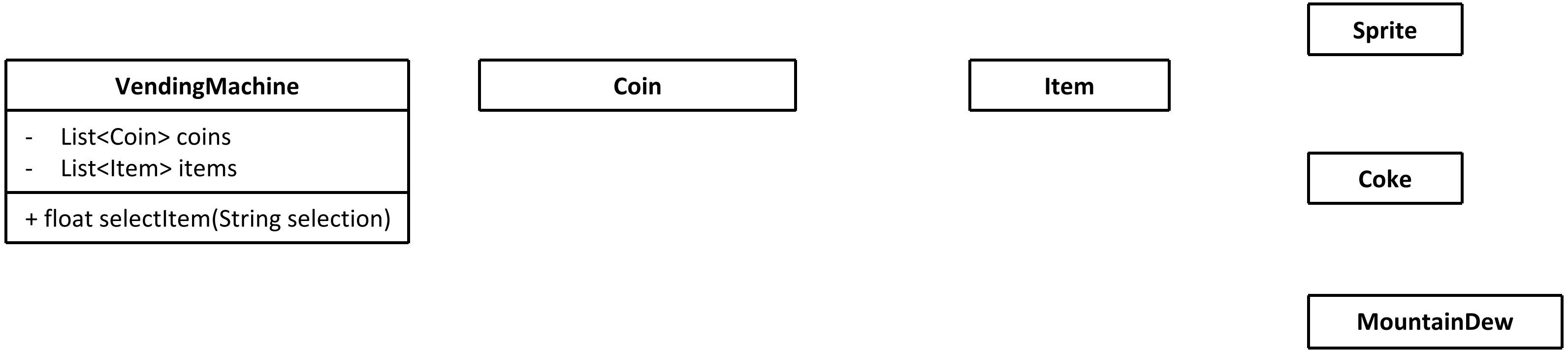
- Vending machine takes an external input, shows the price of that item

Classes



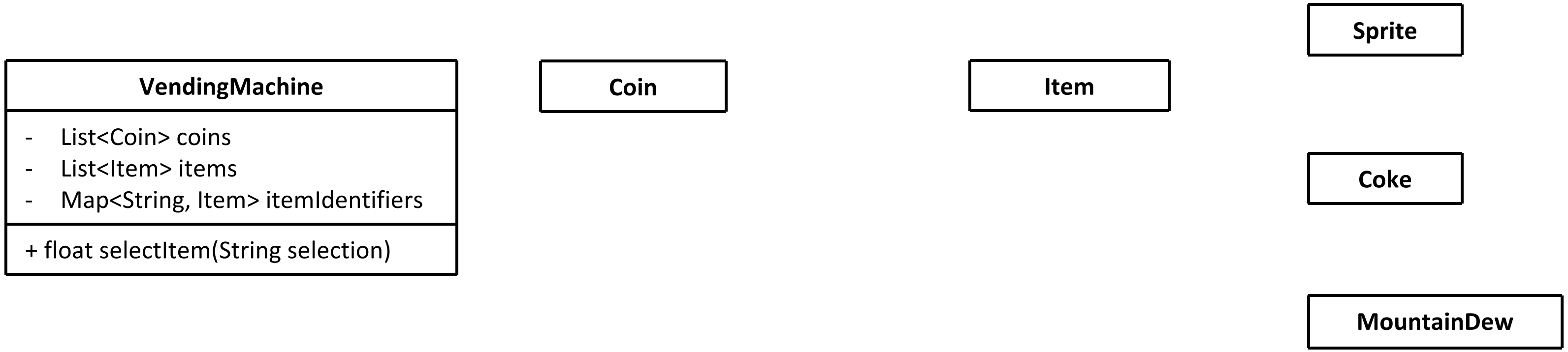
Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes



Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes



VendingMachine

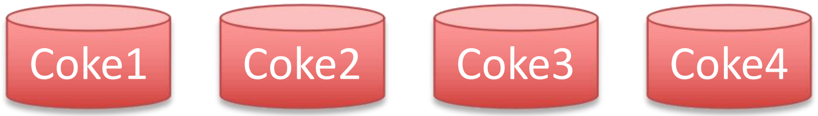
```
-   List<Coin> coins
-   List<Item> items
-   Map<String, Item> itemIdentifiers

+ float selectItem(String selection)
```

Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

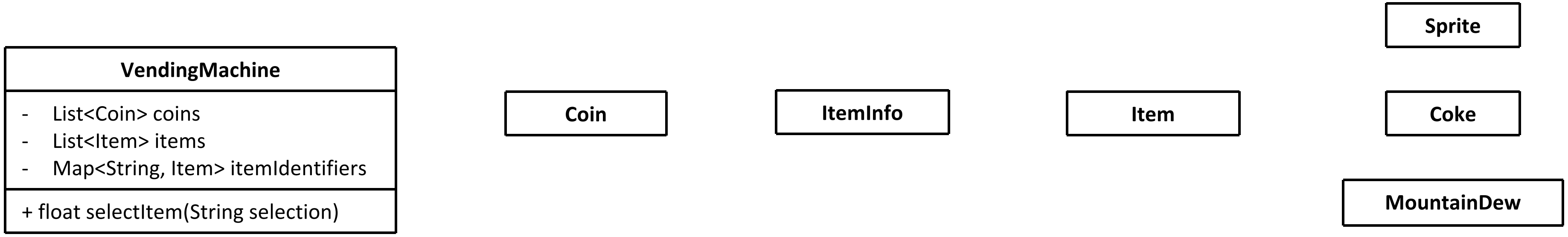
Challenge

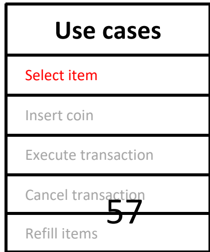
“A1” ->

Vending machine:

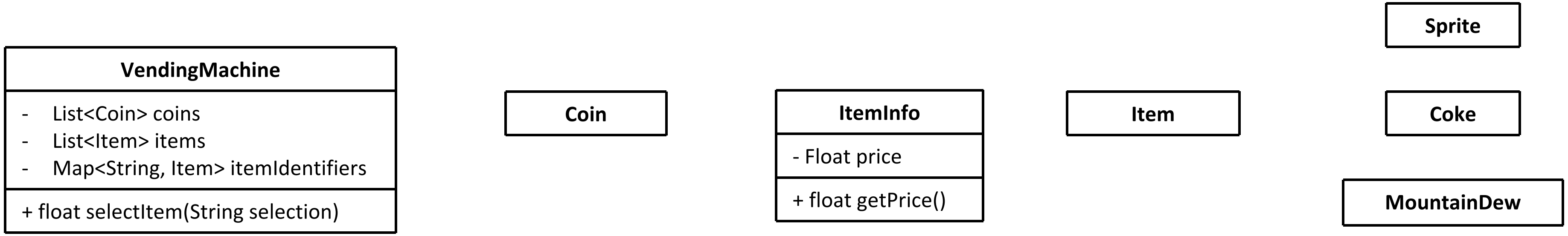


Classes


VendingMachine
- List<Coin> coins
- List<Item> items
- Map<String, Item> itemIdentifiers
+ float selectItem(String selection)
Coin
ItemInfo
Item
Sprite
Coke
MountainDew

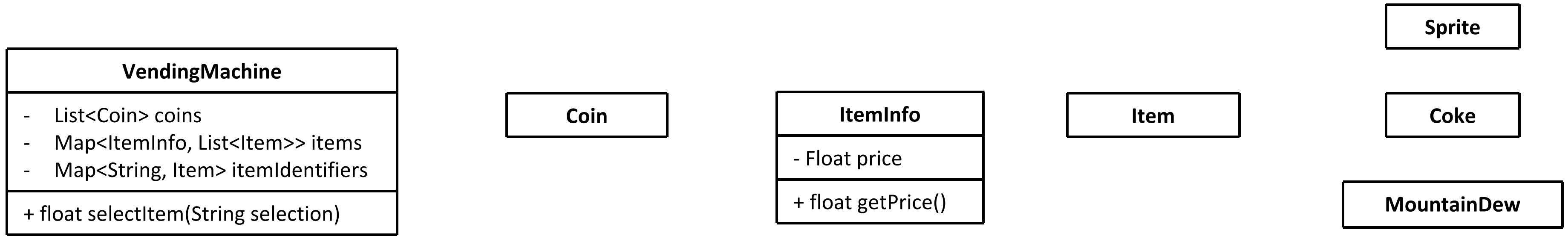

Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes



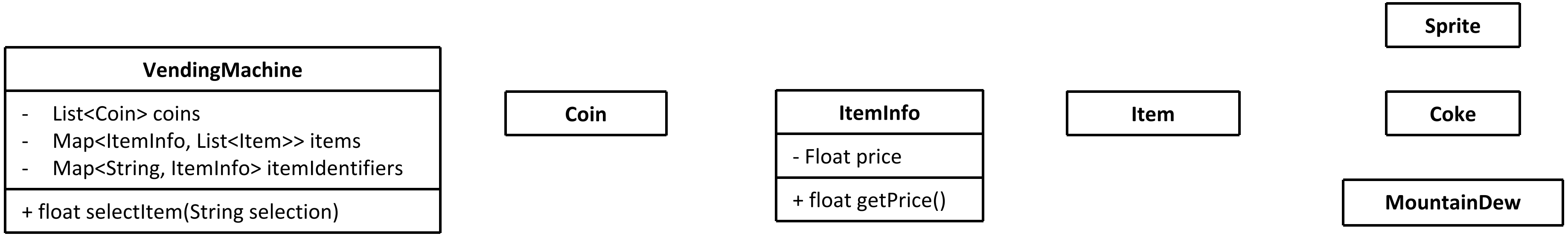
Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes


VendingMachine
- List<Coin> coins
- Map<ItemInfo, List<Item>> items
- Map<String, Item> itemIdentifiers
+ float selectItem(String selection)
Coin
ItemInfo
- Float price
+ float getPrice()
Item
Sprite
Coke
MountainDew

Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes



VendingMachine

- List<Coin> coins
- Map<ItemInfo, List<Item>> items
- Map<String, ItemInfo> itemIdentifiers

+ float selectItem(String selection)

Coin

ItemInfo

- Float price

+ float getPrice()

Item

Sprite

Coke

MountainDew

Use cases

Select item

Insert coin

Execute transaction

Cancel transaction

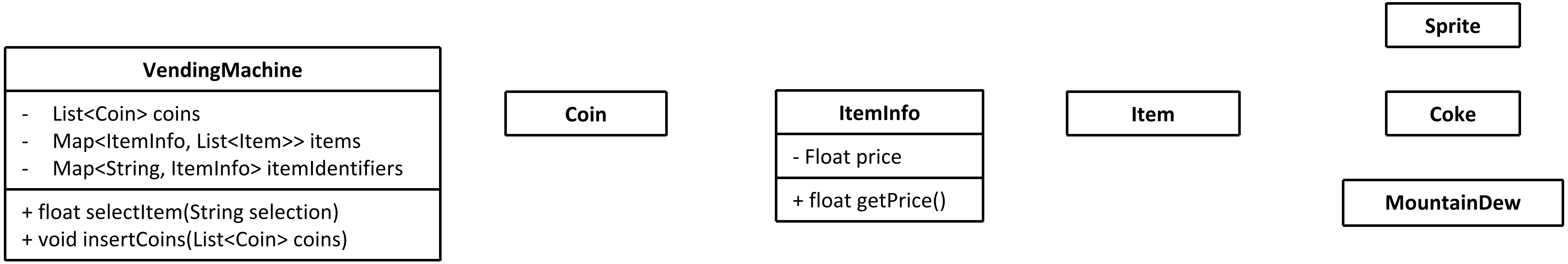
Refill items

Classes

Use case: Insert coin

- Insert a list of coins into vending machine

Classes



VendingMachine

- List<Coin> coins
- Map<ItemInfo, List<Item>> items
- Map<String, ItemInfo> itemIdentifiers

+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)

Coin

ItemInfo

- Float price

+ float getPrice()

Item

Sprite

Coke

MountainDew

Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes

Use case: Execute transaction

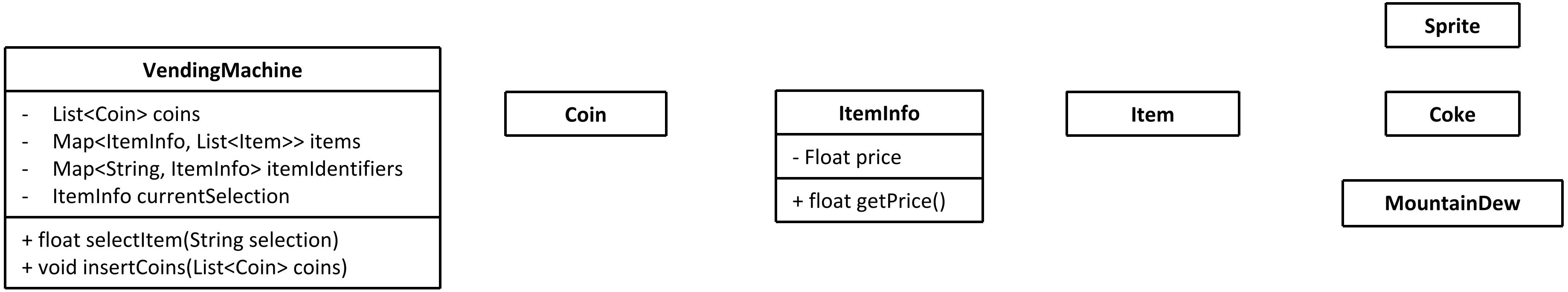
- Get the current selected item
- Compare the item price and inserted coins
- If not enough money, throw an exception
- Else, return the item purchased
- Refund if any

Classes

Use case: Execute transaction

- Get the current selected item

Classes


VendingMachine
- List<Coin> coins
- Map<ItemInfo, List<Item>> items
- Map<String, ItemInfo> itemIdentifiers
- ItemInfo currentSelection
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
Coin
ItemInfo
- Float price
+ float getPrice()
Item
Sprite
Coke
MountainDew

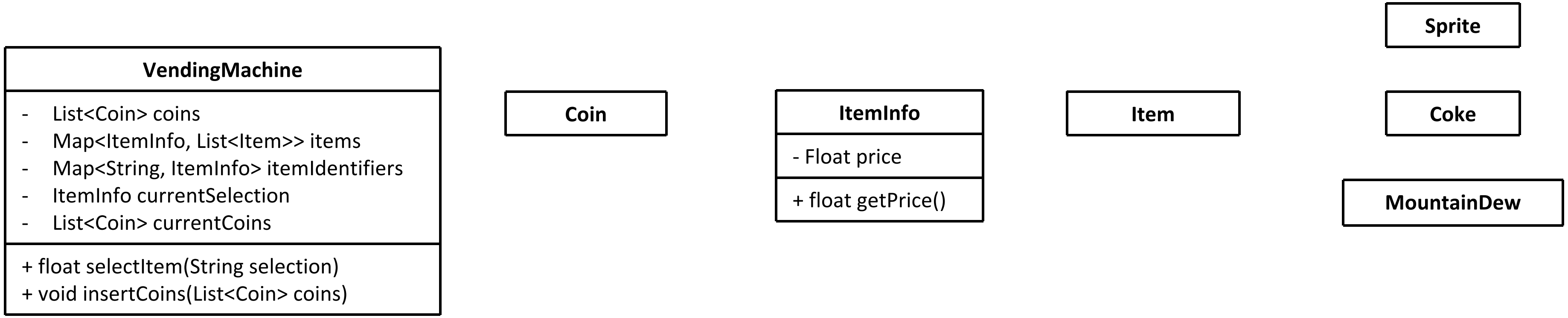
Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes

Use case: Execute transaction

- Get the current selected item
- Compare the item price and inserted coins

Classes



VendingMachine

- List<Coin> coins
- Map<ItemInfo, List<Item>> items
- Map<String, ItemInfo> itemIdentifiers
- ItemInfo currentSelection
- List<Coin> currentCoins

+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)

Coin

ItemInfo

- Float price

+ float getPrice()

Item

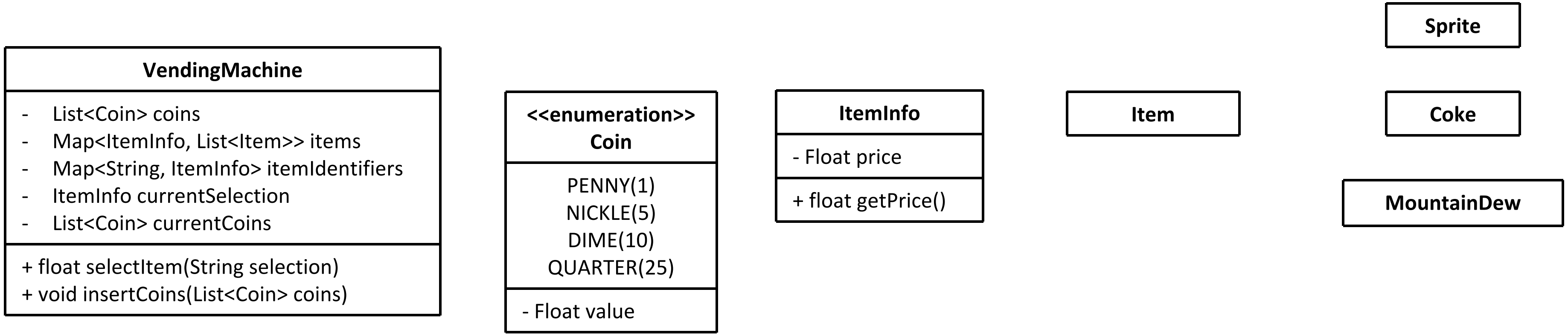
Sprite

Coke

MountainDew

Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes



VendingMachine

- List<Coin> coins
- Map<ItemInfo, List<Item>> items
- Map<String, ItemInfo> itemIdentifiers
- ItemInfo currentSelection
- List<Coin> currentCoins

+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)

<<enumeration>>
Coin

PENNY(1)
NICKLE(5)
DIME(10)
QUARTER(25)

- Float value

ItemInfo

- Float price

+ float getPrice()

Item

Sprite

Coke

MountainDew

Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes

Use case: Execute transaction

- Get the current selected item
- Compare the item price and inserted coins
- If not enough money, throw an exception

Classes

VendingMachine

- List<Coin> coins
- Map<ItemInfo, List<Item>> items
- Map<String, ItemInfo> itemIdentifiers
- ItemInfo currentSelection
- List<Coin> currentCoins

+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)

<<enumeration>>
Coin

PENNY(1)
NICKLE(5)
DIME(10)
QUARTER(25)

- Float value

ItemInfo

- Float price

+ float getPrice()

Item

Sprite

Coke

MountainDew

NotEnoughMoneyException

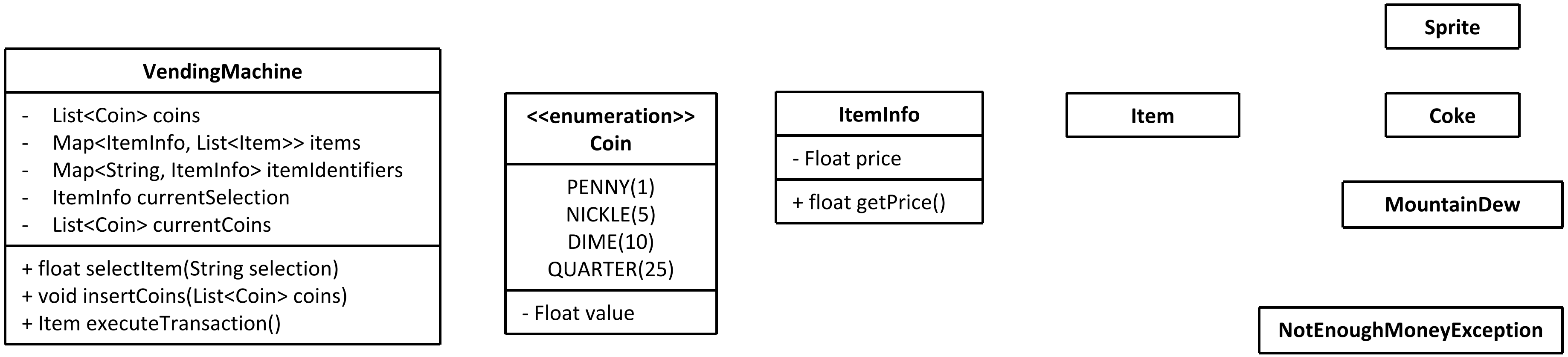
Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes

Use case: Execute transaction

- Get the current selected item
- Compare the item price and inserted coins
- If not enough money, throw an exception
- Else, return the item purchased

Classes



VendingMachine

- List<Coin> coins
- Map<ItemInfo, List<Item>> items
- Map<String, ItemInfo> itemIdentifiers
- ItemInfo currentSelection
- List<Coin> currentCoins

+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Item executeTransaction()

<<enumeration>>
Coin

PENNY(1)
NICKLE(5)
DIME(10)
QUARTER(25)

