DISTRIBUTED VENDING MACHINE (DVM)

1st Cycle development

Project Team: Team 2

Date: 2021.05.11

Team Members :

201310513 황인우 / 201311255 최우석 / 201512265 박인우 / 201711306 박정현

Contents

Activity 2051. Implement Class & Method Definitions

Type	Class
Name	DVM
Purpose	DVM의 주요 기능 수행
Overview	DVM의 주요 기능들을 수행한다.
Cross Reference	Function : All Use case : All
Exceptional Course of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	getDist
Purpose	현재 DVM에 저장된 dist 정보를 리턴한다.
Cross Reference	12. Get None Item Location
Input	void
Output	double
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	setDist
Purpose	기기의 dist 값을 세트한다.
Cross Reference	12. Get None Item Location
Input	double
Output	void
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	getRegion
Purpose	기기의 region 값을 리턴한다.
Cross Reference	12. Get None Item Location
Input	void
Output	String
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	getLocation
Purpose	기기의 Location값을 리턴한다.
Cross Reference	12. Get None Item Location
Input	void
Output	Pair
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	getItemList
Purpose	기기의 itemList값을 리턴한다.
Cross Reference	2. Choose item
Input	void
Output	ArrayList<Item>
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	getCodeTable
Purpose	기기의 codeTable값을 리턴한다.
Cross Reference	5. Use code
Input	void
Output	HashMap<String,String>
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	getItemFromName
Purpose	아이템 Name을 기반으로 DVM에 저장된 item객체를 반환한다.
Cross Reference	10. Give code
Input	String
Output	Item
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	getAnotherDVMInfo
Purpose	주어진 Item값의 재고가 존재하는 다른 DVM의 List를 반환한다.
Cross Reference	12. Get None Item Location
Input	Item
Output	ArrayList<DVM>
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	removeCode
Purpose	codeTable에서 code를 제거한다.
Cross Reference	11. Give code
Input	code
Output	void
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	sortDVM
Purpose	DVM List를 현 DVM에서 가까운 순서대로 정렬한다.
Cross Reference	12. Get None Item Location
Input	ArrayList<DVM>
Output	ArrayList<DVM>
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	getNoneItemLocation
Purpose	현재 DVM에는 재고가 존재하지 않는 Item이 존재하는 다른 DVM들을 거리에 가까운 순서대로 리턴한다.
Cross Reference	12. Get None Item Location
Input	String
Output	ArrayList<DVM>
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	inputCode
Purpose	code를 입력해 상품을 구입한다.
Cross Reference	5. Use code
Input	String
Output	int
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	codeValidation
Purpose	code가 유효한 code인지 확인한다.
Cross Reference	8. Code Validation
Input	String
Output	boolean
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	getItemFromCode
Purpose	code에 해당하는 Item객체를 리턴한다.
Cross Reference	5. Use code
Input	String
Output	Item
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	login
Purpose	관리자로 로그인한다.
Cross Reference	16. Login
Input	String, String
Output	boolean
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	manageAmount
Purpose	특정 상품의 재고를 변경한다.
Cross Reference	17. Manage Amount
Input	String, int
Output	boolean
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	manageAmountCash
Purpose	현 DVM이 보유중인 현금을 변경한다.
Cross Reference	18. Manage Amount Cash
Input	int
Output	boolean
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	inputCash
Purpose	투입된 현금 액수를 입력하고 상품을 구입할 수 있는지 확인한다.
Cross Reference	3. Pay by cash
Input	int, Item
Output	int
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	checkTotalCash
Purpose	거스름돈이 부족한지 확인한다.
Cross Reference	6. Calculate Price (Cash)
Input	int
Output	boolean
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	insertCard
Purpose	카드번호를 입력하고 상품을 구입할 수 있는지 확인한다.
Cross Reference	4. Pay by card
Input	String, Item
Output	int
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	giveItem
Purpose	아이템을 사용자에게 반환하였다고 가정하고 개수를 줄인다.,
Cross Reference	11. Give item
Input	Item
Output	void
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	giveCode
Purpose	특정 Item에 대한 code를 생성하고 반환한다.
Cross Reference	10. Give code
Input	Item
Output	String
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Class
Name	Payment
Purpose	계산의 기능을 담당한다.
Overview	사용자의 결제방법에 따라 결제를 진행하는 역할을 한다.
Cross Reference	Function : All Use case : All
Exceptional Course of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	calculatePriceCash
Purpose	현금으로 상품구입 계산을 진행한다.
Cross Reference	6. Calculate Price (Cash)
Input	int, Item
Output	int
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	isSufficient
Purpose	현재 사용가능한 금액과 상품의 가격을 비교한다.
Cross Reference	6. Calculate Price (Cash)
Input	int, int
Output	boolean
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	calculatePriceCard
Purpose	카드로 계산을 진행한다.
Cross Reference	7. Calculate Price (Card)
Input	String, Item
Output	int
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	consumeCard
Purpose	현 카드 잔여금액에서 상품금액을 제외한다.
Cross Reference	7. Calculate Price (Card)
Input	String, int
Output	void
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	isValidCard
Purpose	입력된 카드가 유효한지 확인한다.
Cross Reference	7. Calculate Price (Card)
Input	String
Output	boolean
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Class
Name	Item
Purpose	상품의 정보를 관리한다.
Overview	상품의 이름, 가격, 재고 등을 관리한다.
Cross Reference	Function : All Use case : All
Exceptional Course of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	changeAmount
Purpose	상품의 재고를 변경한다.
Cross Reference	11. Give item
Input	int
Output	boolean
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2051. Implement Class & Method Definitions

