

Lecture 4 - 实物类: Vending Machine & Juke Box - Factory & Adaptor Pattern

- 实物类 OOD 题型
 - Vending machine
 - Jukebox
 - CD Player
 - Coffee maker
 - ATM
- 实物类
 - 考虑对于实物的输入输出
- 考虑对于实物的输入输出

例子: Coffee maker



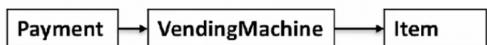
- 技巧
 - State pattern
 - Decorate pattern
 - Factory pattern

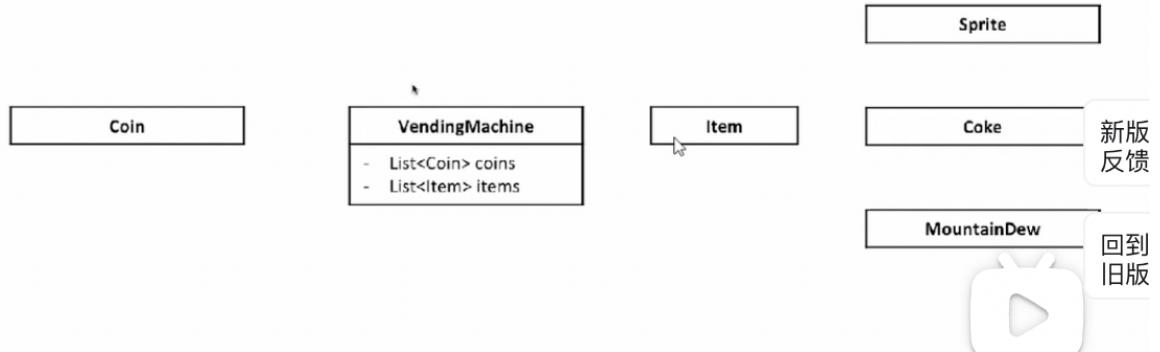
Vending Machine & ATM Machine

Clarify

- What

关键字: Vending machine





Use cases

Vending machine:

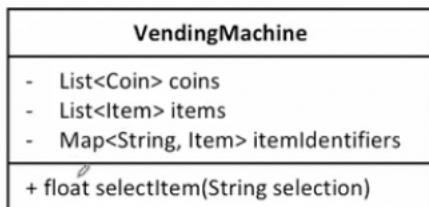
- Select item
- Insert coin
- Execute transaction
- Cancel transaction

Classes

VendingMachine

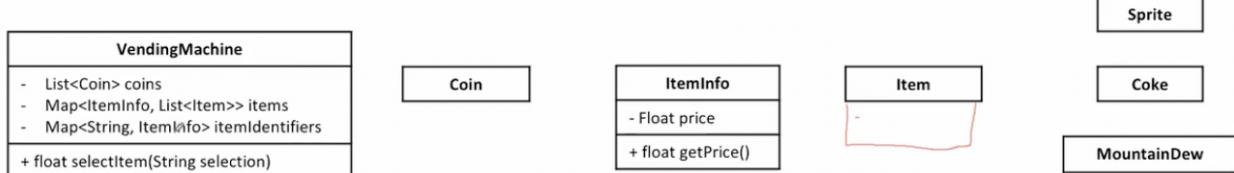
Use case: Select item

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



如果用上面的方式定义 vendingmachine 存在的问题是 map 里是用静态的数据“比如 A1, B4”来对应一个动态的 Item, 那如果当客户拿到了 Item 的信息就可以对它进行修改。解决方法是可以加一个静态的 class ItemInfo