- Float value

ItemInfo

- Float price

+ float getPrice()

Item

Sprite

Coke

MountainDew

NotEnoughMoneyException

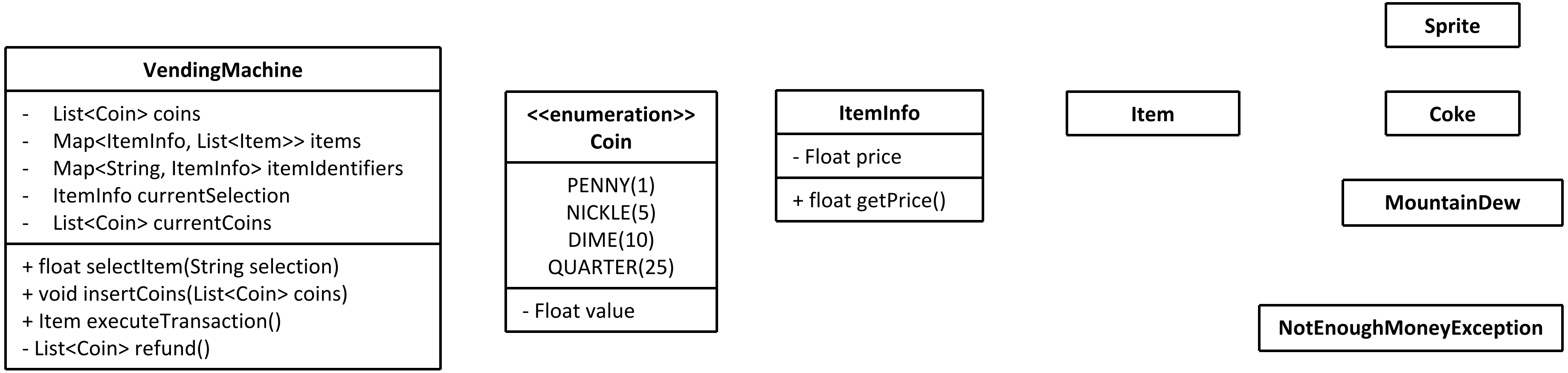
Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes

Use case: Execute transaction

- Get the current selected item
- Compare the item price and inserted coins
- If not enough money, throw an exception
- Else, return the item purchased
- Refund if any

Classes



VendingMachine

- List<Coin> coins
- Map<ItemInfo, List<Item>> items
- Map<String, ItemInfo> itemIdentifiers
- ItemInfo currentSelection
- List<Coin> currentCoins

```
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Item executeTransaction()
- List<Coin> refund()
```

**<<enumeration>>
Coin**

PENNY(1)
NICKLE(5)
DIME(10)
QUARTER(25)

- Float value

ItemInfo

- Float price

+ float getPrice()

Item

Sprite

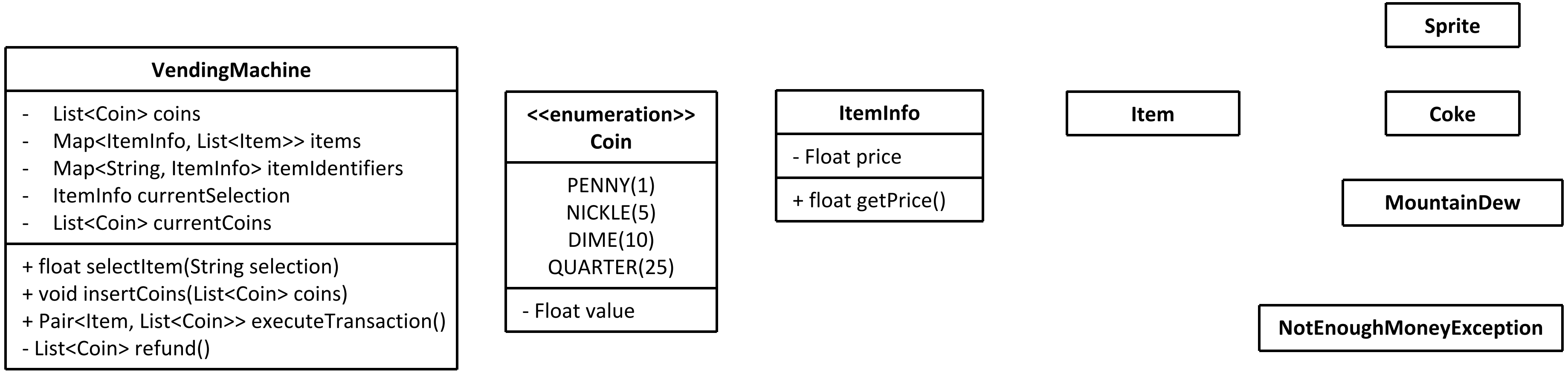
Coke

MountainDew

NotEnoughMoneyException

Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes



VendingMachine

```
-   List<Coin> coins
-   Map<ItemInfo, List<Item>> items
-   Map<String, ItemInfo> itemIdentifiers
-   ItemInfo currentSelection
-   List<Coin> currentCoins

+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Pair<Item, List<Coin>> executeTransaction()
- List<Coin> refund()
```

<<enumeration>> Coin

```
PENNY(1)
NICKLE(5)
DIME(10)
QUARTER(25)

- Float value
```

ItemInfo

```
- Float price

+ float getPrice()
```

Item

Sprite

Coke

MountainDew

NotEnoughMoneyException

Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Coin change

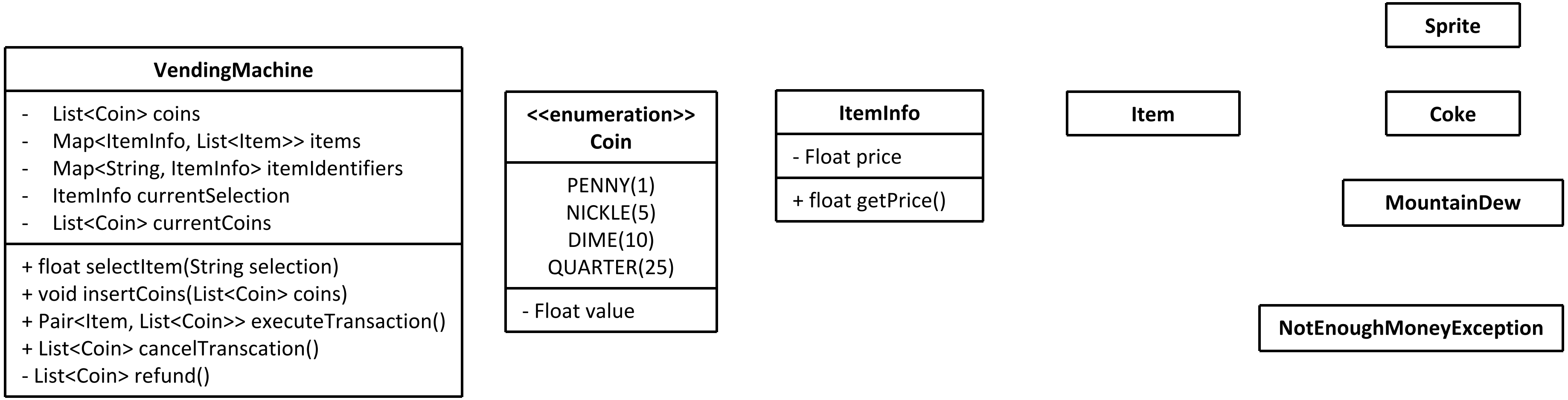
- http://www.cnblogs.com/grandyang/p/4840713.html

Classes

Use case: Cancel transaction

- Return the current coins that has been inserted

Classes



VendingMachine

- List<Coin> coins
- Map<ItemInfo, List<Item>> items
- Map<String, ItemInfo> itemIdentifiers
- ItemInfo currentSelection
- List<Coin> currentCoins

+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Pair<Item, List<Coin>> executeTransaction()
+ List<Coin> cancelTranscation()
- List<Coin> refund()

<<enumeration>>
Coin

PENNY(1)
NICKLE(5)
DIME(10)
QUARTER(25)

- Float value

ItemInfo

- Float price

+ float getPrice()

Item

Sprite

Coke

MountainDew

NotEnoughMoneyException

Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes

Use case: Refill items

- Refill items on top of current stock

Classes

VendingMachine
- List<Coin> coins
- Map<ItemInfo, List<Item>> items
- Map<String, ItemInfo> itemIdentifiers
- ItemInfo currentSelection
- List<Coin> currentCoins
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Pair<Item, List<Coin>> executeTransaction()
+ List<Coin> cancelTranscation()
+ void refillItems(List<Item> items)
- List<Coin> refund()

<<enumeration>> Coin
PENNY(1)
NICKLE(5)
DIME(10)
QUARTER(25)
- Float value

ItemInfo
- Float price
+ float getPrice()

Item
- ItemInfo info

Sprite

Coke

MountainDew

NotEnoughMoneyException

Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes

VendingMachine
- List<Coin> coins
- Map<String, ItemInfo> itemIdentifiers
- ItemInfo currentSelection
- List<Coin> currentCoins
- Map<ItemInfo, List<Item>> stock
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Pair<Item, List<Coin>> executeTransaction()
+ List<Coin> cancelTranscation()
+ void refillItems(List<Item> items)
- List<Coin> refund()

<<enumeration>> Coin
PENNY(1)
NICKLE(5)
DIME(10)
QUARTER(25)
- Float value

ItemInfo
- Float price
+ float getPrice()

Item
- ItemInfo info

Sprite

Coke

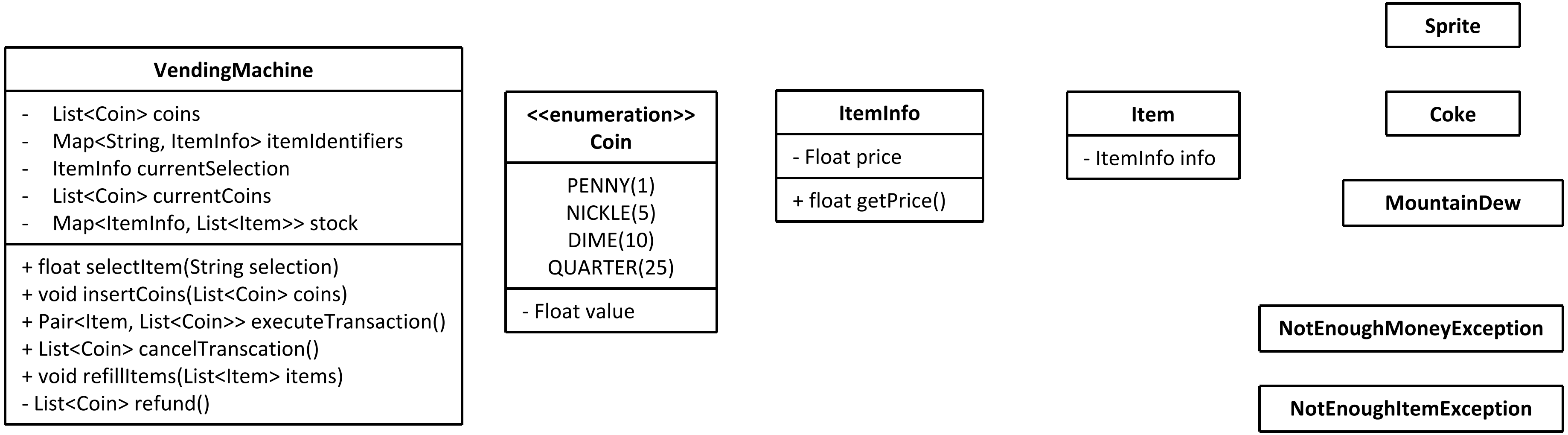
MountainDew

NotEnoughMoneyException

NotEnoughItemException

Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes - Final view



```
VendingMachine
-   List<Coin> coins
-   Map<String, ItemInfo> itemIdentifiers
-   ItemInfo currentSelection
-   List<Coin> currentCoins
-   Map<ItemInfo, List<Item>> stock
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Pair<Item, List<Coin>> executeTransaction()
+ List<Coin> cancelTranscation()
+ void refillItems(List<Item> items)
- List<Coin> refund()
```

```
<<enumeration>>
Coin
PENNY(1)
NICKLE(5)
DIME(10)
QUARTER(25)
- Float value
```

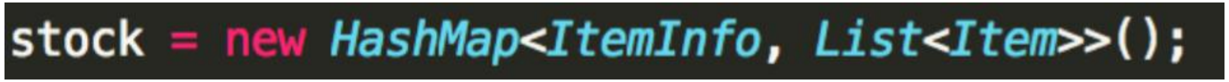
```
ItemInfo
- Float price
+ float getPrice()
```

```
Item
- ItemInfo info
```

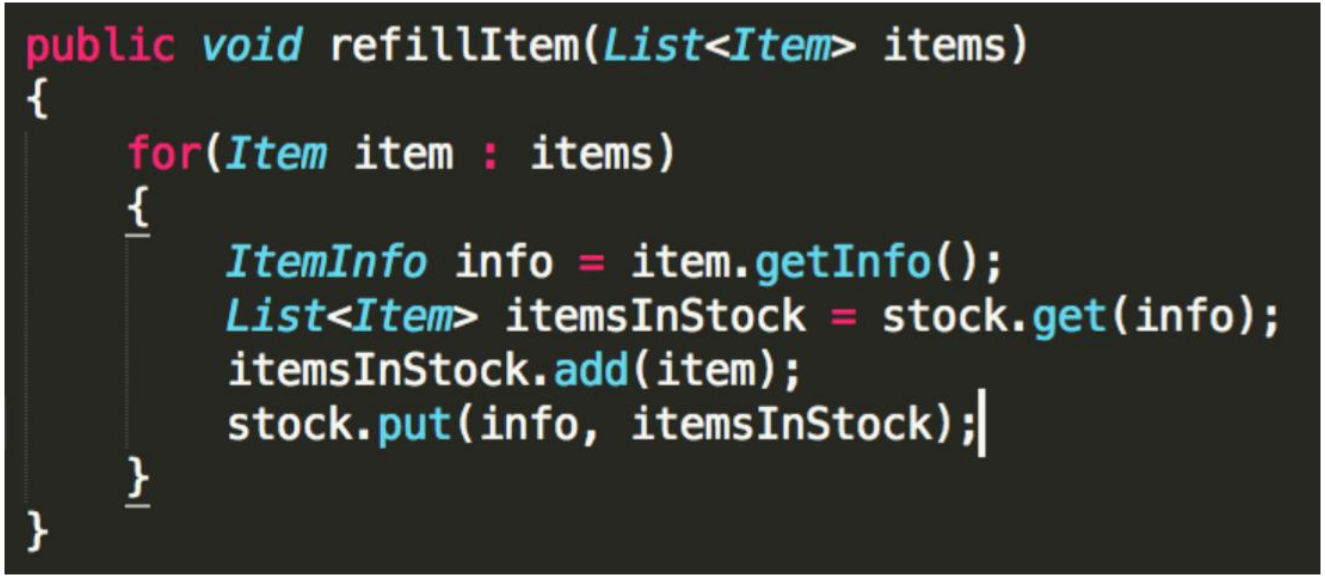
```
Sprite
Coke
MountainDew
NotEnoughMoneyException
NotEnoughItemException
```

Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Good practice

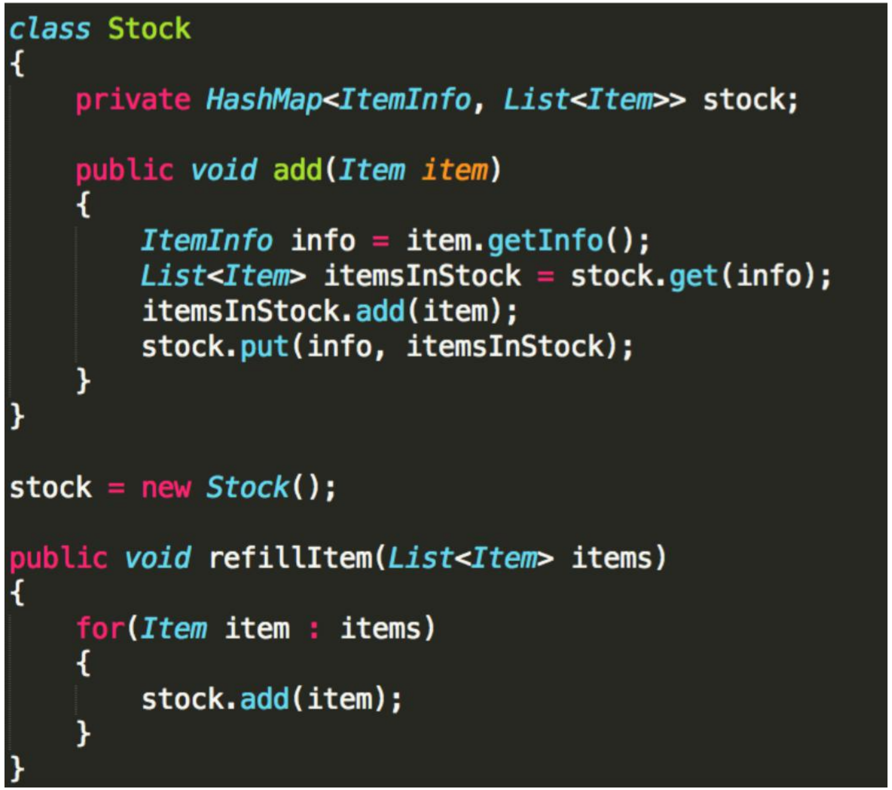


```
stock = new HashMap<ItemInfo, List<Item>>();
```



```
public void refillItem(List<Item> items)
{
    for(Item item : items)
    {
        ItemInfo info = item.getInfo();
        List<Item> itemsInStock = stock.get(info);
        itemsInStock.add(item);
        stock.put(info, itemsInStock);
    }
}
```

Good practice



```
class Stock
{
    private HashMap<ItemInfo, List<Item>> stock;

    public void add(Item item)
    {
        ItemInfo info = item.getInfo();
        List<Item> itemsInStock = stock.get(info);
        itemsInStock.add(item);
        stock.put(info, itemsInStock);
    }
}

stock = new Stock();

public void refillItem(List<Item> items)
{
    for(Item item : items)
    {
        stock.add(item);
    }
}
```

Good practice

```
VendingMachine
-   List<Coin> coins
-   List<Item> items
-   Map<String, ItemInfo> itemIdentifiers
-   ItemInfo currentSelection
-   List<Coin> currentCoins
-   Map<ItemInfo, List<Item>> stock
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Pair<Item, List<Coin>> executeTransaction()
+ List<Coin> cancelTranscation()
+ void refillItems(List<Item> items)
- List<Coin> refund()
```

```
<<enumeration>>
Coin
PENNY(1)
NICKLE(5)
DIME(10)
QUARTER(25)
- Float value
```

```
ItemInfo
- Float price
+ float getPrice()
```

```
Item
- ItemInfo info
```

```
Sprite
```

```
Coke
```

```
MountainDew
```

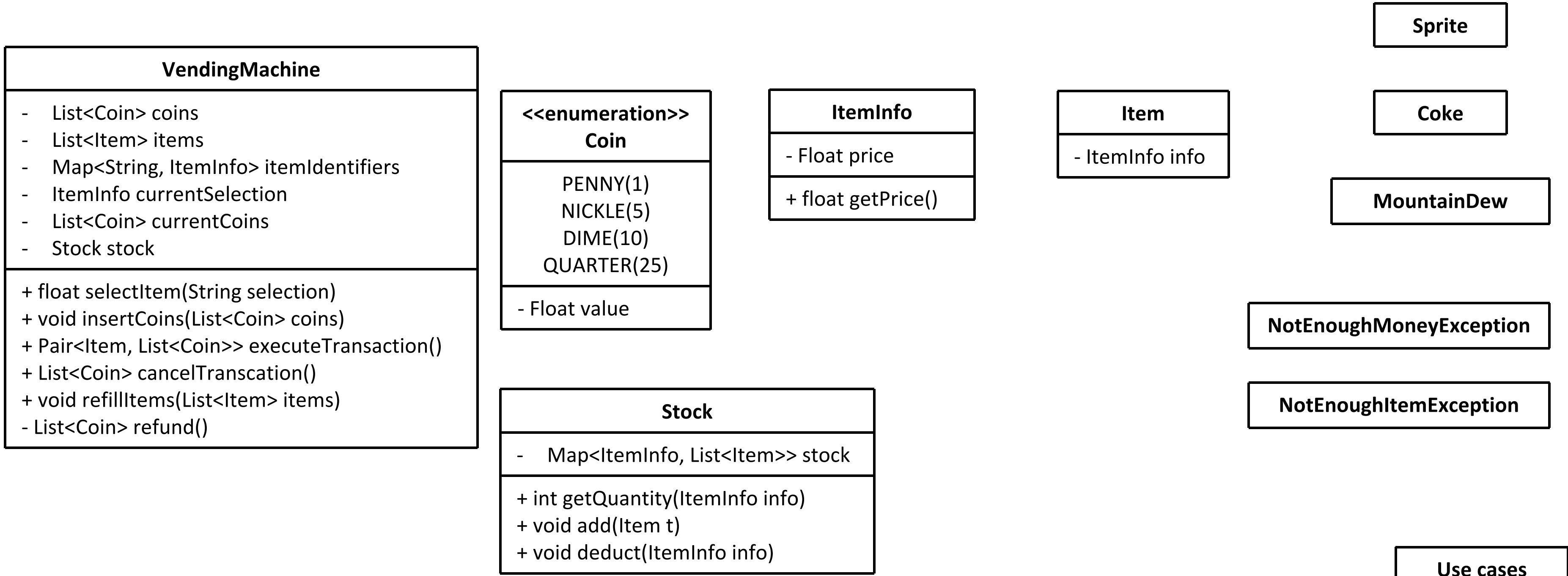
```
NotEnoughMoneyException
```

```
NotEnoughItemException
```

```
Stock
-   Map<ItemInfo, List<Item>> stock
+ int getQuantity(ItemInfo info)
+ void add(Item t)
+ void deduct(ItemInfo info)
```

Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Good practice



VendingMachine

```
-    List<Coin> coins
-    List<Item> items
-    Map<String, ItemInfo> itemIdentifiers
-    ItemInfo currentSelection
-    List<Coin> currentCoins
-    Stock stock

+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Pair<Item, List<Coin>> executeTransaction()
+ List<Coin> cancelTranscation()
+ void refillItems(List<Item> items)
- List<Coin> refund()
```

<<enumeration>> Coin

```
PENNY(1)
NICKLE(5)
DIME(10)
QUARTER(25)

- Float value
```

ItemInfo

```
- Float price

+ float getPrice()
```

Item

```
- ItemInfo info
```

Stock

```
-    Map<ItemInfo, List<Item>> stock

+ int getQuantity(ItemInfo info)
+ void add(Item t)
+ void deduct(ItemInfo info)
```

Sprite

Coke

MountainDew

NotEnoughMoneyException

NotEnoughItemException

Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Challenge

- For these use cases:

- Select item
- Insert coin
- Execute transaction
- Cancel transaction

Challenge

- For these use cases:

- Select item
- Insert coin
- Execute transaction
- Cancel transaction

What will happen if some item has been selected?