Type	Method
Name	reduceAmount
Purpose	상품의 재고를 하나 줄인다.
Cross Reference	11. Give Item
Input	void
Output	void
Abstract Operation	N/A
Exceptional Courses of Events	N/A

Activity 2055. Write Unit Test Code

- DVMTest: getItemFromName Test

```
ArrayList<Item> myItems = new ArrayList<>();
myItems.add(new Item( itemName: "Sprite",  itemID: 1,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Mint Sprite",  itemID: 2,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Coke",  itemID: 3,  itemPrice: 1000,  itemAmount: 0));
myItems.add(new Item( itemName: "Mint Coke",  itemID: 4,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Water",  itemID: 5,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Sparkling",  itemID: 6,  itemPrice: 1000,  itemAmount: 0));
myItems.add(new Item( itemName: "Coffee",  itemID: 7,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Mint Coffee",  itemID: 8,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Milk Coffee",  itemID: 9,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Latte",  itemID: 10,  itemPrice: 1000,  itemAmount: 10));
ArrayList<Item> anotherItems1 = new ArrayList<>();
anotherItems1.add(new Item( itemName: "Sprite",  itemID: 1,  itemPrice: 1000,  itemAmount: 10));
anotherItems1.add(new Item( itemName: "Mint Sprite",  itemID: 2,  itemPrice: 1000,  itemAmount: 10));
anotherItems1.add(new Item( itemName: "Coke",  itemID: 3,  itemPrice: 1000,  itemAmount: 0));
anotherItems1.add(new Item( itemName: "Mint Coke",  itemID: 4,  itemPrice: 1000,  itemAmount: 10));
anotherItems1.add(new Item( itemName: "Water",  itemID: 5,  itemPrice: 1000,  itemAmount: 10));
anotherItems1.add(new Item( itemName: "Sparkling",  itemID: 6,  itemPrice: 1000,  itemAmount: 10));
anotherItems1.add(new Item( itemName: "Coffee",  itemID: 7,  itemPrice: 1000,  itemAmount: 10));
anotherItems1.add(new Item( itemName: "Mint Coffee",  itemID: 8,  itemPrice: 1000,  itemAmount: 10));
anotherItems1.add(new Item( itemName: "Milk Coffee",  itemID: 9,  itemPrice: 1000,  itemAmount: 10));
anotherItems1.add(new Item( itemName: "Latte",  itemID: 10,  itemPrice: 1000,  itemAmount: 10));
ArrayList<Item> anotherItems2 = new ArrayList<>();
anotherItems2.add(new Item( itemName: "Sprite",  itemID: 1,  itemPrice: 1000,  itemAmount: 10));
anotherItems2.add(new Item( itemName: "Mint Sprite",  itemID: 2,  itemPrice: 1000,  itemAmount: 10));
anotherItems2.add(new Item( itemName: "Coke",  itemID: 3,  itemPrice: 1000,  itemAmount: 0));
anotherItems2.add(new Item( itemName: "Mint Coke",  itemID: 4,  itemPrice: 1000,  itemAmount: 10));
anotherItems2.add(new Item( itemName: "Water",  itemID: 5,  itemPrice: 1000,  itemAmount: 10));
anotherItems2.add(new Item( itemName: "Sparkling",  itemID: 6,  itemPrice: 1000,  itemAmount: 10));
anotherItems2.add(new Item( itemName: "Coffee",  itemID: 7,  itemPrice: 1000,  itemAmount: 10));
anotherItems2.add(new Item( itemName: "Mint Coffee",  itemID: 8,  itemPrice: 1000,  itemAmount: 10));
anotherItems2.add(new Item( itemName: "Milk Coffee",  itemID: 9,  itemPrice: 1000,  itemAmount: 10));
anotherItems2.add(new Item( itemName: "Latte",  itemID: 10,  itemPrice: 1000,  itemAmount: 0));
```

```
Payment mainDVMPayment = new Payment();
Payment anotherDVM1Payment = new Payment();
Payment anotherDVM2Payment = new Payment();
DVM mainDVM = new DVM( region: "Main DVM",  x: 10,  y: 3,  totalCash: 500,
        adminID: "admin",  adminPW: "1234", myItems,  dist: 0, mainDVMPayment);
DVM anotherDVM = new DVM( region: "Another DVM 1",  x: 3,  y: 4,  totalCash: 5000000,
        adminID: "admin",  adminPW: "1234", anotherItems1,  dist: 0, anotherDVM1Payment);
DVM anotherDVM2 = new DVM( region: "Another DVM 2",  x: 17,  y: 15,  totalCash: 5000000,
        adminID: "admin",  adminPW: "1234", anotherItems2,  dist: 0, anotherDVM2Payment);
```

```
String[] SuccessItemNames = {"Sprite","Mint Sprite","Coke","Mint Coke","Water",
        "Sparkling","Coffee","Mint Coffee","Milk Coffee","Latte"};
//success
boolean flag = true;
for(int i=0;i<SuccessItemNames.length;i++){
    String Item_name = SuccessItemNames[i];
    if(mainDVM.getItemFromName(Item_name) != mainDVM.getItemList().get(i))flag = false;
}
//Else fail
assertEquals( expected: true,flag);
```