Classes



上图漏了一个点， Item class 下面应该包括 itemInfo 信息

Classes



Use case: Insert coin

- Insert a list of coins into vending machine

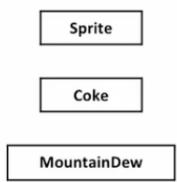
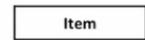
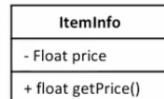
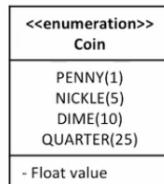
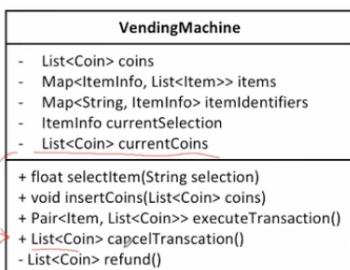
Classes



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



NotEnoughMoneyException 新版反馈



pair 就相当于是可以实现几返回 item 也返回需要的找零

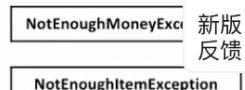
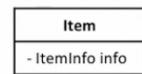
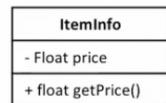
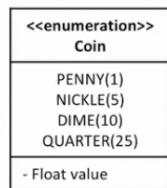
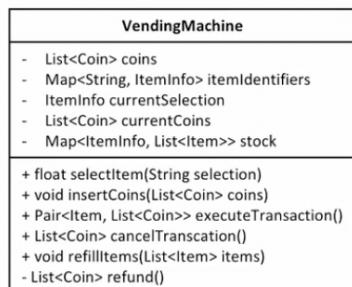
Classes



Use case: Refill items

- Refill items on top of current stock

Classes - Final view



新版
反馈

回到
旧版



Visit www.cs3k.com to get the latest videos.

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

CS3K

```

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();

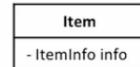
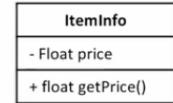
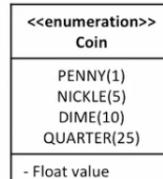
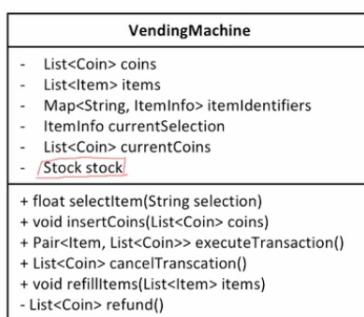
    public void refillItem(List<Item> items)
    {
        for(Item item : items)
        {
            stock.add(item);
        }
    }
}

```

直接把 add method 放到 stock 这个 class 里，nested class 把常用的借口封装起来

CS3K.com

Good practice



Sprite

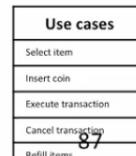
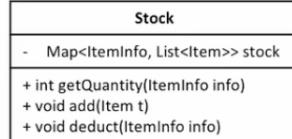
Coke

MountainDew

NotEnoughMoneyException 新版
反馈

NotEnoughItemException |

回到
旧版



Visit www.cs3k.com to get the latest videos.

小错误：item 中的 iteminfo info 应该是 protected，这样的话 spirit, coke 继承的时候可以自己贴商标

State Design Pattern

- States
 - HAS_SELECTION - NO_SELECTION

- COINS_INSERTED - NO_COIN
- SOLD
- SOLD_OUT
- State related actions:
 - select item
 - insert coin
 - execute transaction
 - cancel transaction
- what will happen if some item has been selected?

OOD 面向对象设计专题班
Challenge

CS3K.com

- For these use cases:
- Select item : throws a selection 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

CS3K.com

- 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?