Challenge

- For these use cases:

- Select item : throws a selections has already been made
- Insert coin
- Execute transaction
- Cancel transaction

What will happen if some item has been selected?

Challenge

- For these use cases:

- Select item : throws a selections has already been made
- Insert coin : update current inserted value
- Execute transaction
- Cancel transaction

What will happen if some item has been selected?

Challenge

- For these use cases:

- Select item : throws a selections has already been made
- Insert coin : update current inserted value
- Execute transaction : Get selected item if money is enough
- Cancel transaction

What will happen if some item has been selected?

Challenge

- For these use cases:

- Select item : throws a selections has already been made
- Insert coin : update current inserted value
- Execute transaction : Get selected item if money is enough
- Cancel transaction : return money and empty selected item

What will happen if some item has been selected?

Challenge

- For these use cases:

- Select item
- Insert coin
- Execute transaction
- Cancel transaction

What will happen if none item has been selected?

Challenge

- For these use cases:

- Select item : make a selection
- Insert coin
- Execute transaction
- Cancel transaction

What will happen if none item has been selected?

Challenge

- For these use cases:

- Select item : make a selection
- Insert coin : throws to ask user make a selection first
- Execute transaction
- Cancel transaction

What will happen if none item has been selected?

Challenge

- For these use cases:

- Select item : make a selection
- Insert coin : throws to ask user make a selection first
- Execute transaction : throws to ask user to make a selection first
- Cancel transaction

What will happen if none item has been selected?

Challenge

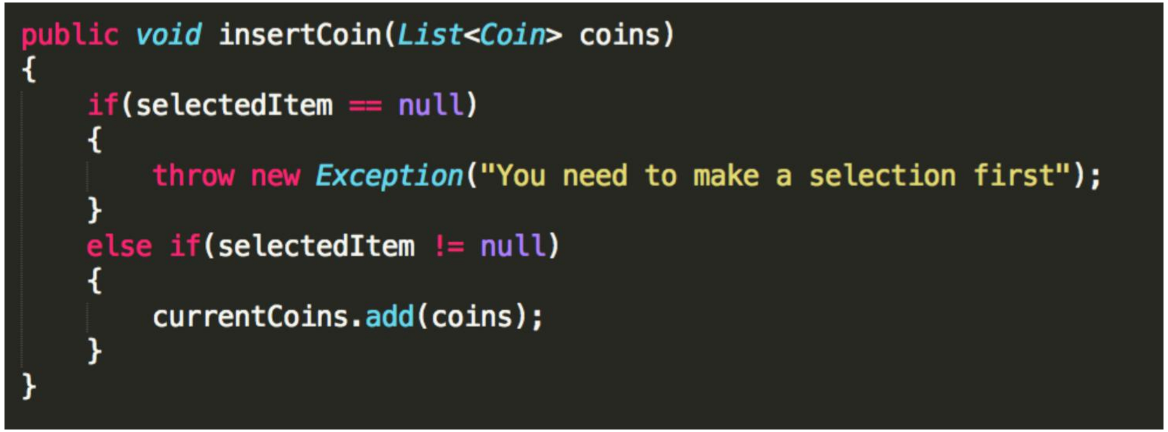
- For these use cases:

- Select item : make a selection
- Insert coin : throws to ask user make a selection first
- Execute transaction : throws to ask user to make a selection first
- Cancel transaction : maybe not doing anything or throw

What will happen if none item has been selected?

Challenge

- Insert coin



```
public void insertCoin(List<Coin> coins)
{
    if(selectedItem == null)
    {
        throw new Exception("You need to make a selection first");
    }
    else if(selectedItem != null)
    {
        currentCoins.add(coins);
    }
}
```

Challenge

- 我们刚刚考虑了HAS_SELECTION 和 NO_SELECTION 的情况

Challenge

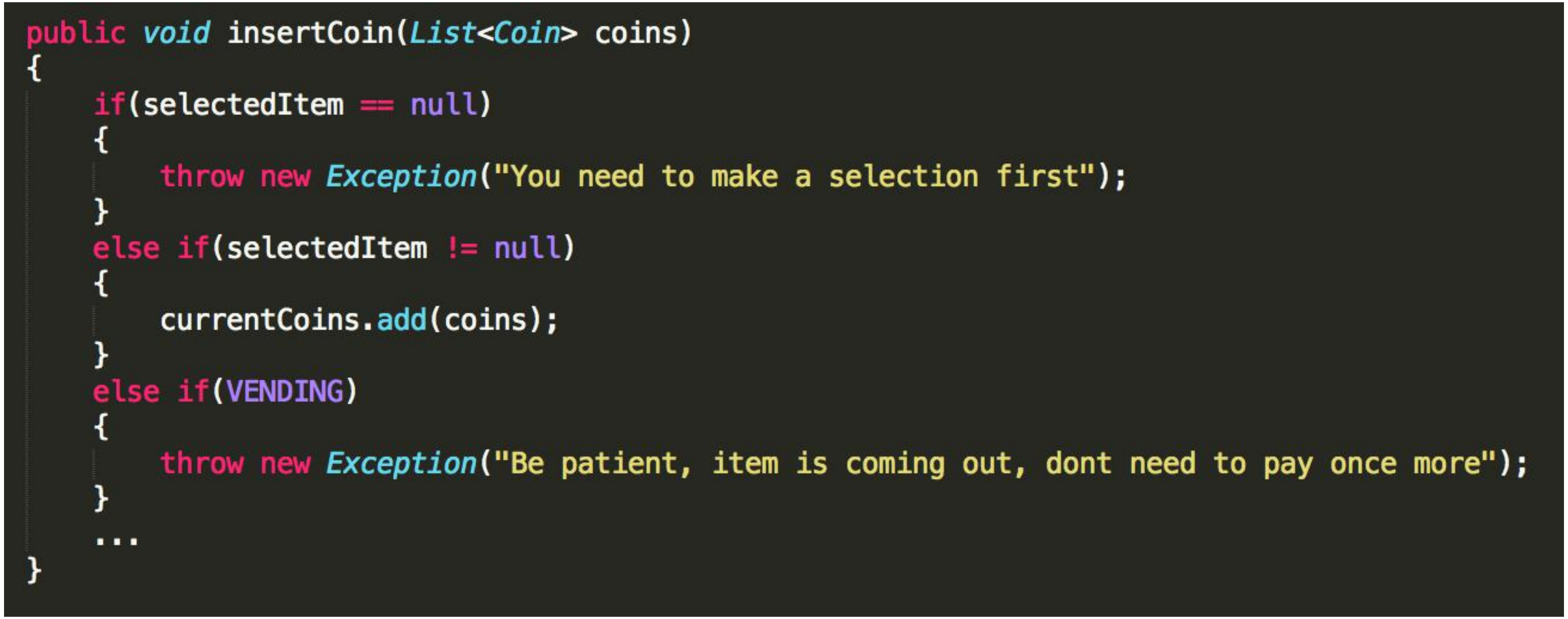
- 我们刚刚考虑了HAS_SELECTION 和 NO_SELECTION 的情况

- 那么对于：

- COINS_INSERTED
- VENDING

应该怎么办？

Challenge



```
public void insertCoin(List<Coin> coins)
{
    if(selectedItem == null)
    {
        throw new Exception("You need to make a selection first");
    }
    else if(selectedItem != null)
    {
        currentCoins.add(coins);
    }
    else if(VENDING)
    {
        throw new Exception("Be patient, item is coming out, dont need to pay once more");
    }
    ...
}
```

Challenge

- State Design Pattern

Challenge

- State Design Pattern

States:

- HAS_SELECTION
- NO_SELECTION
- COINS_INSERTED
-VENDING

Challenge

- State Design Pattern

State related actions:

- select item
- insert coin
- execute transaction
- cancel transaction

Classes

VendingMachine
- List<Coin> coins
- List<Item> items
- Map<String, ItemInfo> itemIdentifiers
- ItemInfo currentSelection
- List<Coin> currentCoins
- Map<ItemInfo, List<Item>> stock
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Item executeTransaction()
+ List<Coin> cancelTranscation()
+ void refillItems(List<Item> items)
- List<Coin> refund()

Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes

VendingMachine

- List<Coin> coins
- List<Item> items
- Map<String, ItemInfo> itemIdentifiers
- ItemInfo currentSelection
- List<Coin> currentCoins
- Map<ItemInfo, List<Item>> stock

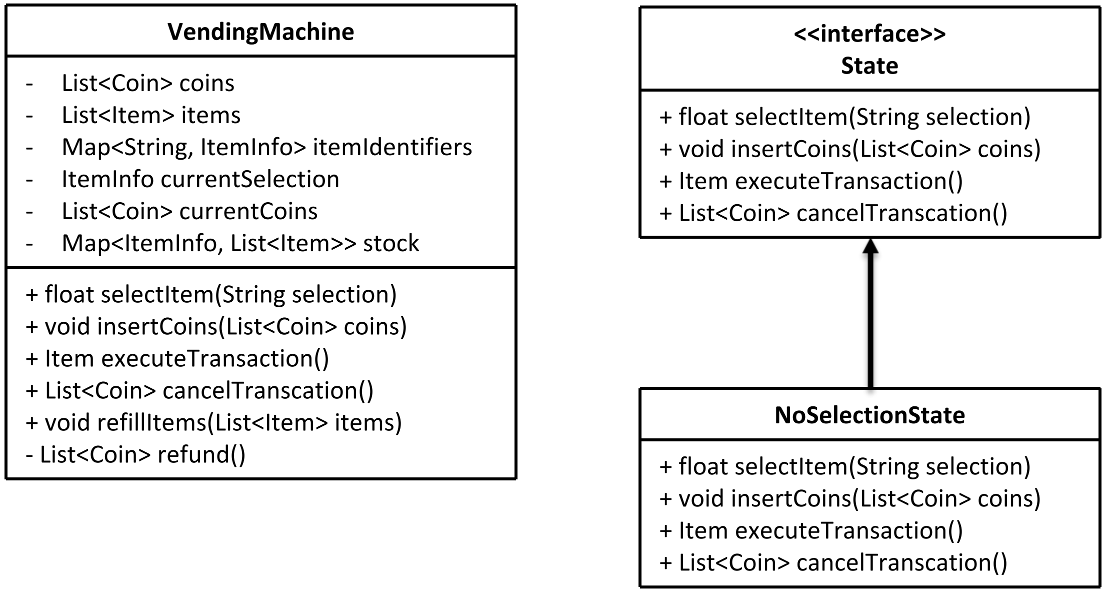
```
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Item executeTransaction()
+ List<Coin> cancelTranscation()
+ void refillItems(List<Item> items)
- List<Coin> refund()
```

<<interface>>
State

```
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Item executeTransaction()
+ List<Coin> cancelTranscation()
```

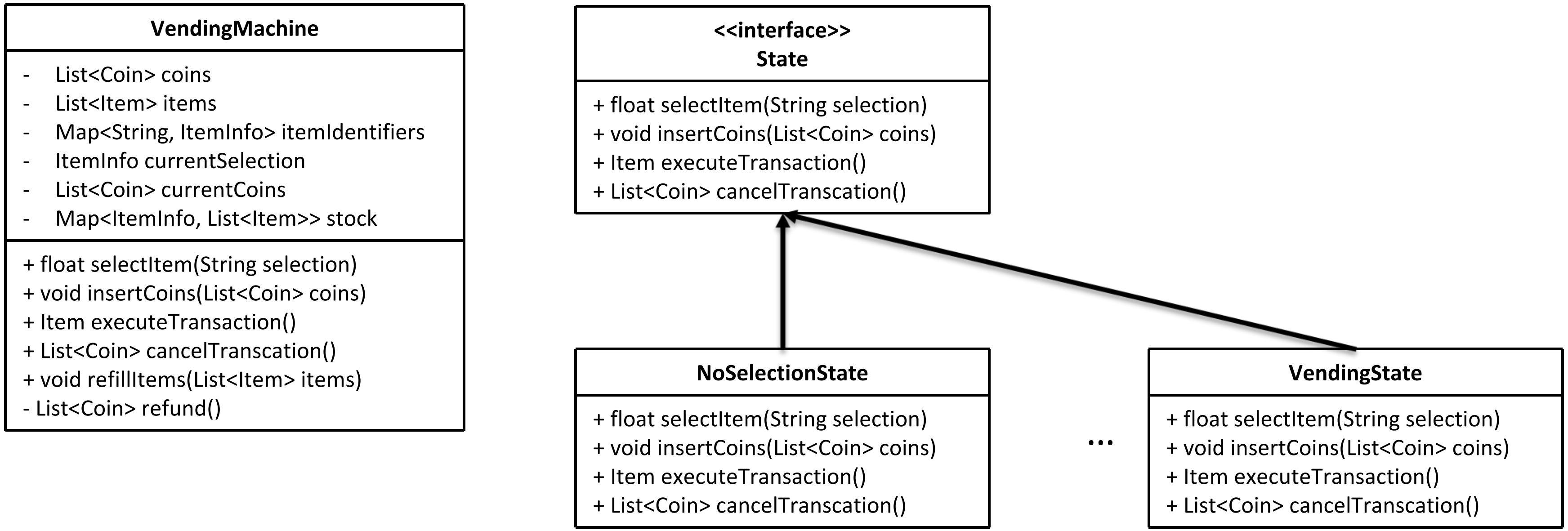
Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes


VendingMachine
- List<Coin> coins
- List<Item> items
- Map<String, ItemInfo> itemIdentifiers
- ItemInfo currentSelection
- List<Coin> currentCoins
- Map<ItemInfo, List<Item>> stock
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Item executeTransaction()
+ List<Coin> cancelTranscation()
+ void refillItems(List<Item> items)
- List<Coin> refund()
<<interface>>
State
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Item executeTransaction()
+ List<Coin> cancelTranscation()
NoSelectionState
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Item executeTransaction()
+ List<Coin> cancelTranscation()

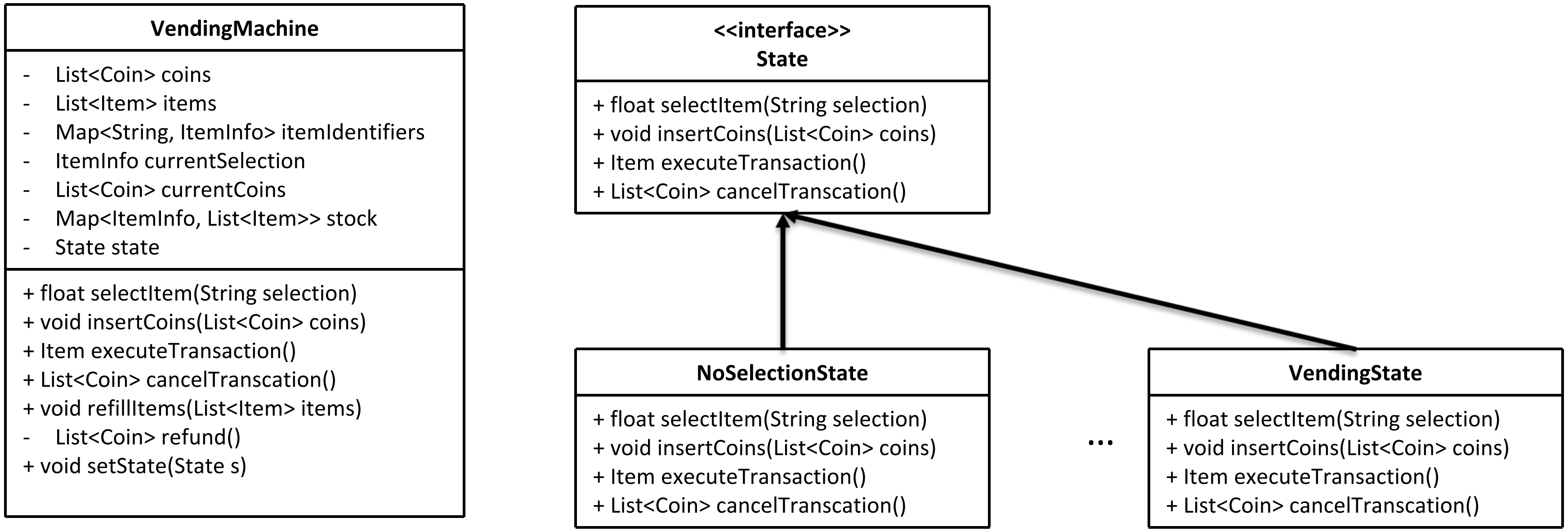
Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes


VendingMachine
- List<Coin> coins
- List<Item> items
- Map<String, ItemInfo> itemIdentifiers
- ItemInfo currentSelection
- List<Coin> currentCoins
- Map<ItemInfo, List<Item>> stock
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Item executeTransaction()
+ List<Coin> cancelTranscation()
+ void refillItems(List<Item> items)
- List<Coin> refund()
<<interface>>
State
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Item executeTransaction()
+ List<Coin> cancelTranscation()
NoSelectionState
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Item executeTransaction()
+ List<Coin> cancelTranscation()
...
VendingState
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Item executeTransaction()
+ List<Coin> cancelTranscation()

Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes



```
VendingMachine
-   List<Coin> coins
-   List<Item> items
-   Map<String, ItemInfo> itemIdentifiers
-   ItemInfo currentSelection
-   List<Coin> currentCoins
-   Map<ItemInfo, List<Item>> stock
-   State state
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Item executeTransaction()
+ List<Coin> cancelTranscation()
+ void refillItems(List<Item> items)
-   List<Coin> refund()
+ void setState(State s)
```

```
<<interface>>
State
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Item executeTransaction()
+ List<Coin> cancelTranscation()
```