Activity 2055. Write Unit Test Code

- DVMTest: getAnotherDVMInfo Tests

Test를 위해 생성한 변수들 및 객체들은 getItemFromName Test와 동일함.

```
ArrayList<DVM> TestDVMList1 =  mainDVM.getAnotherDVMInfo( itemName: "Sprite");
assertEquals( expected: 2,TestDVMList1.size());
```

```
ArrayList<DVM> TestDVMList2 = mainDVM.getAnotherDVMInfo( itemName: "Coke");
assertEquals( expected: 0,TestDVMList2.size());
```

```
ArrayList<DVM> TestDVMList3 = mainDVM.getAnotherDVMInfo( itemName: "Water");
int cnt = 0;
for(int i=0;i<TestDVMList3.size();i++){
    System.out.println(TestDVMList3.get(i).region);
    cnt++;
}
System.out.println(cnt);
assertEquals( expected: 2,cnt);
```

Activity 2055. Write Unit Test Code

- DVMTest: removeCode Tests

Test를 위해 생성한 변수들 및 객체들은 getItemFromName Test와 동일함.

```
HashMap<String,String> test_table = mainDVM.getCodeTable();
Item testItem = myItems.get(0);
String code = mainDVM.giveCode(testItem);
mainDVM.removeCode(code);
String result = test_table.get(code);
assertEquals( expected: null ,result);
```

```
//Testing 부여되지 않은 코드일 경우
//런타임 에러가 나지 않는다면 flag = true
String Invalid_code = "INVALID_CODE";
boolean flag = true;
try{
    mainDVM.removeCode(Invalid_code);
}catch(RuntimeException e){
    flag = false;
}
assertEquals( expected: true ,flag);
```

Activity 2055. Write Unit Test Code

- DVMTest: sortDVM Test

Test를 위해 생성한 변수들 및 객체들은 getItemFromName Test와 동일함.

```
//sorted_list 가 오름차순이 아닐경우 flag = false.
boolean flag = true;
ArrayList<DVM> temp = mainDVM.sortDVM(mainDVM.DVMList);
for(int i=0;i<temp.size()-1;i++){
    if(temp.get(i).dist > temp.get(i+1).dist)flag = false;
}
assertEquals( expected: true,flag);
```