- 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
- 那么对于:
 - COINS_INSERTED
 - NO_COIN
 - SOLD
 - SOLD_OUT

应该怎么办?



```
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(SOLD)
    {
        throw new Exception("Be patient, item is coming out, dont need to pay once more");
    }
    else if(SOLD_OUT)
    {
        throw new Exception("Item has been sold out, please dont pay for nothing");
    }
    ...
}
```

Challenge

- State Design Pattern

States:

- HAS_SELECTION
- NO_SELECTION
- COINS_INSERTED
- NO_COIN
- SOLD
- SOLD_OUT

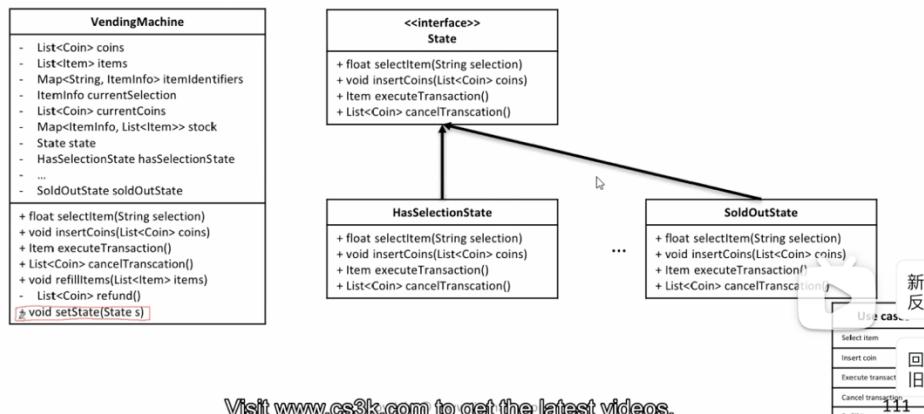
- State Design Pattern

State related actions:

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

Classes

UML Class Diagram



Visit www.cs8k.com to see the latest videos..

Vending machine

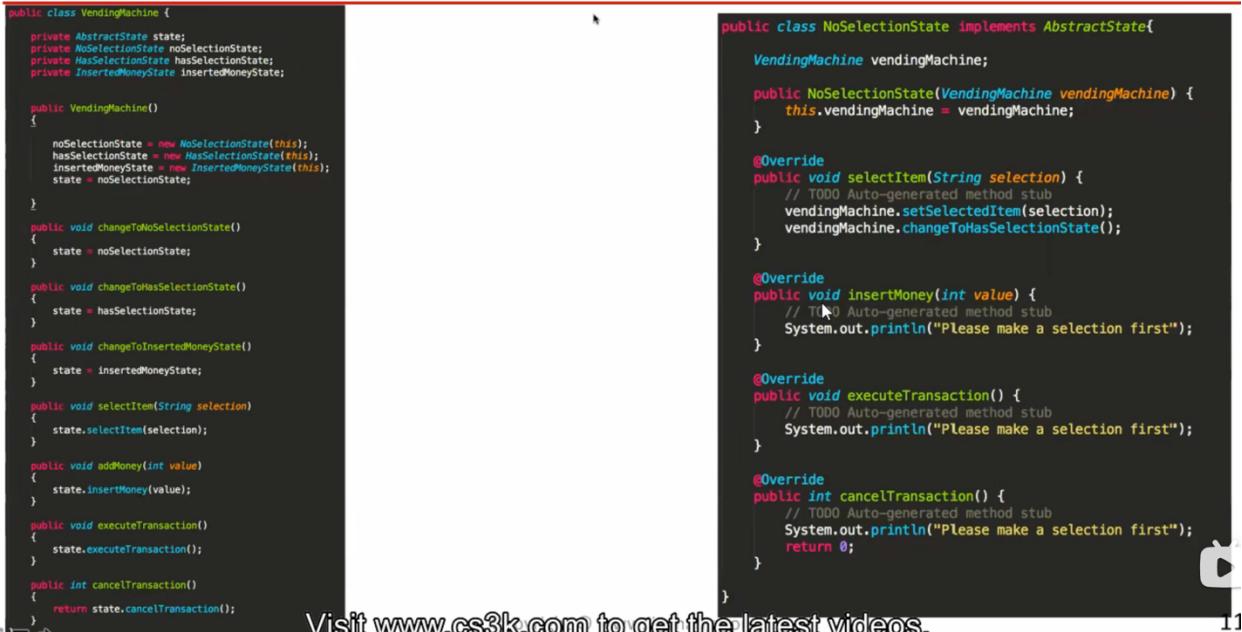
LS3K.I

```

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

```

Vending machine



```
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();
    }
}

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;
    }
}
```

Visit www.cs3k.com to get the latest videos.