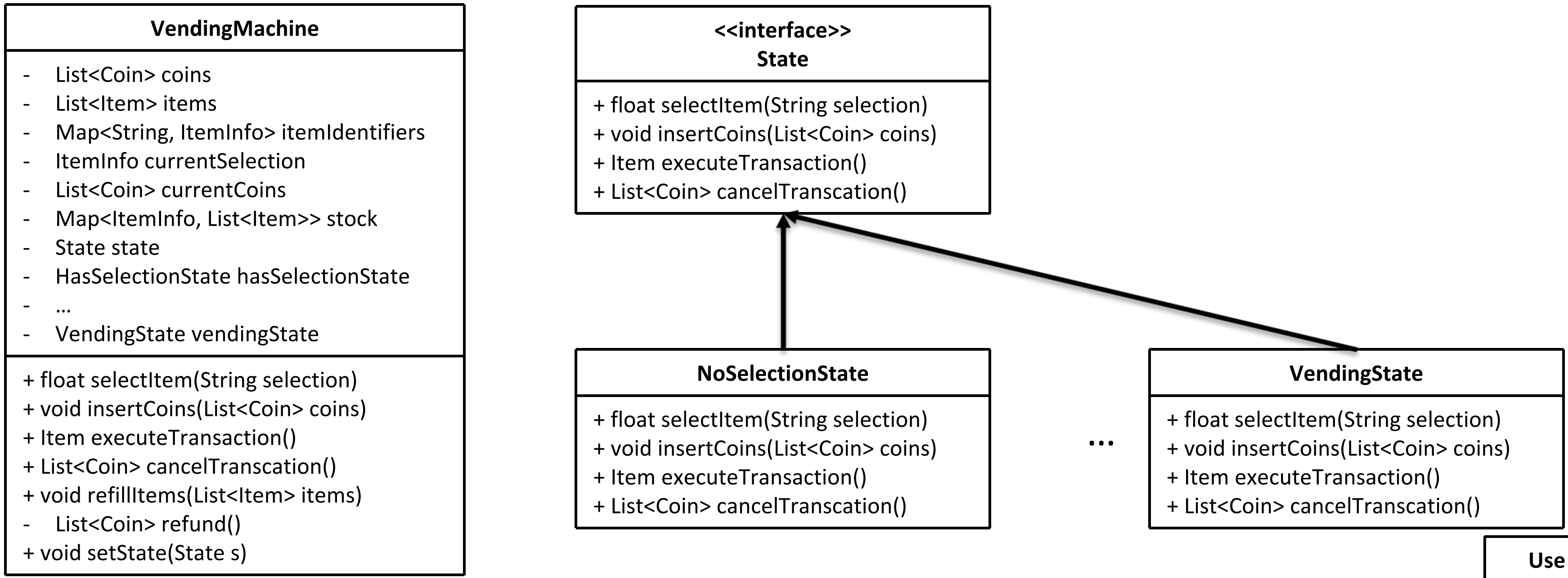
```
NoSelectionState
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Item executeTransaction()
+ List<Coin> cancelTranscation()
```

…

```
VendingState
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Item executeTransaction()
+ List<Coin> cancelTranscation()
```

Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Classes



```
VendingMachine
-    List<Coin> coins
-    List<Item> items
-    Map<String, ItemInfo> itemIdentifiers
-    ItemInfo currentSelection
-    List<Coin> currentCoins
-    Map<ItemInfo, List<Item>> stock
-    State state
-    HasSelectionState hasSelectionState
-    …
-    VendingState vendingState
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Item executeTransaction()
+ List<Coin> cancelTranscation()
+ void refillItems(List<Item> items)
-    List<Coin> refund()
+ void setState(State s)
```

```
<<interface>>
State
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Item executeTransaction()
+ List<Coin> cancelTranscation()
```

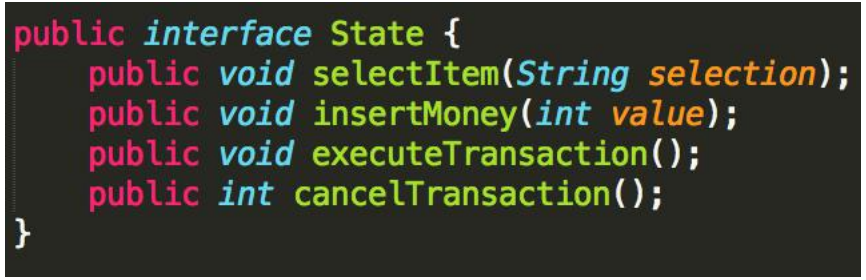
```
NoSelectionState
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Item executeTransaction()
+ List<Coin> cancelTranscation()
```

…

```
VendingState
+ float selectItem(String selection)
+ void insertCoins(List<Coin> coins)
+ Item executeTransaction()
+ List<Coin> cancelTranscation()
```

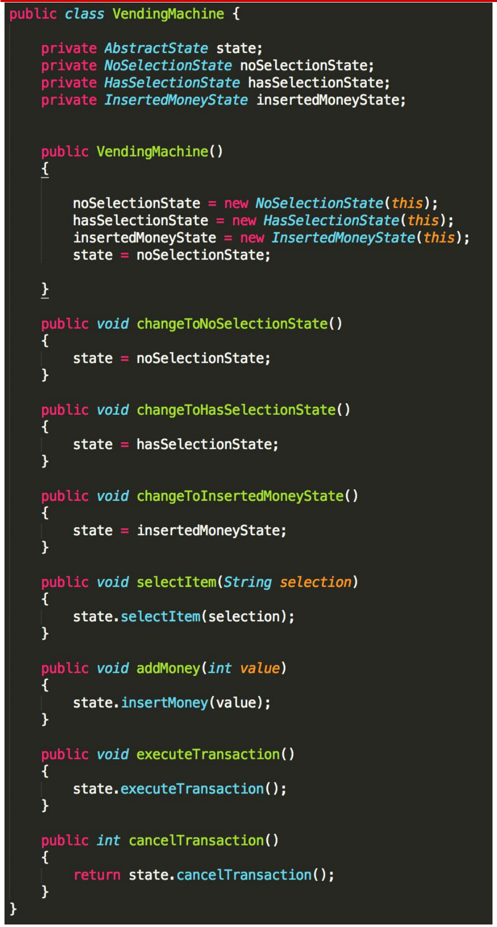
Use cases
Select item
Insert coin
Execute transaction
Cancel transaction
Refill items

Vending machine

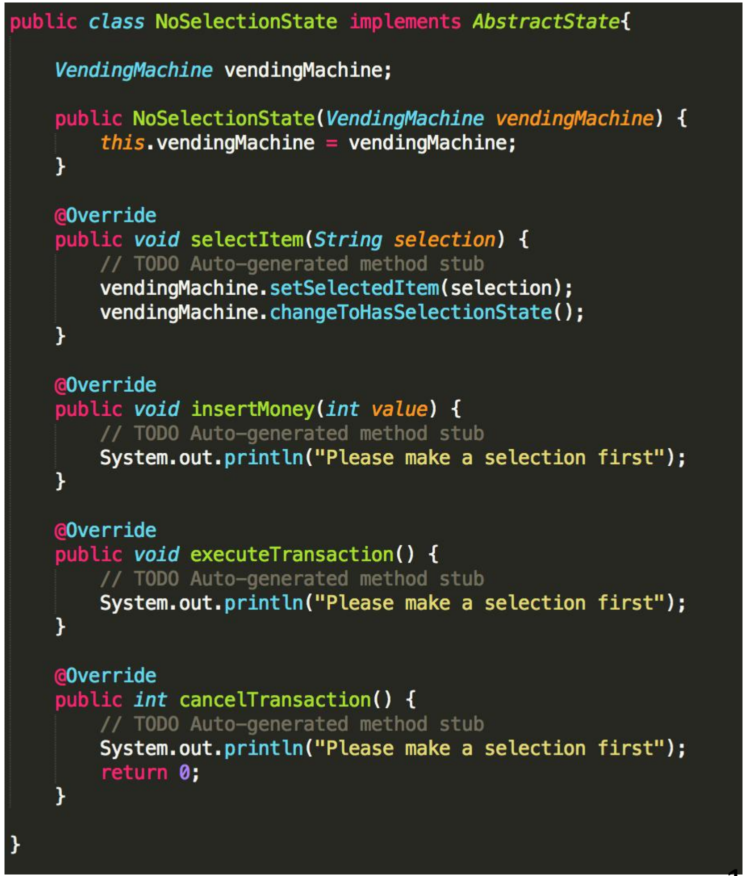


```
public interface State {
    public void selectItem(String selection);
    public void insertMoney(int value);
    public void executeTransaction();
    public int cancelTransaction();
}
```

Vending machine



```java
public class VendingMachine {

    private AbstractState state;
    private NoSelectionState noSelectionState;
    private HasSelectionState hasSelectionState;
    private InsertedMoneyState insertedMoneyState;


    public VendingMachine()
    {

        noSelectionState = new NoSelectionState(this);
        hasSelectionState = new HasSelectionState(this);
        insertedMoneyState = new InsertedMoneyState(this);
        state = noSelectionState;

    }
    public void changeToNoSelectionState()
    {
        state = noSelectionState;
    }

    public void changeToHasSelectionState()
    {
        state = hasSelectionState;
    }

    public void changeToInsertedMoneyState()
    {
        state = insertedMoneyState;
    }

    public void selectItem(String selection)
    {
        state.selectItem(selection);
    }

    public void addMoney(int value)
    {
        state.insertMoney(value);
    }

    public void executeTransaction()
    {
        state.executeTransaction();
    }

    public int cancelTransaction()
    {
        return state.cancelTransaction();
    }
}
```



```java
public class NoSelectionState implements AbstractState{

    VendingMachine vendingMachine;

    public NoSelectionState(VendingMachine vendingMachine) {
        this.vendingMachine = vendingMachine;
    }

    @Override
    public void selectItem(String selection) {
        // TODO Auto-generated method stub
        vendingMachine.setSelectedItem(selection);
        vendingMachine.changeToHasSelectionState();
    }

    @Override
    public void insertMoney(int value) {
        // TODO Auto-generated method stub
        System.out.println("Please make a selection first");
    }

    @Override
    public void executeTransaction() {
        // TODO Auto-generated method stub
        System.out.println("Please make a selection first");
    }

    @Override
    public int cancelTransaction() {
        // TODO Auto-generated method stub
        System.out.println("Please make a selection first");
        return 0;
    }

}
```

Real life object

- 难度不大

- 从Input / Output 考虑
- 继承关系
- 考虑Exception
- Design pattern if possible

Coffee maker

STARBUCKS COFFEE

Clarify

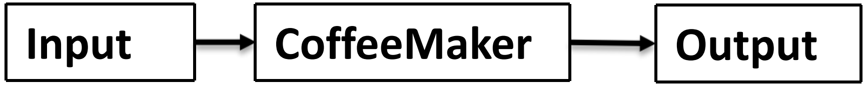
- What

关键字： Coffee maker

Clarify

- What

关键字： Coffee maker



Clarify

ELETTA
DēLonghi

OPEN
CARAFE
FILL TANK
STRENGTH
TIME NOW
POWER
SINGLE CUP
CARAFE
STRENGTH FLAVOR
PROGRAM SET CLOCK
START CANCEL

KEURIG

Clarify

- Input

Clarify

- Output

Types of Coffee
Water
Cream
Coffee
Milk
Chocolate
Expresso
Iced Coffee
American Coffee
Macchiato
Venice Coffee
Latte
Coffee Mocca
Coffee with Milk
Coffee with Cream
Capuccino
Chocolate Coffee
Long Black
designed by freepik.com

Clarify

- 对于本题

Input: Coffee packs

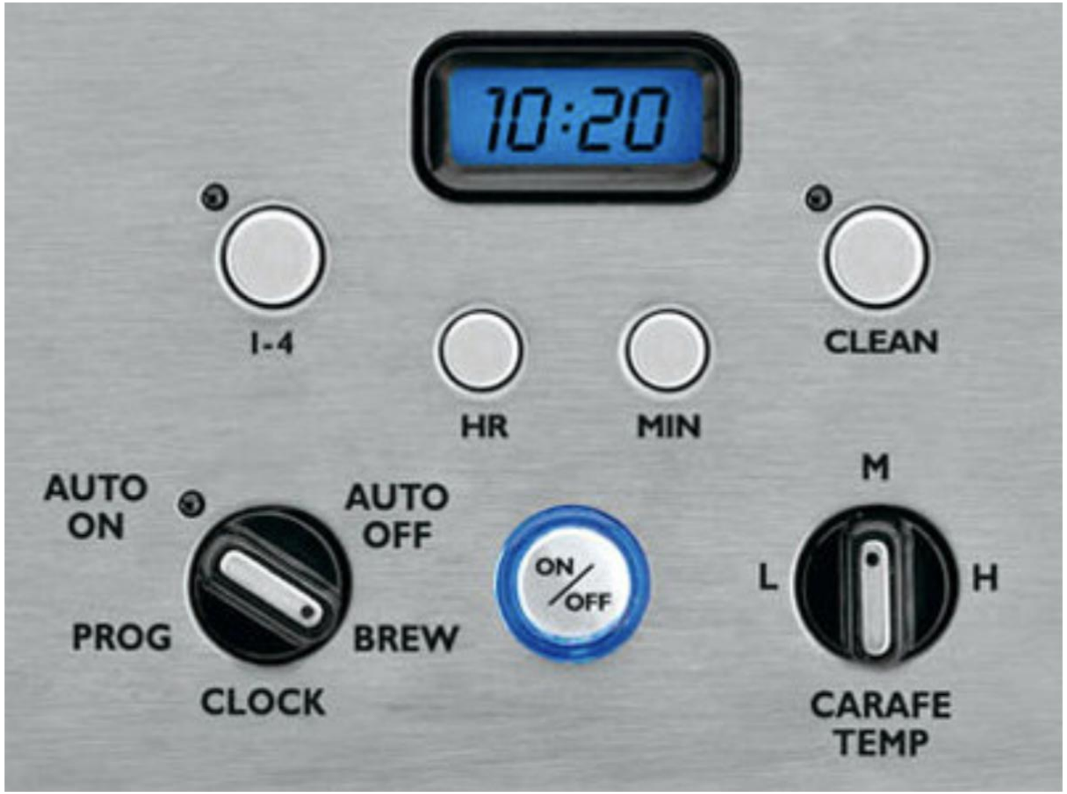
Output: Expresso

Clarify

- How

Clarify

- How


10:20
I-4
CLEAN
HR
MIN
AUTO ON
AUTO OFF
PROG
BREW
CLOCK
ON/OFF
M
L
H
CARAFE TEMP

Clarify

- What are the functions that out coffee maker supports?

Clarify

- 对于本题：

- Brew

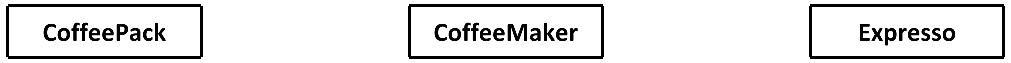
Core object

CoffeeMaker

Core object

CoffeePack
CoffeeMaker

Core object

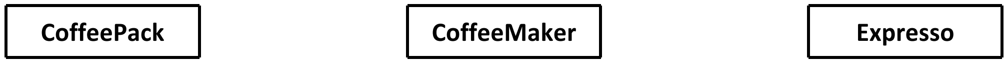


Use cases

- Coffee maker

- Brew

Classes

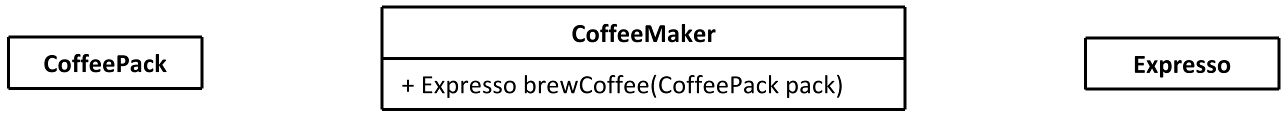


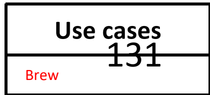
Classes

- Use case: Brew

Coffee machine expected to use a coffee pack to get expresso coffee

Classes


CoffeePack
CoffeeMaker
+ Expresso brewCoffee(CoffeePack pack)
Expresso

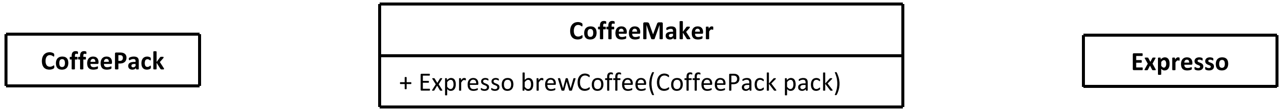

Use cases
Brew

Challenge

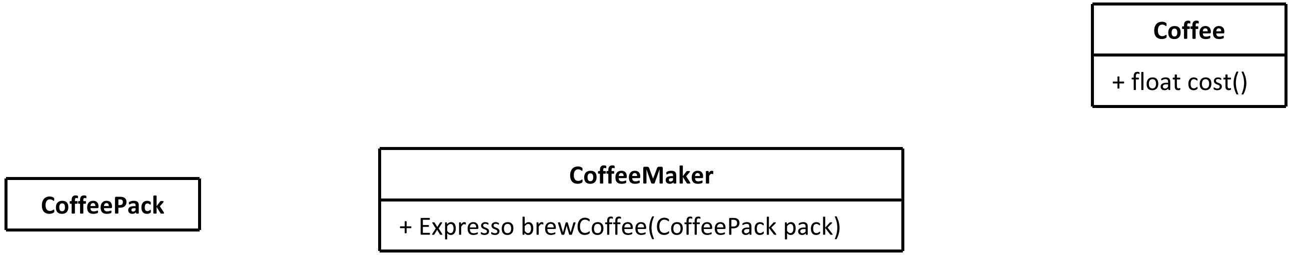
Types of Coffee
Water
Cream
Coffee
Milk
Chocolate
Expresso
Iced Coffee
American Coffee
Macchiato
Venice Coffee
Latte
Coffee Mocca
Coffee with Milk
Coffee with Cream
Capuccino
Chocolate Coffee
Long Black
designed by freepik.com

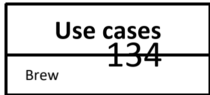
如果需要能制作出多种咖啡
(价格不同)，需要怎么做？

Classes

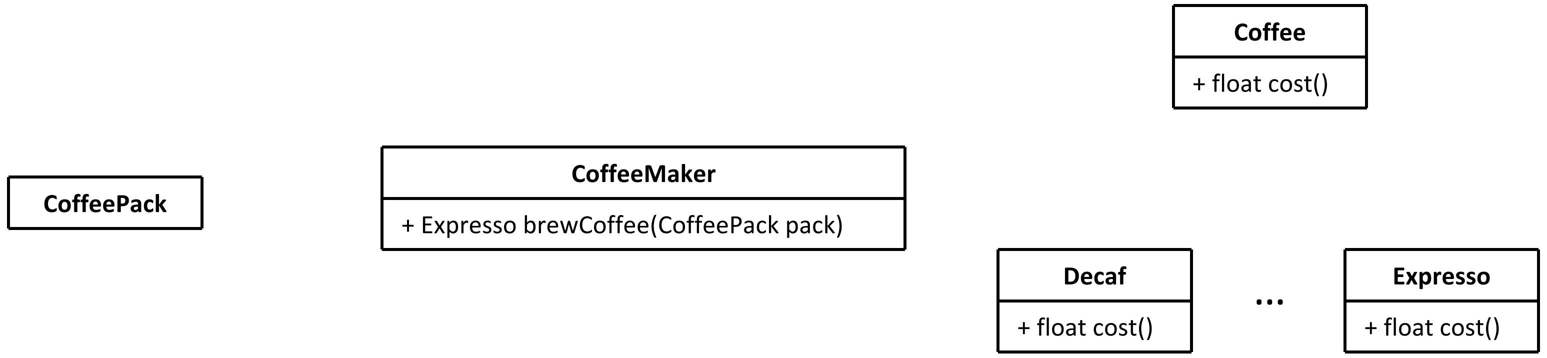

CoffeePack
CoffeeMaker
+ Expresso brewCoffee(CoffeePack pack)
Expresso

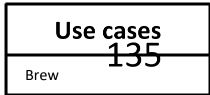
继承


Coffee
+ float cost()
CoffeePack
CoffeeMaker
+ Expresso brewCoffee(CoffeePack pack)

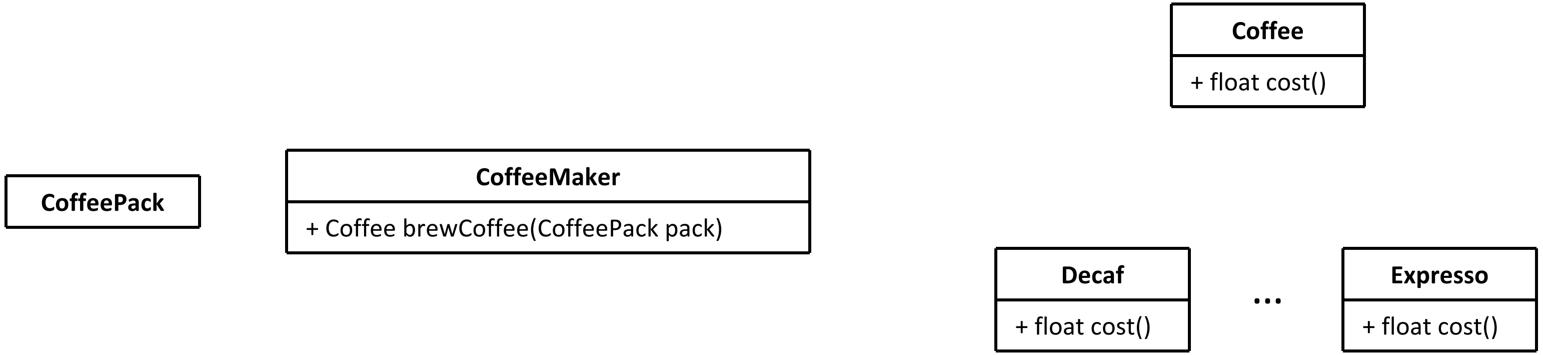

Use cases
Brew

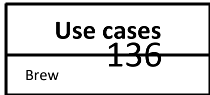
继承


Coffee
+ float cost()
CoffeePack
CoffeeMaker
+ Expresso brewCoffee(CoffeePack pack)
Decaf
+ float cost()
…
Expresso
+ float cost()