Activity 2055. Write Unit Test Code

- DVMTest: getNoneItemLocation Test

Test를 위해 생성한 변수들 및 객체들은 getItemFromName Test와 동일함.

```
String test_Item = "Sprite";
ArrayList<DVM> lists = mainDVM.getNoneItemLocation(test_Item);
boolean flag = true;
//스프라이트는 서로 다른 두개의 DVM 이 가지고있으므로 2개가 아니면 flag = false
if(lists.size()!=2)flag = false;
assertEquals( expected: true,flag);
```

Activity 2055. Write Unit Test Code

- DVMTest: inputCode Tests

Test를 위해 생성한 변수들 및 객체들은 getItemFromName Test와 동일함.

```
HashMap<String,String> test_table = mainDVM.getCodeTable();
Item test_item = mainDVM.getItemList().get(0);
String test_code = mainDVM.giveCode(test_item);
int success_ret_code = mainDVM.inputCode(test_code);
// 성공 = 1
assertEquals( expected: 1,success_ret_code);
```

```
HashMap<String,String> test_table = mainDVM.getCodeTable();
int Invalid_ret_code = mainDVM.inputCode("INVALID_CODE");
//등록되지 않은 쿠폰일 경우 코드 반환값 = -1
assertEquals( expected: -1,Invalid_ret_code);
```

```
HashMap<String,String> test_table = mainDVM.getCodeTable();
Item No_Amount_Item = anotherDVM2.getItemList().get(2);
String Fail_code = mainDVM.giveCode(No_Amount_Item);
int No_Amount_ret_code = mainDVM.inputCode(Fail_code);
//현재 재고가 부족하므로 코드 반환값 = 0
assertEquals( expected: 0,No_Amount_ret_code);
```

Activity 2055. Write Unit Test Code

- DVMTest: codeValidation Tests

Test를 위해 생성한 변수들 및 객체들은 getItemFromName Test와 동일함.

```
//등록된 코드일경우 success = true
HashMap<String,String> test_table = mainDVM.getCodeTable();
Item test_item = mainDVM.getItemList().get(0);
String success_test_code = mainDVM.giveCode(test_item);
boolean success = test_table.containsKey(success_test_code);
assertEquals( expected: true,success);
```

```
//등록되지 않은 코드 failed = false
HashMap<String,String> test_table = mainDVM.getCodeTable();
String failed_test_code = "INVALID_CODE";
boolean failed = test_table.containsKey(failed_test_code);
assertEquals( expected: false,failed);
```

Activity 2055. Write Unit Test Code

- DVMTest: getItemFromCode Tests

```
// DVM에 존재하는 아이템에 대한 실행
ArrayList<Item> myItems = new ArrayList<>();
Item sprite = new Item( itemName: "Sprite",  itemID: 1,  itemPrice: 1000,  itemAmount: 10);
myItems.add(sprite);
Payment mainDVMPayment = new Payment();
DVM mainDVM = new DVM( region: "Main DVM", x: 10, y: 3,  totalCash: 500,
        adminID: "admin",  adminPW: "1234", myItems, dist: 0, mainDVMPayment);
String code = mainDVM.giveCode(sprite);
Item findItem = mainDVM.getItemFromCode(code);
assertEquals(sprite, findItem);
```

```
// DVM에 존재하지 않는 아이템에 대한 실행
ArrayList<Item> myItems = new ArrayList<>();
Item sprite = new Item( itemName: "Sprite",  itemID: 1,  itemPrice: 1000,  itemAmount: 10);
Item latte = new Item( itemName: "Latte",  itemID: 10,  itemPrice: 1000,  itemAmount: 10);
myItems.add(sprite);
Payment mainDVMPayment = new Payment();
DVM mainDVM = new DVM( region: "Main DVM", x: 10, y: 3,  totalCash: 500,
        adminID: "admin",  adminPW: "1234", myItems, dist: 0, mainDVMPayment);
String code = mainDVM.giveCode(latte);
Item findItem = mainDVM.getItemFromCode(code);
assertTrue( condition: findItem.getItemName()=="null");
```