11

右图第一行要有 vendingMachine 的目的是因为你要改变这个 machine 的 state 所以就必须要有这个变量

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

public class NoSelectionState implements State {
    VendingMachine vendingMachine;

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

    //Implement, and change state
    @Override
    public void selectItem(String selection) {
        vendingMachine.setSelectedItem(selection);
        vendingMachine.changeToHasSelectionState();
    }
}

public class VendingMachine {
```

```

private State state;
private NoSelectionState noSelectionState;
privte HasSelectionState hasSelectionState

//Constructor: init all state, and set initial state
public VendingMachine() {
    noSelectionState = new NoSelectionState(this);
    hasSelectionState = new HasSelectionState(this);
    state = noSelectionState;
}

//Expose to other State class for changing
public void changeToHasSelectionState() {
    state = hasSelectionState();
}

//API, use different state to implement
public void selectItem(String selection) {
    state.selectItem(selection);
}
}

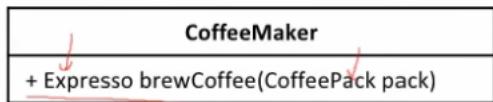
```

Coffee Machine



-
- Use case: Brew

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

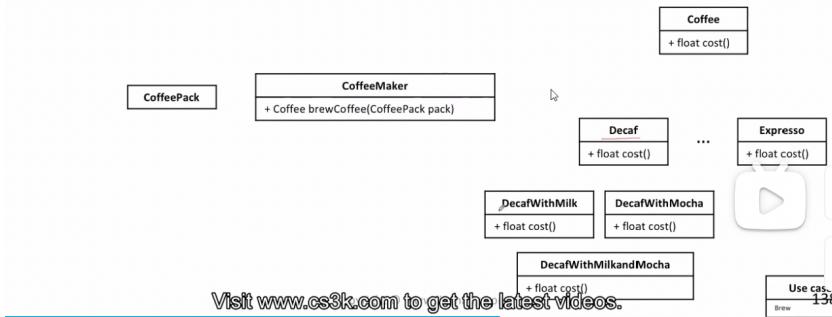


如果需要能制作出多种咖啡
(价格不同), 需要怎么做?

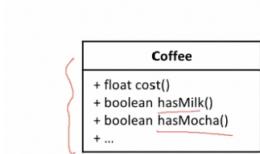
做法一：利用继承，缺点是如果后面有很多不同种的 coffee 那就必须再加新的类，

继承

CS3K.COM



解决办法是可以直接在 coffee 这个类上加 hasMilk, hasMocha 这些属性



这个缺点是首先如果加很多份 mocha 或者 milk 价格不会 adjust，而且 cost () 可能会很长，

另一种继承

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

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

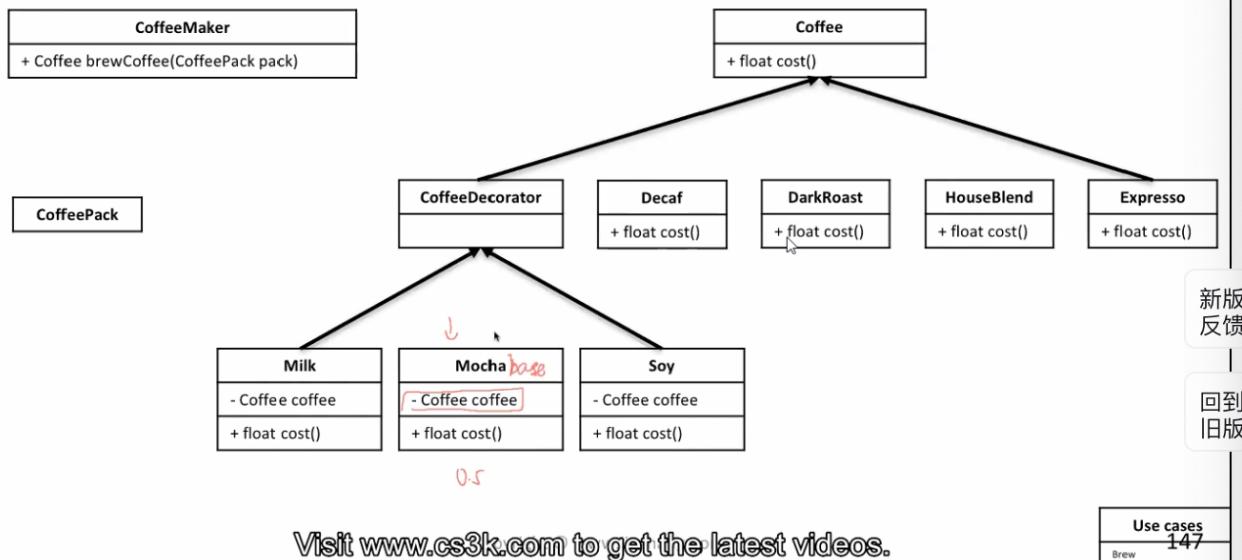
    if(hasSoy())
    {
        cost += 0.5;
    }

    ...

    return cost;
}
```