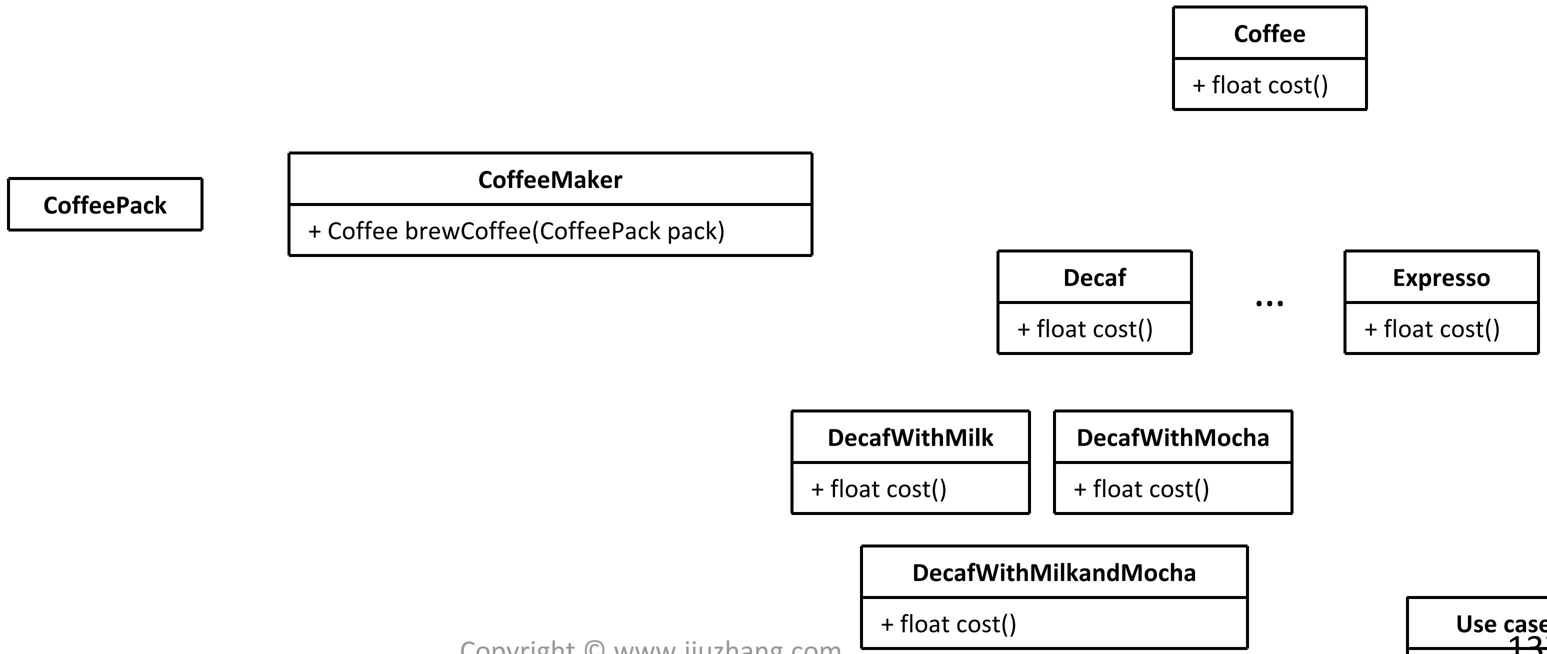

Use cases
Brew

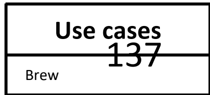
继承


Coffee
+ float cost()
CoffeePack
CoffeeMaker
+ Coffee brewCoffee(CoffeePack pack)
Decaf
+ float cost()
...
Expresso
+ float cost()

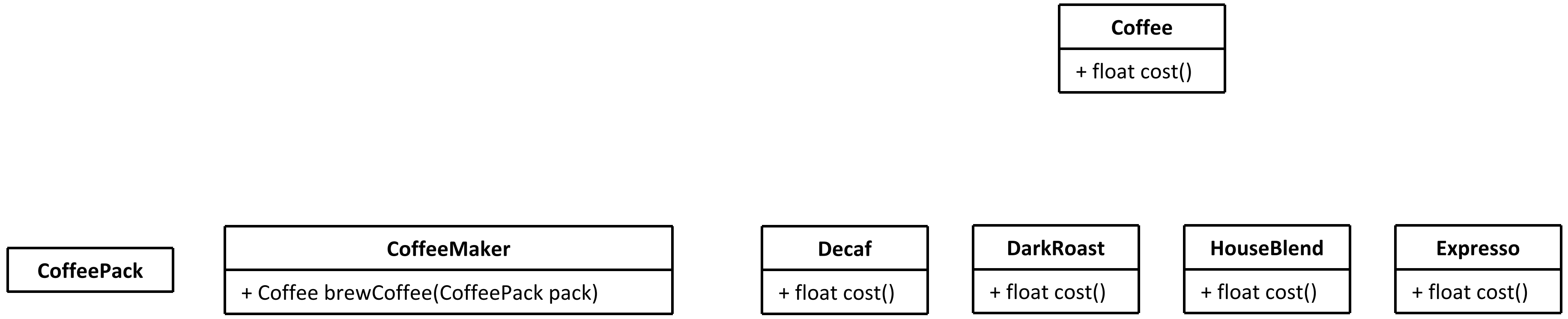

Use cases
Brew

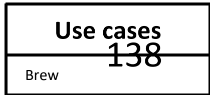
继承


CoffeePack
CoffeeMaker
+ Coffee brewCoffee(CoffeePack pack)
Coffee
+ float cost()
Decaf
+ float cost()
…
Expresso
+ float cost()
DecafWithMilk
+ float cost()
DecafWithMocha
+ float cost()
DecafWithMilkandMocha
+ float cost()

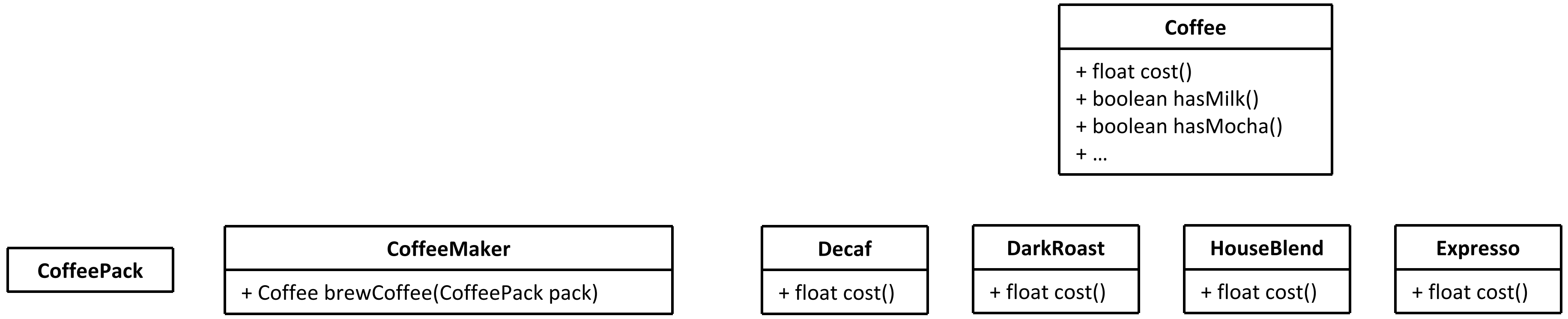

Use cases
Brew

另一种继承

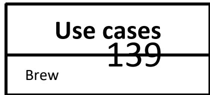

Coffee
+ float cost()
CoffeePack
CoffeeMaker
+ Coffee brewCoffee(CoffeePack pack)
Decaf
+ float cost()
DarkRoast
+ float cost()
HouseBlend
+ float cost()
Expresso
+ float cost()


Use cases
Brew

另一种继承

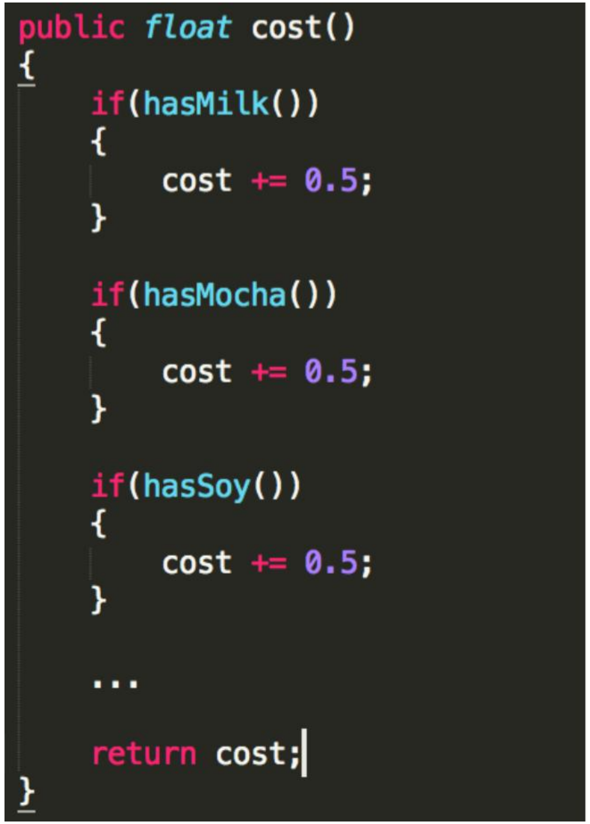


Coffee
+ float cost()
+ boolean hasMilk()
+ boolean hasMocha()
+ …
CoffeePack
CoffeeMaker
+ Coffee brewCoffee(CoffeePack pack)
Decaf
+ float cost()
DarkRoast
+ float cost()
HouseBlend
+ float cost()
Expresso
+ float cost()



Use cases
Brew

另一种继承

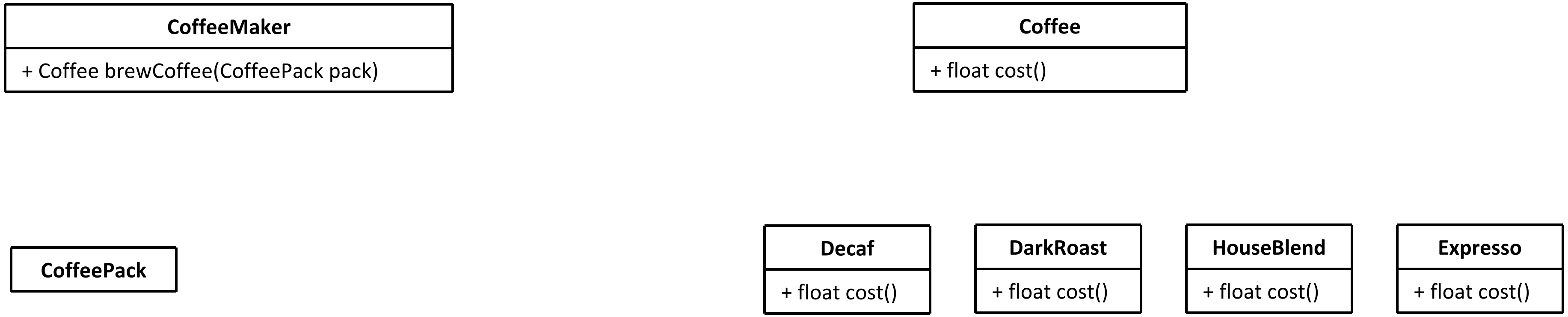


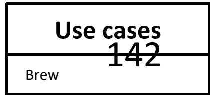
```java
public float cost()
{
    if(hasMilk())
    {
        cost += 0.5;
    }

    if(hasMocha())
    {
        cost += 0.5;
    }

    if(hasSoy())
    {
        cost += 0.5;
    }

    ...

    return cost;
}
```

Decorator Design Pattern

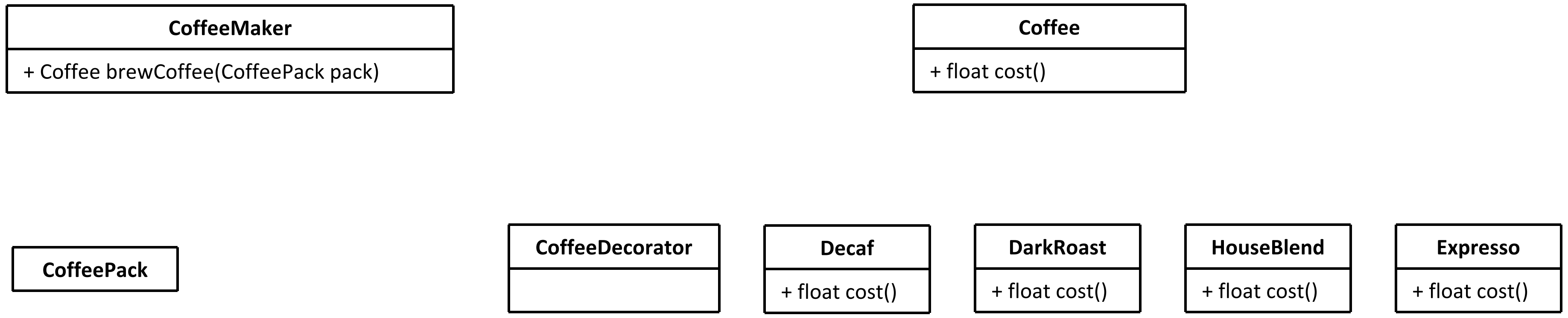
- Decorator pattern allows a user to add new functionality to an existing object without altering its structure. This type of design pattern comes under structural pattern as this pattern acts as a wrapper to existing class.

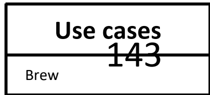
Decorator


CoffeeMaker
+ Coffee brewCoffee(CoffeePack pack)
Coffee
+ float cost()
CoffeePack
Decaf
+ float cost()
DarkRoast
+ float cost()
HouseBlend
+ float cost()
Expresso
+ float cost()

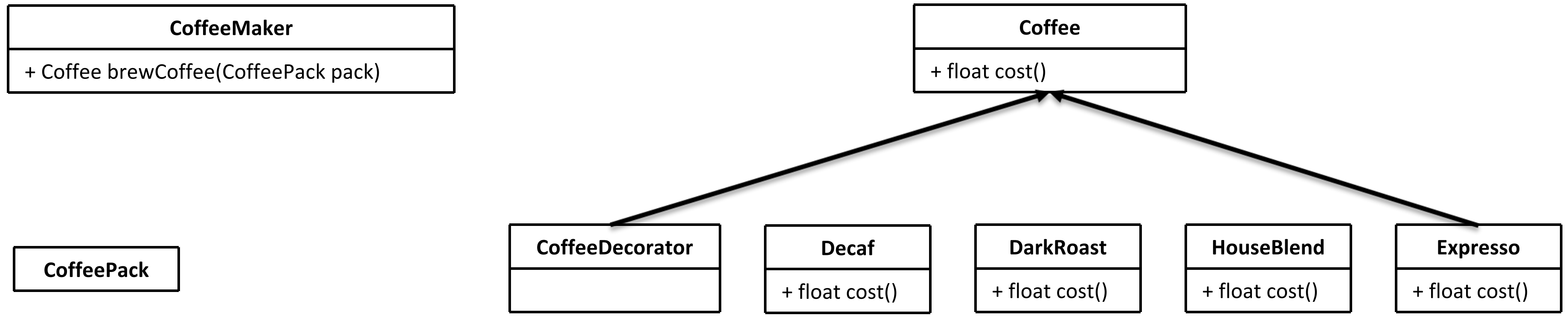

Use cases
Brew

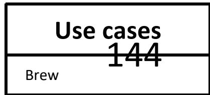
Decorator


CoffeeMaker
+ Coffee brewCoffee(CoffeePack pack)
Coffee
+ float cost()
CoffeePack
CoffeeDecorator
Decaf
+ float cost()
DarkRoast
+ float cost()
HouseBlend
+ float cost()
Expresso
+ float cost()

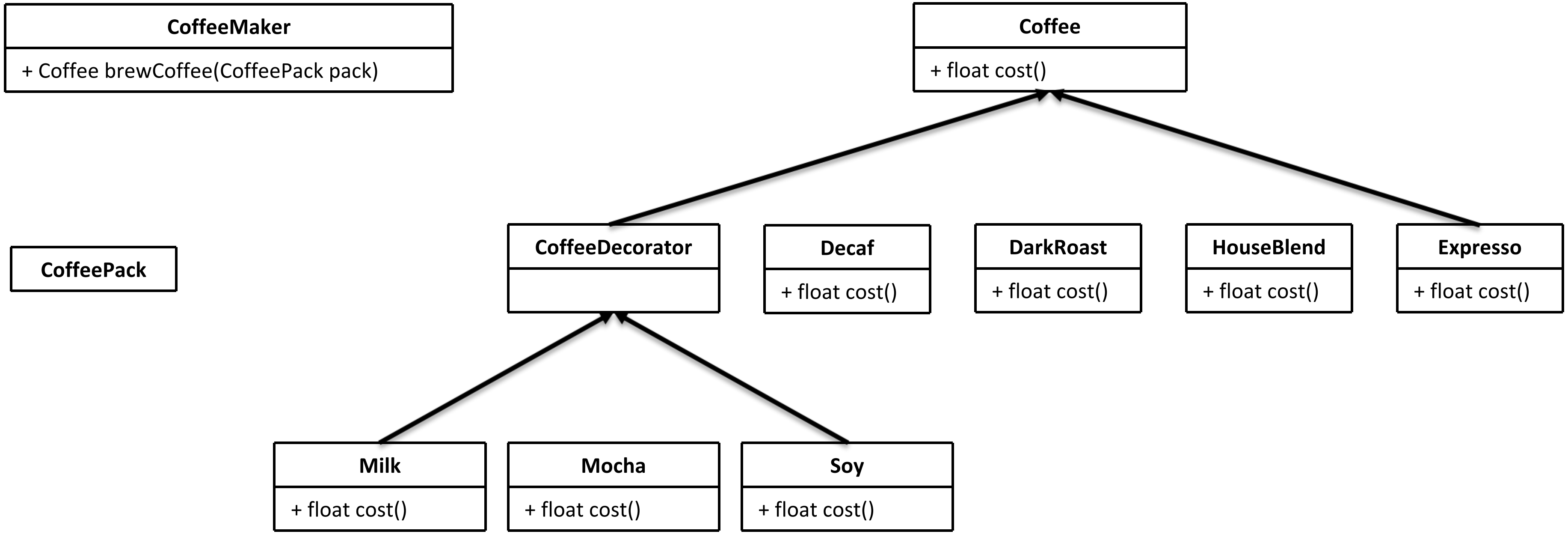

Use cases
Brew

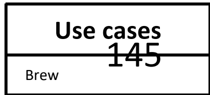
Decorator


CoffeeMaker
+ Coffee brewCoffee(CoffeePack pack)
CoffeePack
Coffee
+ float cost()
CoffeeDecorator
Decaf
+ float cost()
DarkRoast
+ float cost()
HouseBlend
+ float cost()
Expresso
+ float cost()