Activity 2055. Write Unit Test Code

- DVMTest: login Tests

```
//정상입력
ArrayList<Item> myItems = new ArrayList<>();
Payment mainDVMPayment = new Payment();
DVM dvm = new DVM( region: "Main DVM", x: 10, y: 3, totalCash: 500,
        adminID: "admin", adminPW: "1234", myItems, dist: 0, mainDVMPayment);
String id = "admin"; String pw = "1234";
assertTrue(dvm.login(id, pw));
```

```
//pw만 다를 경우
ArrayList<Item> myItems = new ArrayList<>();
Payment mainDVMPayment = new Payment();
DVM dvm = new DVM( region: "Main DVM", x: 10, y: 3, totalCash: 500,
        adminID: "admin", adminPW: "1234", myItems, dist: 0, mainDVMPayment);
String id = "admin"; String pw = "134";
assertTrue(!dvm.login(id, pw));
```

```
//id만 다를 경우
ArrayList<Item> myItems = new ArrayList<>();
Payment mainDVMPayment = new Payment();
DVM dvm = new DVM( region: "Main DVM", x: 10, y: 3, totalCash: 500,
        adminID: "admin", adminPW: "1234", myItems, dist: 0, mainDVMPayment);
String id = "ad"; String pw = "1234";
assertTrue(!dvm.login(id, pw));
```

```
//id, pw 둘다 다를 경우
ArrayList<Item> myItems = new ArrayList<>();
Payment mainDVMPayment = new Payment();
DVM dvm = new DVM( region: "Main DVM", x: 10, y: 3, totalCash: 500,
        adminID: "admin", adminPW: "1234", myItems, dist: 0, mainDVMPayment);
String id = "ad"; String pw = "134";
assertTrue(!dvm.login(id, pw));
```

Activity 2055. Write Unit Test Code

- DVMTest: manageAmount Tests

```
ArrayList<Item> myItems = new ArrayList<>();
myItems.add(new Item( itemName: "Sprite",  itemID: 1,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Mint Sprite",  itemID: 2,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Coke",  itemID: 3,  itemPrice: 1000,  itemAmount: 0));
myItems.add(new Item( itemName: "Mint Coke",  itemID: 4,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Water",  itemID: 5,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Sparkling",  itemID: 6,  itemPrice: 1000,  itemAmount: 0));
myItems.add(new Item( itemName: "Coffee",  itemID: 7,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Mint Coffee",  itemID: 8,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Milk Coffee",  itemID: 9,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Latte",  itemID: 10,  itemPrice: 1000,  itemAmount: 10));
Payment mainDVMPayment = new Payment();
DVM dvm = new DVM( region: "Main DVM",  x: 10,  y: 3,  totalCash: 500,
        adminID: "admin",  adminPW: "1234", myItems,  dist: 0, mainDVMPayment);
```

```
//정상입력
String ItemName = "Coke"; int newAmount = 12;
assertTrue(dvm.manageAmount(ItemName, newAmount));
```

```
//아이템 이름 오류
String ItemName = "Cola"; int newAmount = 12;
assertTrue(!dvm.manageAmount(ItemName, newAmount));
```

```
//입력 수량 오류
String ItemName = "Coke"; int newAmount = -3;
assertTrue(!dvm.manageAmount(ItemName, newAmount));
```

```
//이름,수량 입력 오류
String ItemName = "Cola"; int newAmount = -3;
assertTrue(!dvm.manageAmount(ItemName, newAmount));
```

Activity 2055. Write Unit Test Code

- DVMTest: manageAmountCash Tests

```
ArrayList<Item> myItems = new ArrayList<>();
Payment mainDVMPayment = new Payment();
DVM dvm = new DVM( region: "Main DVM", x: 10, y: 3, totalCash: 500,
        adminID: "admin", adminPW: "1234", myItems, dist: 0, mainDVMPayment);
//newAmountCash>0
int newAmountCash = 5000;
assertTrue(dvm.manageAmountCash(newAmountCash));
```

```
//newAmountCash<0
int newAmountCash = -5000;
assertTrue(!dvm.manageAmountCash(newAmountCash));
```