Decorator

CS3K.com



coffeeDecorator 是 Coffee 的一个子类

```
public class Expresso extends Coffee
{
    public Expresso()
    {
        description = "Expresso";
    }

    public float cost()
    {
        return 1.99;
    }
}
```

```
public class Mocha extends CoffeeDecorator
{
    Coffee coffee;

    public Mocha(Coffee coffee)
    {
        this.coffee = coffee;
    }

    public String getDescription()
    {
        return coffee.getDescription() + ", Mocha";
    }

    public float cost()
    {
        return coffee.getCost() + 0.5;
    }
}
```

```

public void test()
{
    Coffee coffee_1 = new Espresso();

    System.out.println(coffee_1.getDescription() + " $" + coffee_1.getCost());

    Coffee coffee_2 = new Mocha(coffee_1);

    System.out.println(coffee_2.getDescription() + " $" + coffee_2.getCost());

    Coffee coffee_3 = new Mocha(coffee_2);

    System.out.println(coffee_3.getDescription() + " $" + coffee_3.getCost());
}

```

```

Expresso $ 1.99
Expresso, Mocha $ 2.49
Expresso, Mocha, Mocha $ 2.99
,
```

Decorator Design Pattern

```

public interface Coffee {
    public double getCost();
    public String getDescription();
}

public class Espresso implements Coffee {
    public Espresso() {
        description = "Expresso";
    }

    public float getCost() {
        return 1.99;
    }

    public String getDescription() {
        return description;
    }
}

public abstract class CoffeeDecorator implements Coffee {
    protected final Coffee decoratedCoffee;

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

```

```

public double getCost() {
    return decoratedCoffee.getCost();
}

public double getDescription() {
    return decoratedCoffee.getDescription();
}
}

public class WithMocha extends CoffeeDecorator {
    public WithMocha(Coffee coffee) {
        super(c);
    }

    public String getDescription() {
        return super.getDescription() + ", Mocha";
    }

    public float getCost() {
        return super.getCost() + 0.5;
    }
}

public void test() {
    Coffee c = new Espresso();
    c = new WithMocha(c);
    c = new WithMocha(c);
    // Espresso, Mocha, Mocha $2.99
    System.out.println(c.getDescription() + "$ " + c.getCost());
}

```

Kindle

Factory Design Pattern

- Simple Factory: 将 if-else 放入一个单独的 factory 里面
- Factory method
- Abstract Factory

Clarify

- What

关键字: Kindle, Book

关键字: Kindle



需不需要设计不同版本?