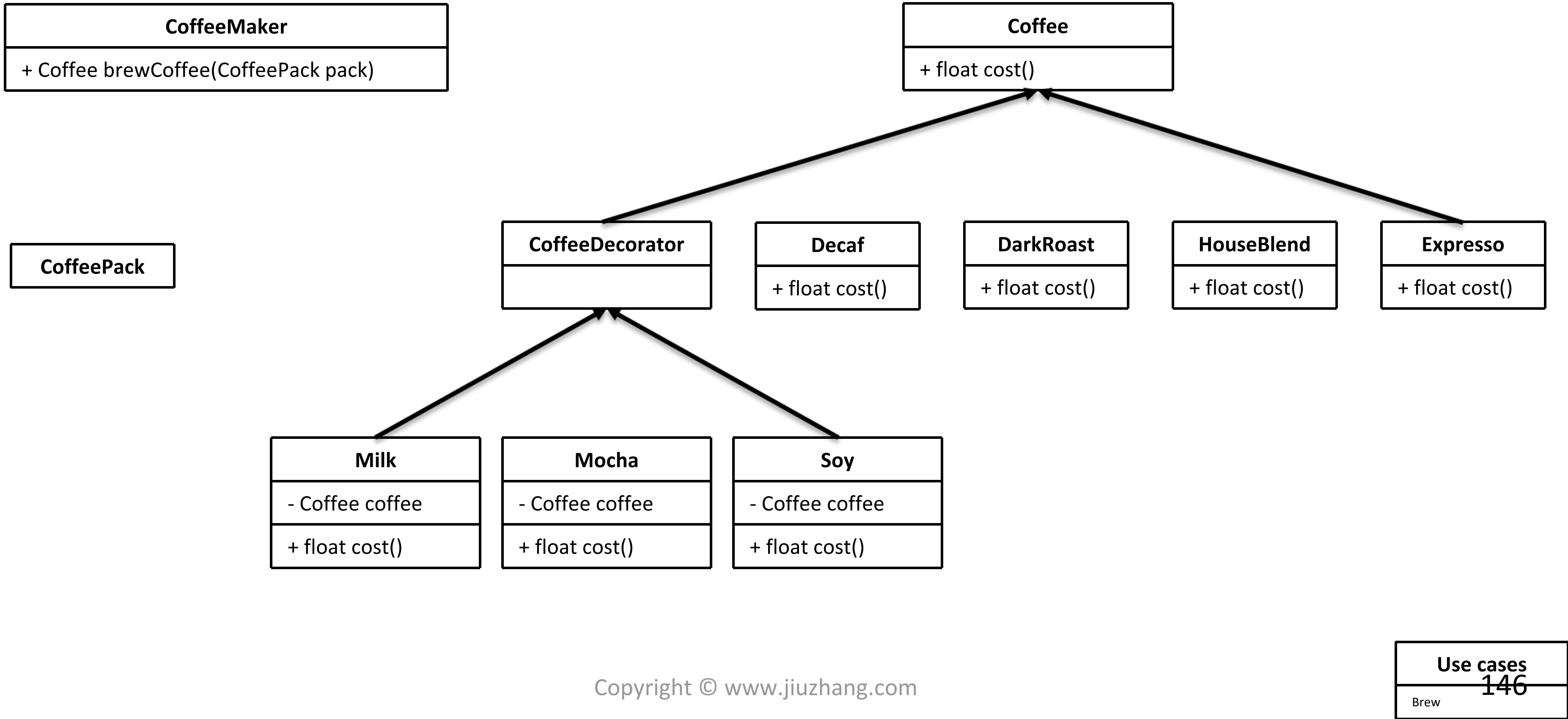

Use cases
Brew

Decorator

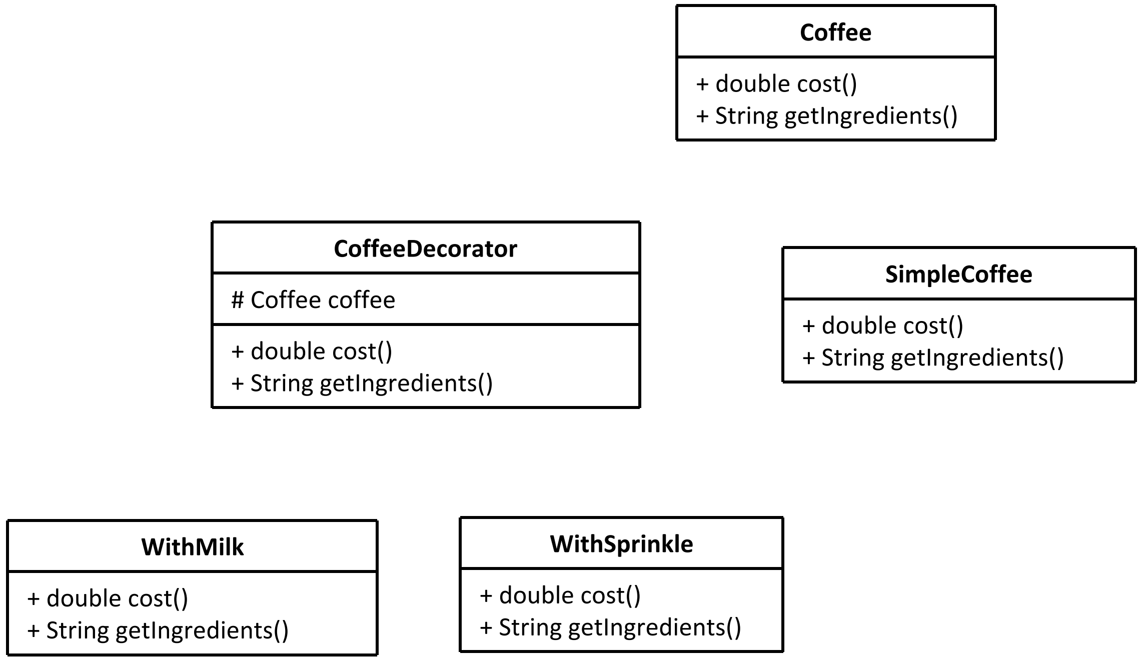

CoffeeMaker
+ Coffee brewCoffee(CoffeePack pack)
CoffeePack
Coffee
+ float cost()
CoffeeDecorator
Decaf
+ float cost()
DarkRoast
+ float cost()
HouseBlend
+ float cost()
Expresso
+ float cost()
Milk
+ float cost()
Mocha
+ float cost()
Soy
+ float cost()


Use cases
Brew

Decorator


CoffeeMaker
+ Coffee brewCoffee(CoffeePack pack)
CoffeePack
Coffee
+ float cost()
CoffeeDecorator
Decaf
+ float cost()
DarkRoast
+ float cost()
HouseBlend
+ float cost()
Expresso
+ float cost()
Milk
- Coffee coffee
+ float cost()
Mocha
- Coffee coffee
+ float cost()
Soy
- Coffee coffee
+ float cost()
Use cases
Brew

Decorator


Coffee
+ double cost()
+ String getIngredients()
CoffeeDecorator
# Coffee coffee
+ double cost()
+ String getIngredients()
SimpleCoffee
+ double cost()
+ String getIngredients()
WithMilk
+ double cost()
+ String getIngredients()
WithSprinkle
+ double cost()
+ String getIngredients()

Decorator

```java
// The interface Coffee defines the functionality of Coffee implemented by decorator
public interface Coffee {
    public double getCost(); // Returns the cost of the coffee
    public String getIngredients(); // Returns the ingredients of the coffee
}

// Extension of a simple coffee without any extra ingredients
public class SimpleCoffee implements Coffee {
    @Override
    public double getCost() {
        return 1;
    }

    @Override
    public String getIngredients() {
        return "Coffee";
    }
}
```

Decorator

```
// Abstract decorator class - note that it implements Coffee interface
public abstract class CoffeeDecorator implements Coffee {
    protected final Coffee decoratedCoffee;

    public CoffeeDecorator(Coffee c) {
        this.decoratedCoffee = c;
    }

    public double getCost() { // Implementing methods of the interface
        return decoratedCoffee.getCost();
    }

    public String getIngredients() {
        return decoratedCoffee.getIngredients();
    }
}
```

Decorator

```java
// Decorator WithMilk mixes milk into coffee.
// Note it extends CoffeeDecorator.
class WithMilk extends CoffeeDecorator {
    public WithMilk(Coffee c) {
        super(c);
    }

    public double getCost() { // Overriding methods defined in the abstract superclass
        return super.getCost() + 0.5;
    }

    public String getIngredients() {
        return super.getIngredients() + ", Milk";
    }
}

// Decorator WithSprinkles mixes sprinkles onto coffee.
// Note it extends CoffeeDecorator.
class WithSprinkles extends CoffeeDecorator {
    public WithSprinkles(Coffee c) {
        super(c);
    }

    public double getCost() {
        return super.getCost() + 0.2;
    }

    public String getIngredients() {
        return super.getIngredients() + ", Sprinkles";
    }
}
```

Decorator

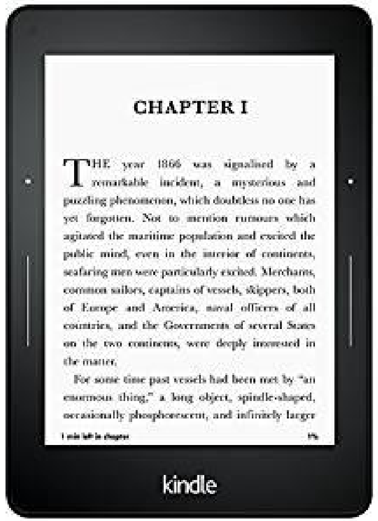
```java
public class Main {
    public static void printInfo(Coffee c) {
        System.out.println("Cost: " + c.getCost() + "; Ingredients: " + c.getIngredients());
    }

    public static void main(String[] args) {
        Coffee c = new SimpleCoffee();
        printInfo(c);

        c = new WithMilk(c);
        printInfo(c);

        c = new WithSprinkles(c);
        printInfo(c);
    }
}
```

The output of this program is given below:

```
Cost: 1.0; Ingredients: Coffee
Cost: 1.5; Ingredients: Coffee, Milk
Cost: 1.7; Ingredients: Coffee, Milk, Sprinkles
```

Kindle

Can you design Kindle?



CHAPTER I

THE year 1866 was signalized by a remarkable incident, a mysterious and puzzling phenomenon, which doubtless no one has yet forgotten. Not to mention rumours which agitated the maritime population and excited the public mind, even in the interior of continents, seafaring men were particularly excited. Merchants, common sailors, captains of vessels, skippers, both of Europe and America, naval officers of all countries, and the Governments of several States on the two continents, were deeply interested in the matter.

For some time past vessels had been met by "an enormous thing," a long object, spindle-shaped, occasionally phosphorescent, and infinitely larger

1 min left in chapter 1%

Clarify

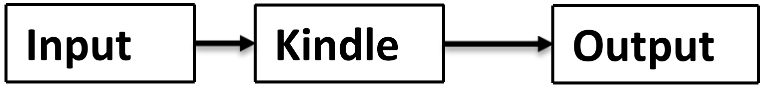
- What

关键字： Kindle

Clarify

- What

关键字： Kindle


$$\begin{tikzcd}\boxed{\textbf{Input}} \arrow[r] & \boxed{\textbf{Kindle}} \arrow[r] & \boxed{\textbf{Output}}\end{tikzcd}$$

Clarify

- What

关键字： Kindle

$$\begin{tikzcd}\boxed{\text{N/A}} \arrow[r] & \boxed{\text{Kindle}} \arrow[r] & \boxed{\text{Book}}\end{tikzcd}$$

Clarify

- What

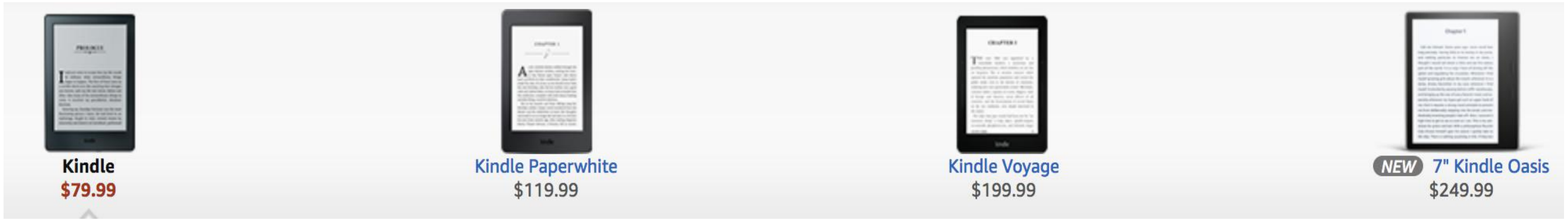
关键字： Kindle, Book

Clarify

关键字： Kindle

Clarify

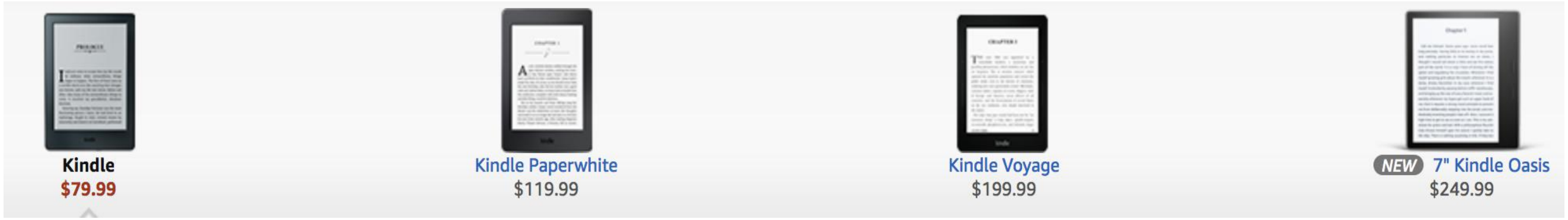
关键字： Kindle



需不需要设计不同版本？

Clarify

关键字： Kindle

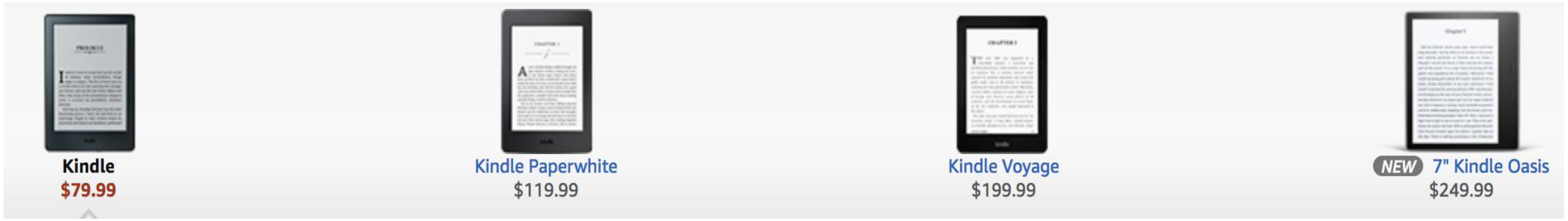


需不需要设计不同版本？

- Design: get price

Clarify

关键字： Kindle


Kindle
$79.99
Kindle Paperwhite
$119.99
Kindle Voyage
$199.99
NEW 7" Kindle Oasis
$249.99

需不需要设计不同版本？

- Design: get price
- Design: Memory difference

Clarify

关键字： Book

Clarify

关键字：　Book

- 支持哪些格式的电子书？

Clarify

- 对于本题：

- 不需要考虑不同的版本
- 不需要考虑内存和书的大小
- 支持pdf, epub 和 mobi三种格式

Clarify

How

Clarify

如何获取电子书？

Clarify

如何获取电子书？

- 是否支持Upload
- 是否支持Download

Clarify

如何获取电子书？

- Upload
- Download

对于付费的电子书，提供哪些支付功能？

Clarify

如何获取电子书？

- Upload
- Download

对于付费的电子书，提供哪些支付功能？

Payment -> Strategy design pattern

Clarify

- 对于本题：支持上传，下载
- 对于本题：不需要考虑付费

Clarify

- Who

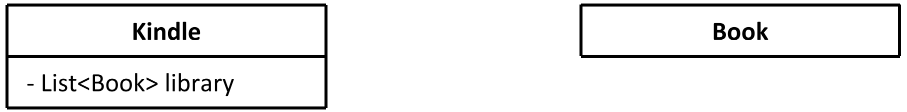
- N/A

Core Object

Core Object

Kindle
Book

Core Object



Use cases

- Kindle

Use cases

- Kindle

- Upload book

Use cases

- Kindle

- Upload book
- Download book

Use cases

- Kindle

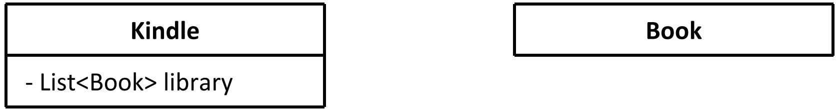
- Upload book
- Download book
- Read book

Use cases

- Kindle

- Upload book
- Download book
- Read book
- Remove book

Classes

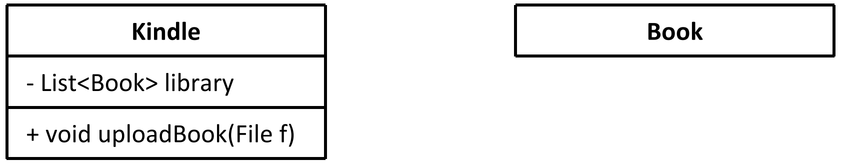


Use cases
Upload book
Download book
Read book
Remove book

Upload book

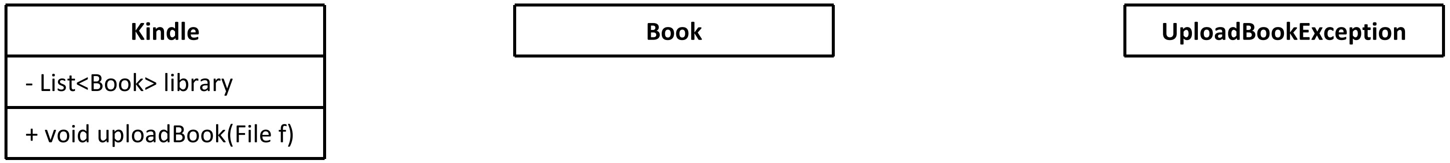
- Upload a file to kindle and store as a book

Classes



Use cases
Upload book
Download book
Read book
Remove book

Classes

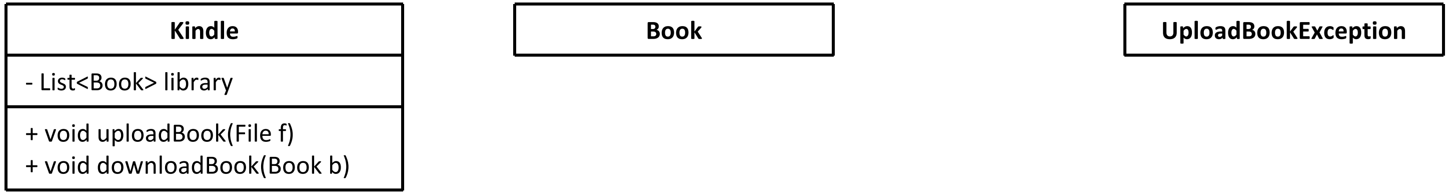


Use cases
Upload book
Download book
Read book
Remove book

Download book

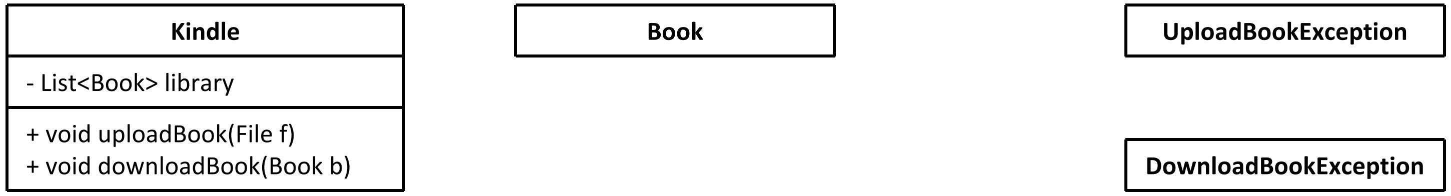
- Download a book and put in library

Classes



Use cases
Upload book
Download book
Read book
Remove book

Classes



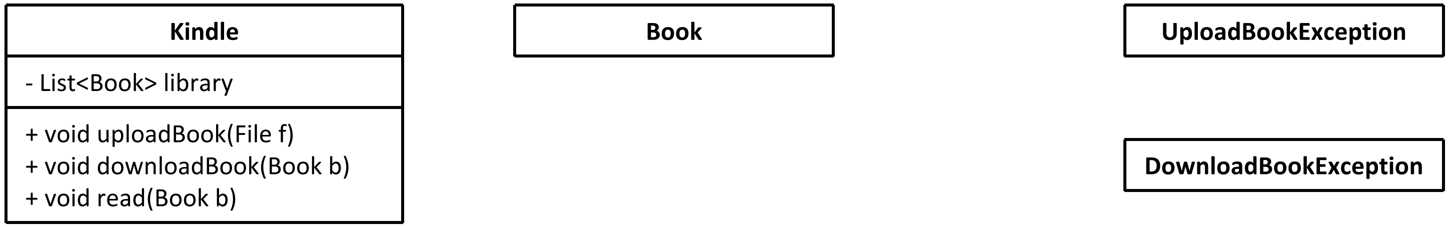
```
Kindle
- List<Book> library
+ void uploadBook(File f)
+ void downloadBook(Book b)
```