- Design: get price
- Design: Memory difference



关键字: Book



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



如何获取电子书?

- 是否支持Upload
- 是否支持Download



对于付费的电子书, 提供哪些支付功能?

Payment -> Strategy design pattern

Clarify
OOD 面向对象设计专题班



- 对于本题: 支持上传, 下载
- 对于本题: 不需要考虑付费

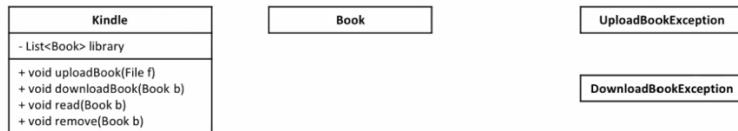
Core Object

UML Class Diagram



Classes

UML Class Diagram



不好的方案：

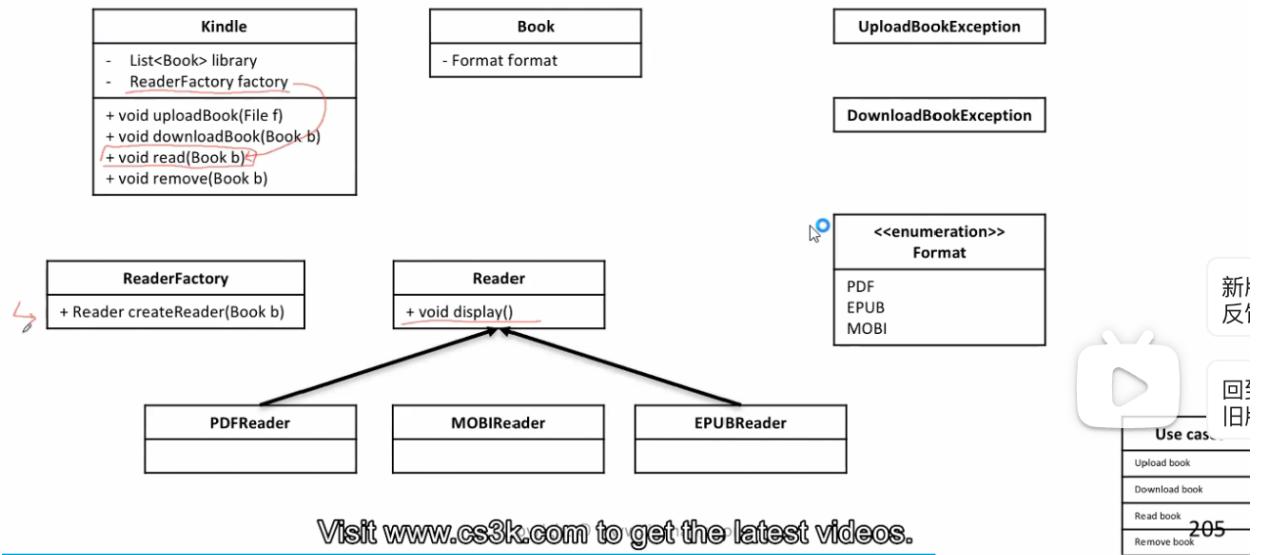
- How would read book work?

```
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();
    }
}
```