Use cases
Upload book
Download book
Read book
Remove book

Read book

- Select a book and display it

Classes

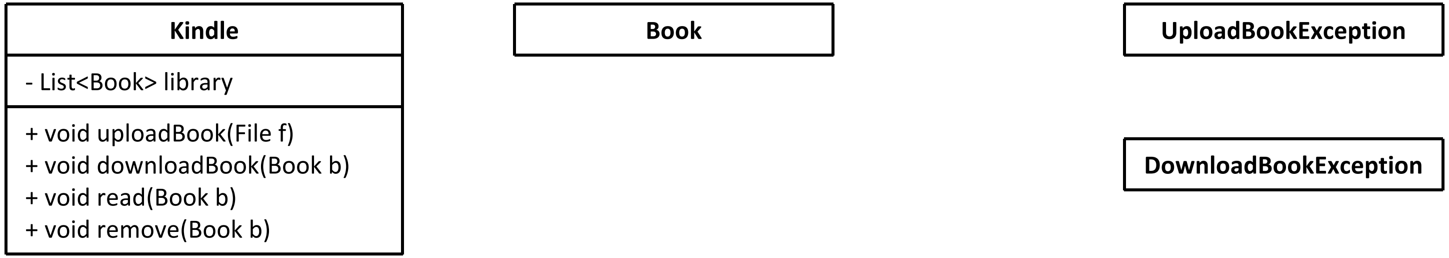


Use cases
Upload book
Download book
Read book
Remove book

Remove book

- Remove a book from library

Classes

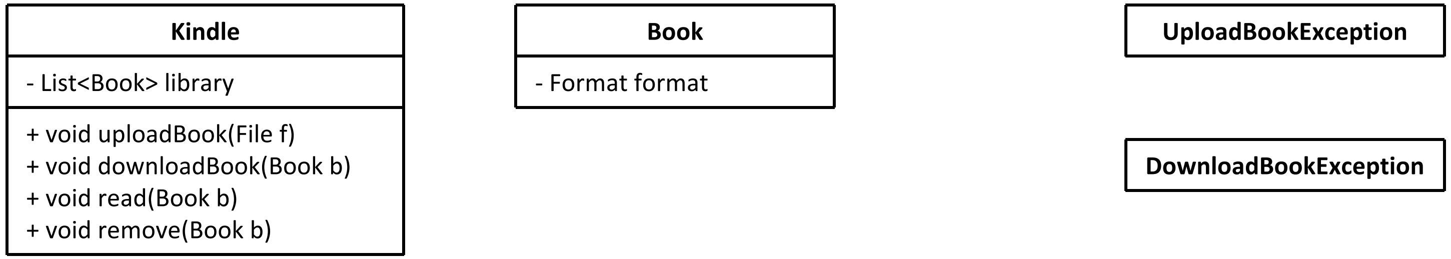


Use cases
Upload book
Download book
Read book
Remove book

Challenge

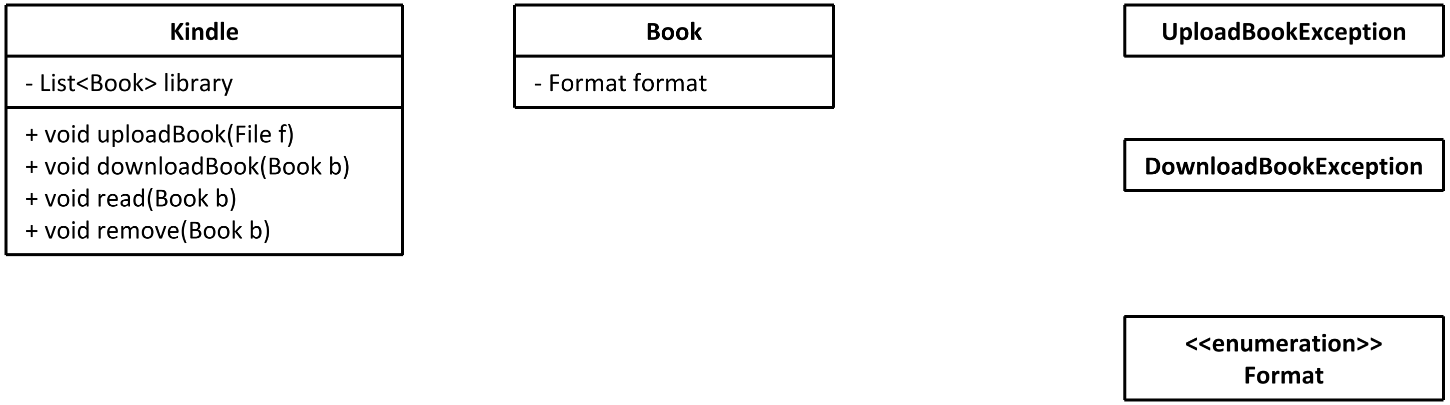
- What about different book format?

Classes



Use cases
Upload book
Download book
Read book
Remove book

Classes



Use cases
Upload book
Download book
Read book
Remove book

Classes

Kindle
- List<Book> library
+ void uploadBook(File f) + void downloadBook(Book b) + void read(Book b) + void remove(Book b)

Book
- Format format

UploadBookException

DownloadBookException

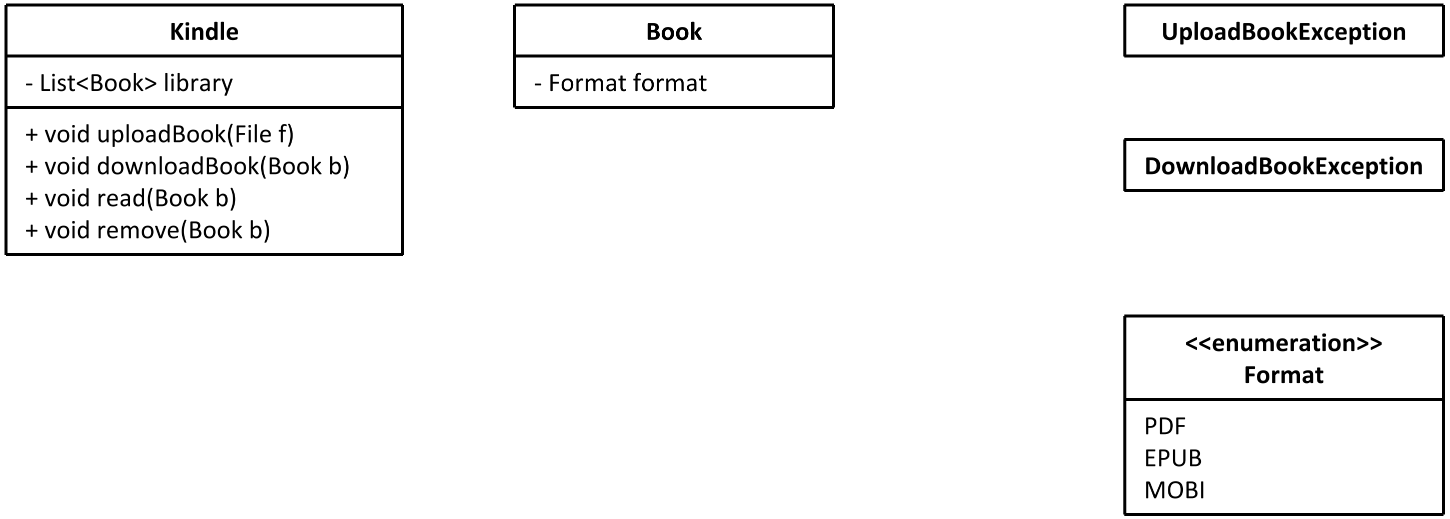
<<enumeration>> Format
PDF EPUB MOBI

Use cases
Upload book
Download book
Read book
Remove book

Challenge

- How would read book work?

Classes



```
Kindle
- List<Book> library
+ void uploadBook(File f)
+ void downloadBook(Book b)
+ void read(Book b)
+ void remove(Book b)
```

```
Book
- Format format
```

```
UploadBookException
```

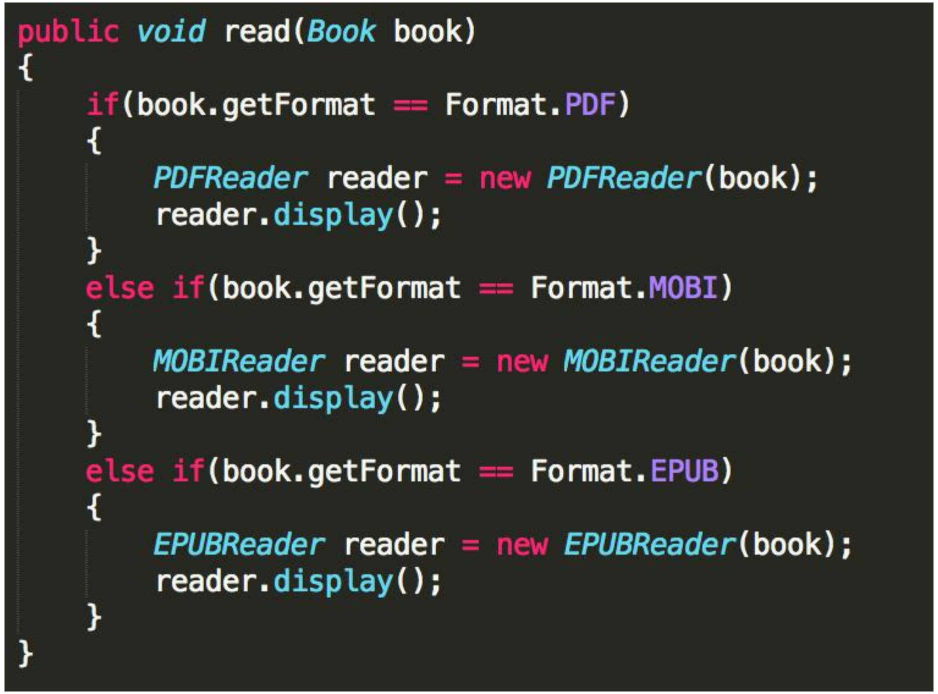
```
DownloadBookException
```

```
<<enumeration>>
Format
PDF
EPUB
MOBI
```

Use cases
Upload book
Download book
Read book
Remove book

Challenge

- How would read book work?

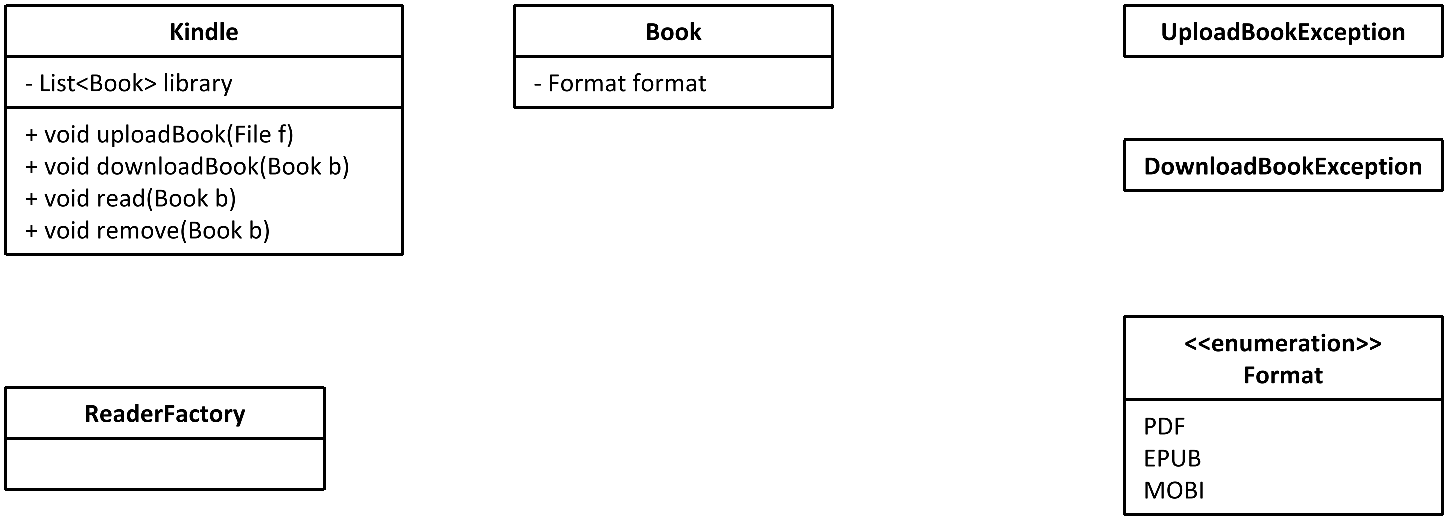


```
public void read(Book book)
{
    if(book.getFormat == Format.PDF)
    {
        PDFReader reader = new PDFReader(book);
        reader.display();
    }
    else if(book.getFormat == Format.MOBI)
    {
        MOBIReader reader = new MOBIReader(book);
        reader.display();
    }
    else if(book.getFormat == Format.EPUB)
    {
        EPUBReader reader = new EPUBReader(book);
        reader.display();
    }
}
```

Challenge

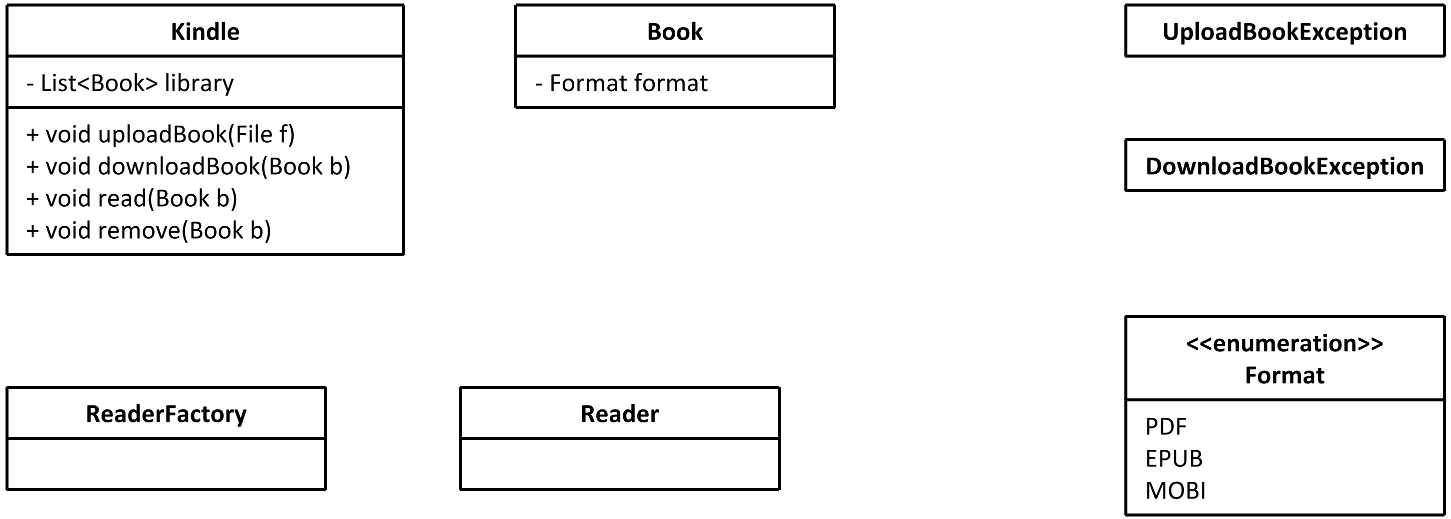
- Solution: Factory design pattern

Factory design pattern



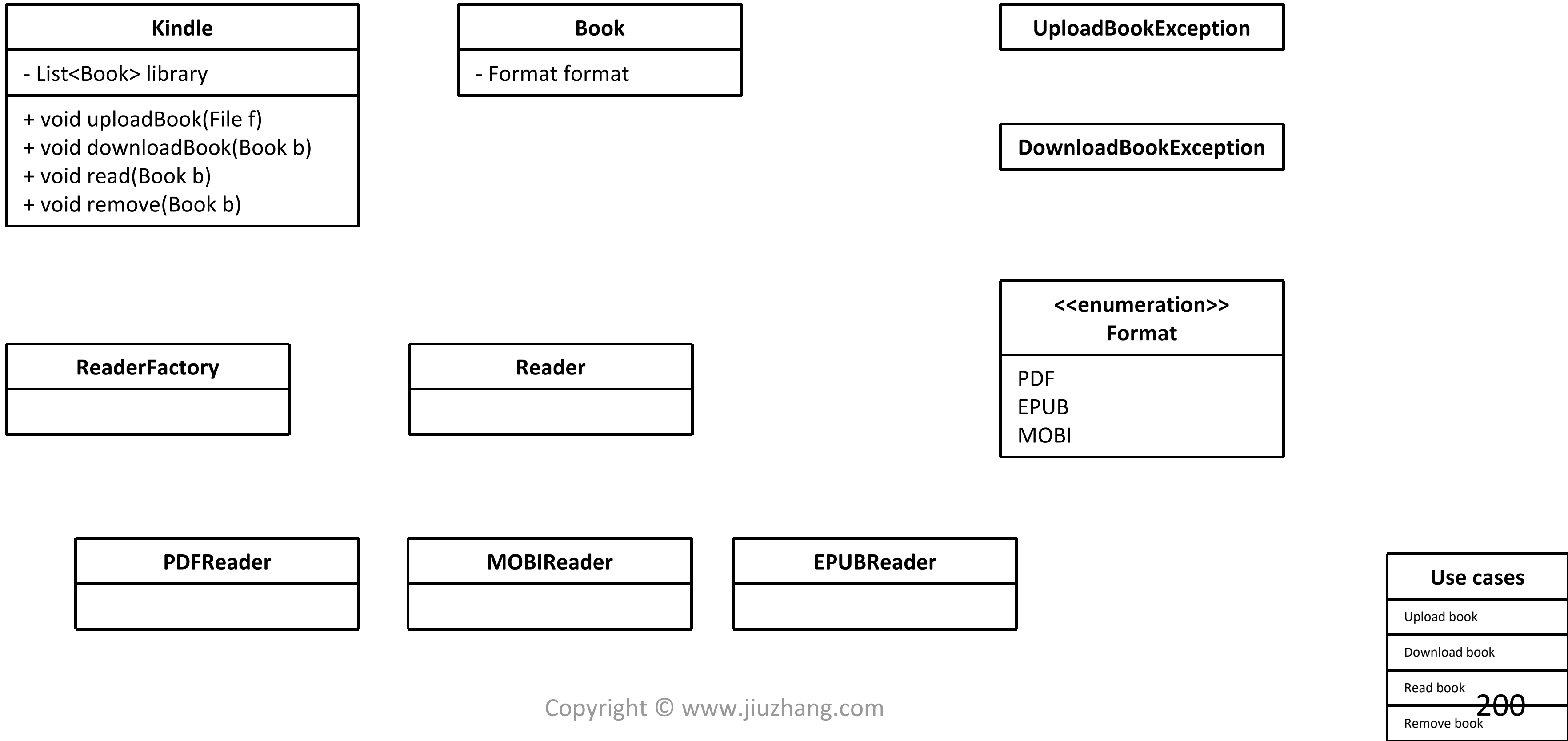
Use cases
Upload book
Download book
Read book
Remove book