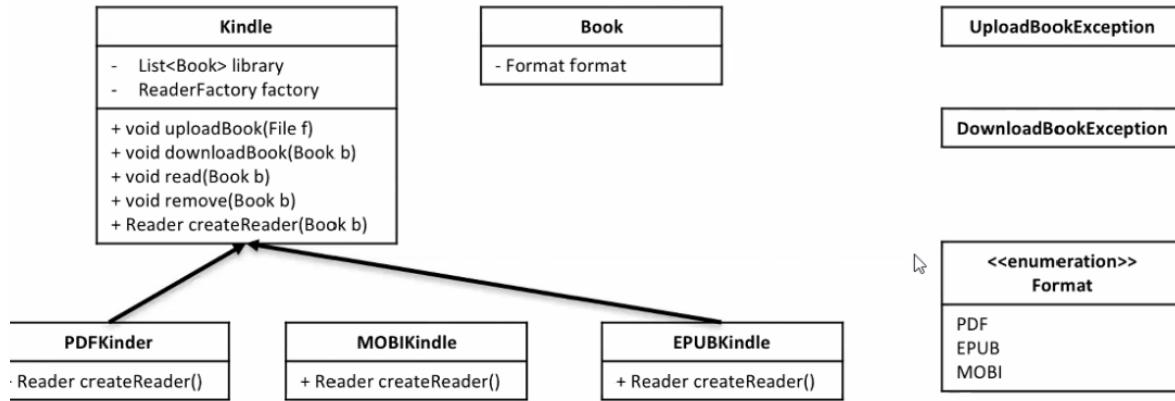
2:03:01 / 2:48:50 1080P 高清

- Solution: Factory design pattern

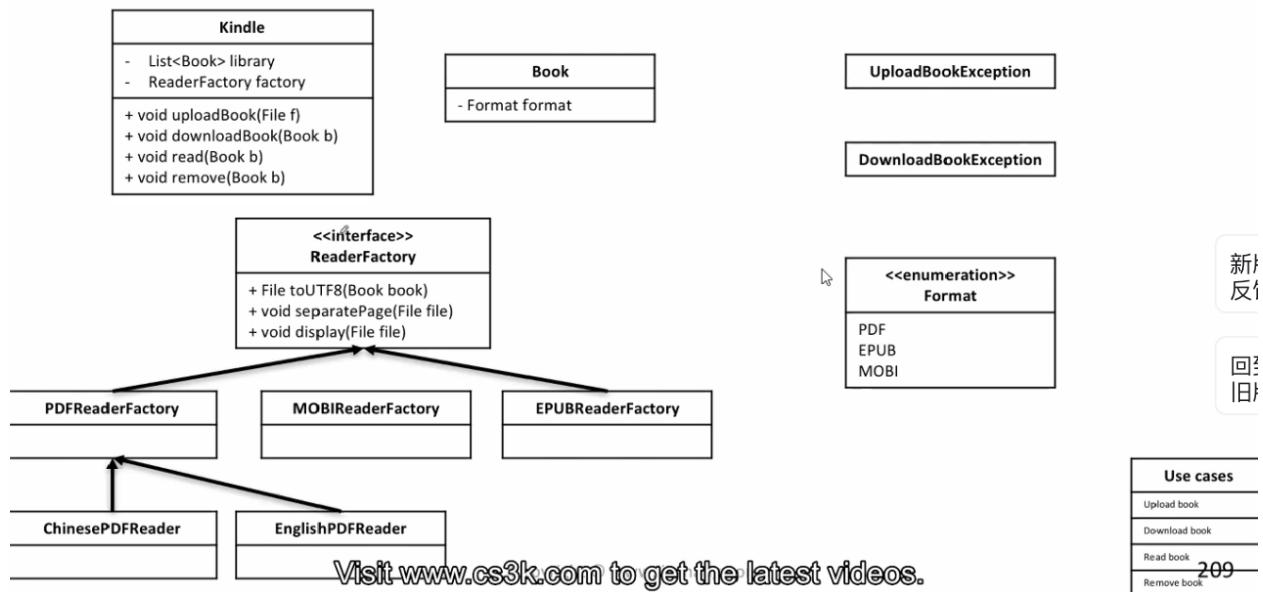
Factory design pattern



Factory method



Abstract factory



Visit www.cs3k.com to get the latest videos.

新
反

回
旧

Use cases
Upload book
Download book
Read book
Remove book
209