Factory design pattern

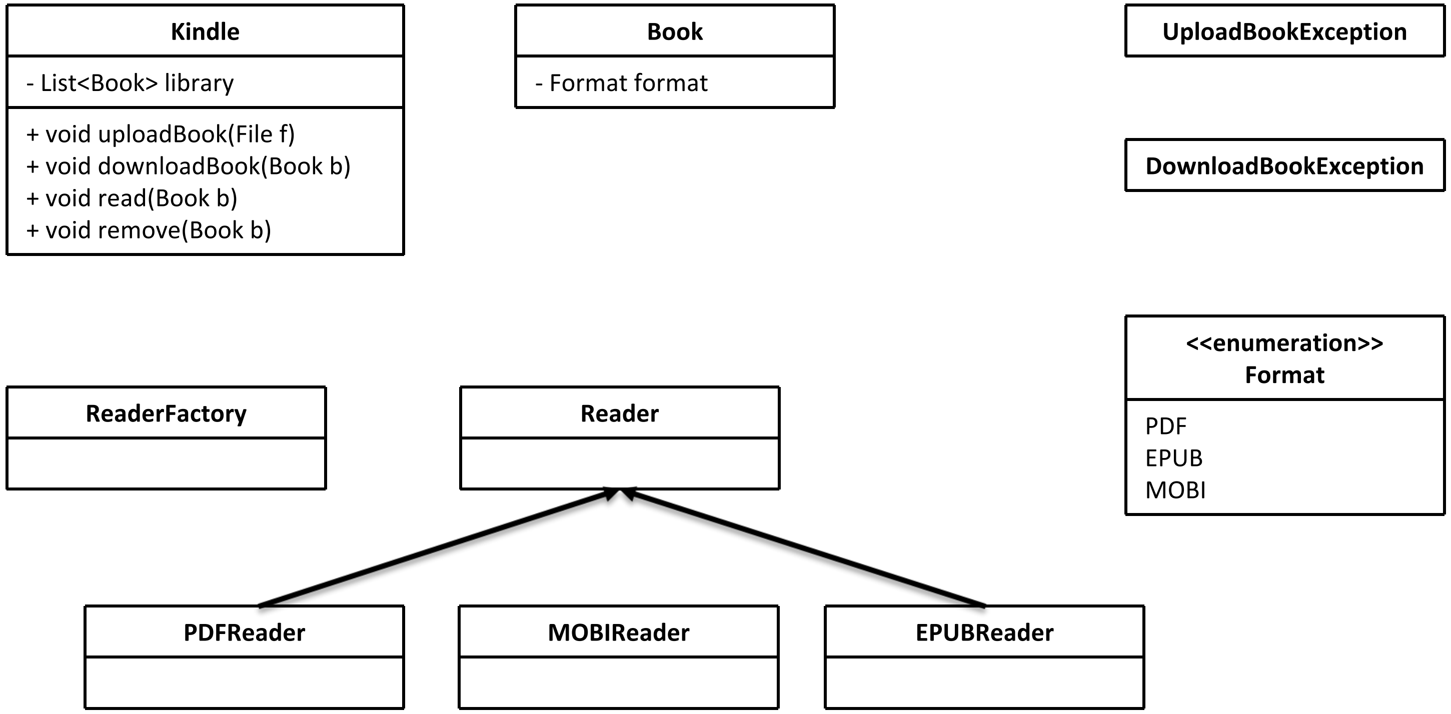


Use cases
Upload book
Download book
Read book
Remove book

Factory design pattern

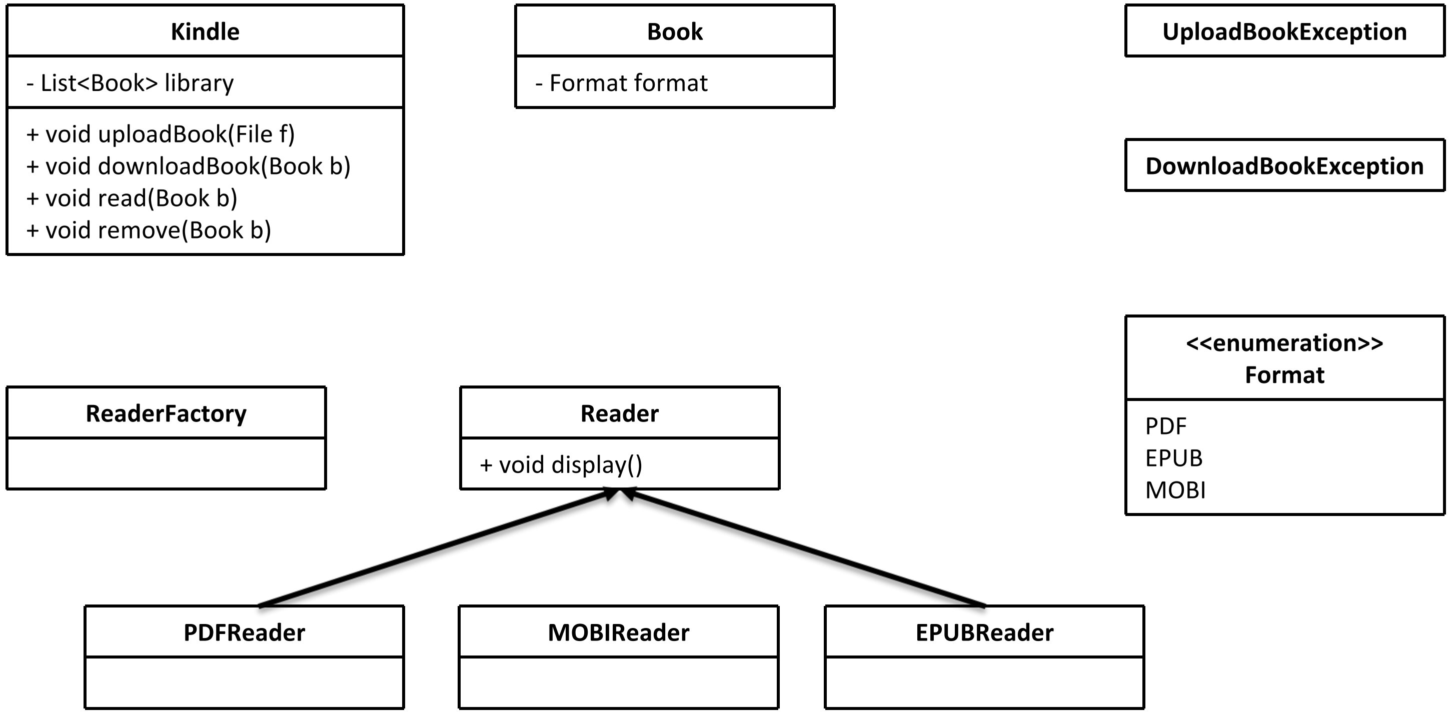


Factory design pattern



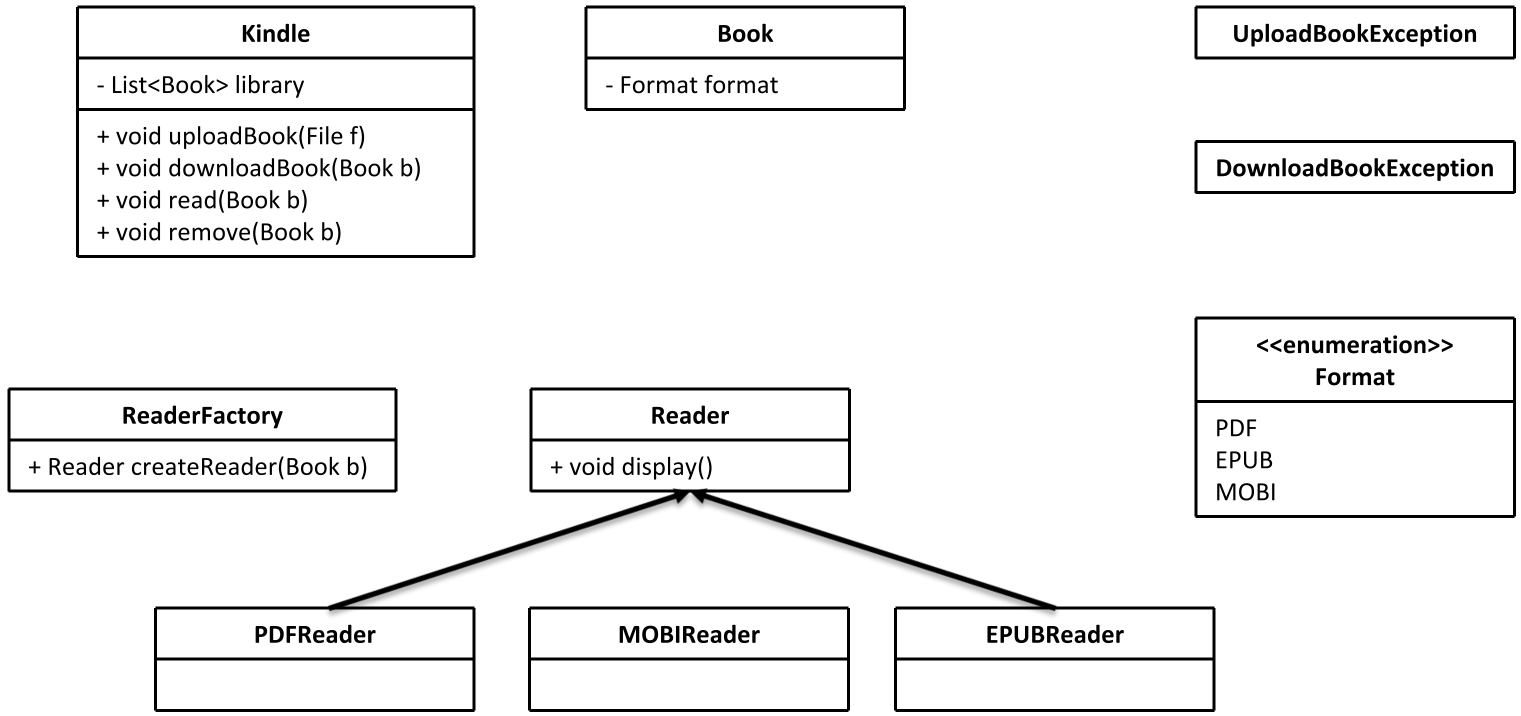
Use cases
Upload book
Download book
Read book
Remove book

Factory design pattern



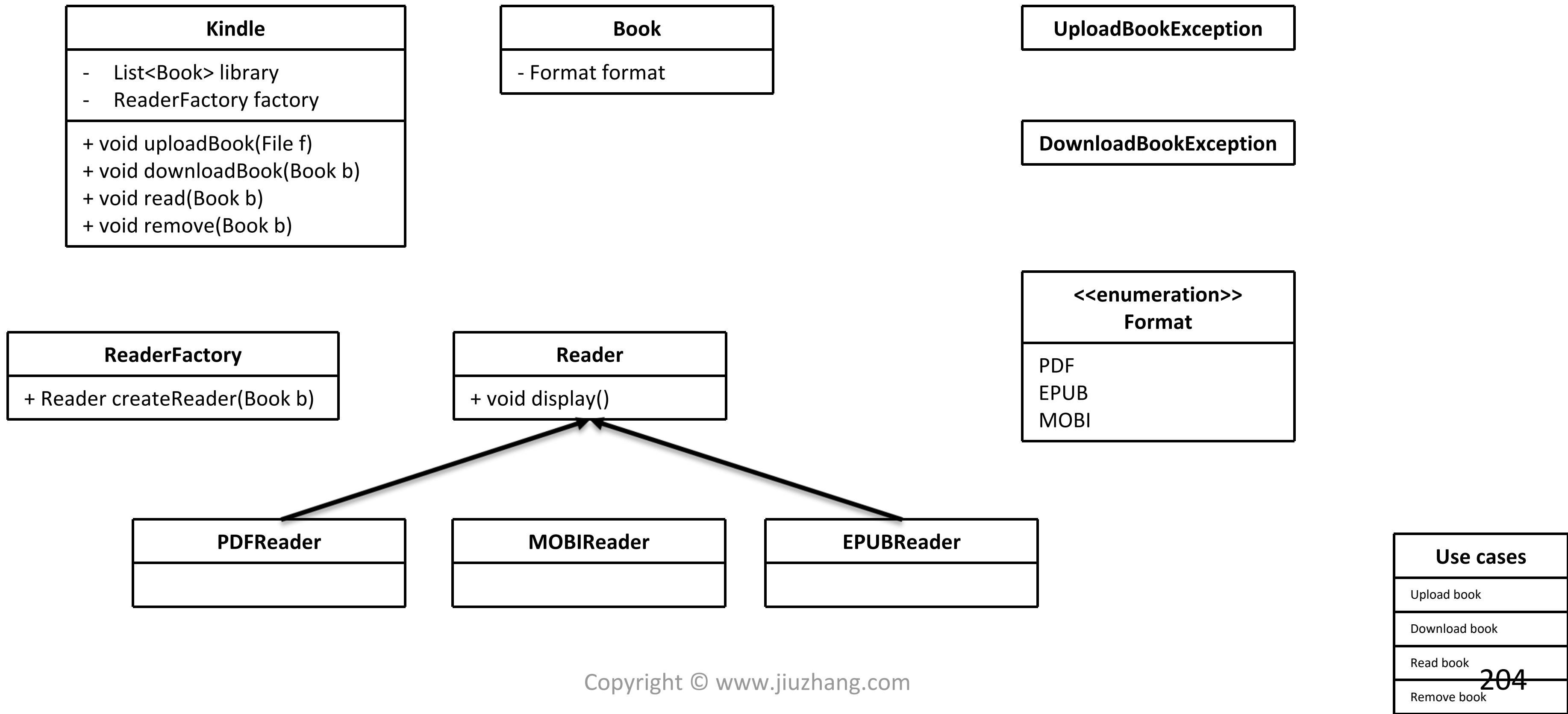
Use cases
Upload book
Download book
Read book
Remove book

Factory design pattern

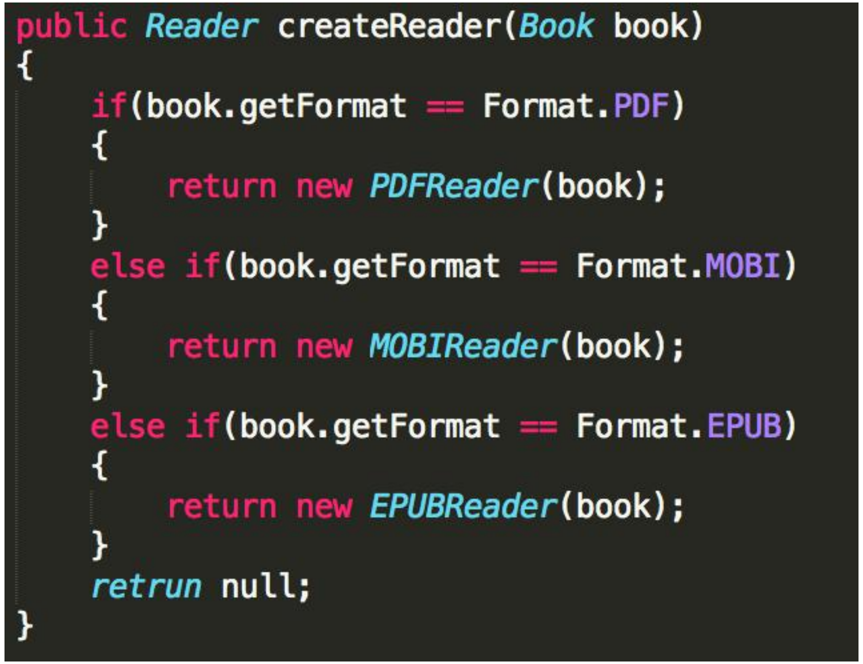


Use cases
Upload book
Download book
Read book
Remove book

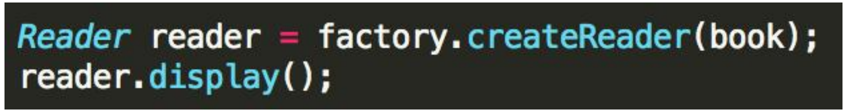
Factory design pattern



Simple factory



```
public Reader createReader(Book book)
{
    if(book.getFormat == Format.PDF)
    {
        return new PDFReader(book);
    }
    else if(book.getFormat == Format.MOBI)
    {
        return new MOBIReader(book);
    }
    else if(book.getFormat == Format.EPUB)
    {
        return new EPUBReader(book);
    }
    retrun null;
}
```

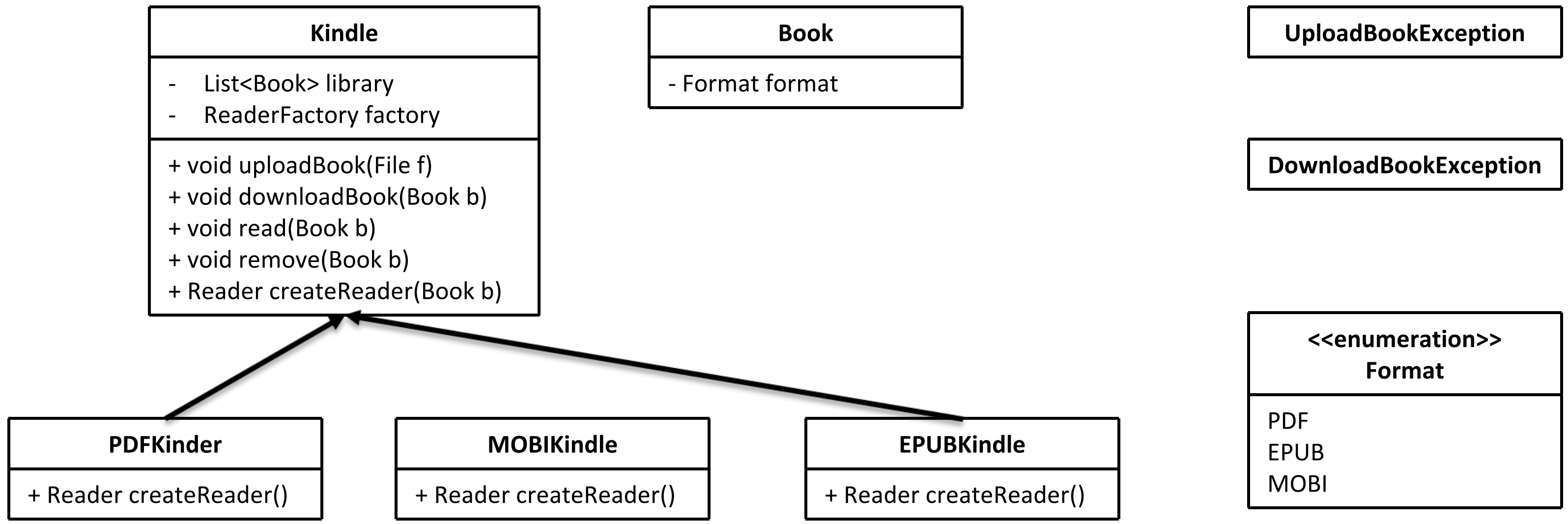


```
Reader reader = factory.createReader(book);
reader.display();
```

Factory design pattern

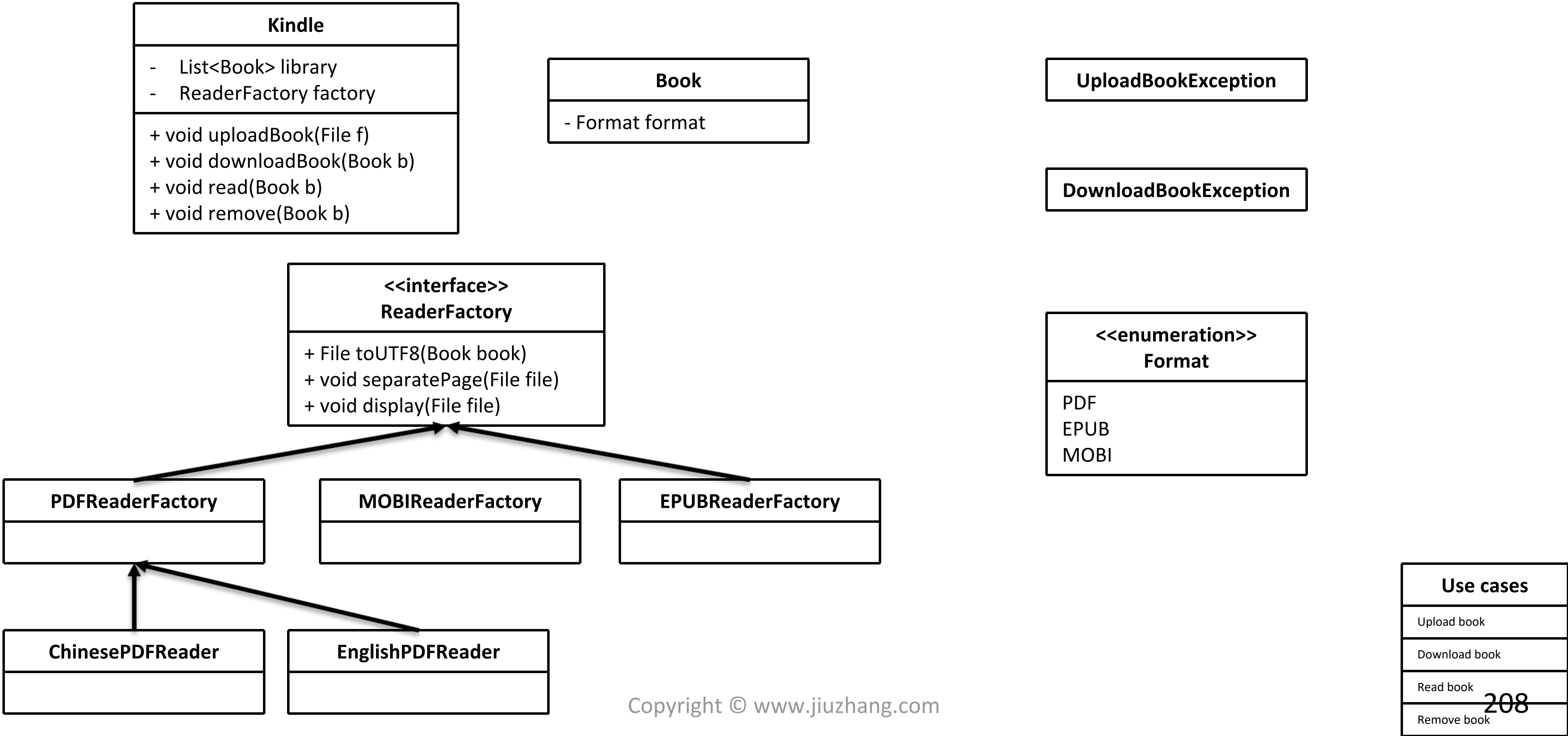
- Factory method
- Abstract factory

Factory method



Use cases
Upload book
Download book
Read book
Remove book

Abstract factory



Recap

- 常见的实物类面向对象设计

Recap

- 常见的实物类面向对象设计
- Input -> 题目主体 -> Output
- State design pattern
- Decorate design pattern
- Factory design pattern

Q & A

扫描二维码关注微信/微博
获取最新面试题及权威解答

微信: ninechapter
知乎专栏：http://zhuanlan.zhihu.com/jiuzhang
微博: http://www.weibo.com/ninechapter
官网: www.jiuzhang.com