Activity 2055. Write Unit Test Code

- DVMTest: inputCash Tests

```
ArrayList<Item> myItems = new ArrayList<>();
myItems.add(new Item( itemName: "Sprite",  itemID: 1,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Mint Sprite",  itemID: 2,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Coke",  itemID: 3,  itemPrice: 1000,  itemAmount: 0));
myItems.add(new Item( itemName: "Mint Coke",  itemID: 4,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Water",  itemID: 5,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Sparkling",  itemID: 6,  itemPrice: 1000,  itemAmount: 0));
myItems.add(new Item( itemName: "Coffee",  itemID: 7,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Mint Coffee",  itemID: 8,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Milk Coffee",  itemID: 9,  itemPrice: 1000,  itemAmount: 10));
myItems.add(new Item( itemName: "Latte",  itemID: 10,  itemPrice: 1000,  itemAmount: 10));
```

```
//정상입력
Payment mainDVMPayment = new Payment();
DVM dvm = new DVM( region: "Main DVM",  x: 10,  y: 3,  totalCash: 1500,
        adminID: "admin",  adminPW: "1234", myItems,  dist: 0, mainDVMPayment);
int inputCash = 2000;
Item selectedItem = new Item( itemName: "Sprite",  itemID: 1,  itemPrice: 1000,  itemAmount: 10);
int result = dvm.inputCash(inputCash, selectedItem);
assertEquals( expected: 1000,result);
```

```
//거스름돈 부족
Payment mainDVMPayment = new Payment();
DVM dvm = new DVM( region: "Main DVM",  x: 10,  y: 3,  totalCash: 500,
        adminID: "admin",  adminPW: "1234", myItems,  dist: 0, mainDVMPayment);
int inputCash = 2000;
Item selectedItem = new Item( itemName: "Sprite",  itemID: 1,  itemPrice: 1000,  itemAmount: 10);
int result = dvm.inputCash(inputCash, selectedItem);
assertEquals( expected: -2,result);
```

```
//투입금액 부족
Payment mainDVMPayment = new Payment();
DVM dvm = new DVM( region: "Main DVM",  x: 10,  y: 3,  totalCash: 1500,
        adminID: "admin",  adminPW: "1234", myItems,  dist: 0, mainDVMPayment);
int inputCash = 500;
Item selectedItem = new Item( itemName: "Sprite",  itemID: 1,  itemPrice: 1000,  itemAmount: 10);
int result = dvm.inputCash(inputCash, selectedItem);
assertEquals( expected: -1,result);
```

Activity 2055. Write Unit Test Code

- DVMTest: checkTotalCash Tests

```
ArrayList<Item> myItems = new ArrayList<>();
Item sprite = new Item( itemName: "Sprite",  itemID: 1,  itemPrice: 1000,  itemAmount: 10);
myItems.add(sprite);
Payment mainDVMPayment = new Payment();
DVM mainDVM = new DVM( region: "Main DVM", x: 10, y: 3,  totalCash: 10000,
        adminID: "admin",  adminPW: "1234", myItems, dist: 0, mainDVMPayment);
```

```
// change > totalCash
boolean result = mainDVM.checkTotalCash( change: 5000000);
assertTrue(result);
```

```
// change <= totalCash
boolean result = mainDVM.checkTotalCash( change: 500);
assertFalse(result);
```

Activity 2055. Write Unit Test Code

- DVMTest: insertCard Tests

```
// 정상구매
ArrayList<Item> myItems = new ArrayList<>();
Item sprite = new Item( itemName: "Sprite",  itemID: 1,  itemPrice: 1000,  itemAmount: 10);
myItems.add(sprite);
Payment mainDVMPayment = new Payment();
DVM mainDVM = new DVM( region: "Main DVM", x: 10, y: 3,  totalCash: 500,
        adminID: "admin",  adminPW: "1234", myItems, dist: 0, mainDVMPayment);
int result = mainDVM.insertCard( cardNumber: "1234", sprite);
assertEquals( expected: 1, result);
```

```
// 잔액부족
ArrayList<Item> myItems = new ArrayList<>();
Item sprite = new Item( itemName: "Sprite",  itemID: 1,  itemPrice: 999999,  itemAmount: 10);
myItems.add(sprite);
Payment mainDVMPayment = new Payment();
DVM mainDVM = new DVM( region: "Main DVM", x: 10, y: 3,  totalCash: 500,
        adminID: "admin",  adminPW: "1234", myItems, dist: 0, mainDVMPayment);
int result = mainDVM.insertCard( cardNumber: "3572", sprite);
assertEquals( expected: 0, result);
```

```
// 잘못된카드번호
ArrayList<Item> myItems = new ArrayList<>();
Item sprite = new Item( itemName: "Sprite",  itemID: 1,  itemPrice: 999999,  itemAmount: 10);
myItems.add(sprite);
Payment mainDVMPayment = new Payment();
DVM mainDVM = new DVM( region: "Main DVM", x: 10, y: 3,  totalCash: 500,
        adminID: "admin",  adminPW: "1234", myItems, dist: 0, mainDVMPayment);
int result = mainDVM.insertCard( cardNumber: "이상한번호", sprite);
assertEquals( expected: -1, result);
```

Activity 2055. Write Unit Test Code

- DVMTest: giveItem Tests

```
ArrayList<Item> myItems = new ArrayList<>();
Item sprite = new Item( itemName: "Sprite", itemID: 1, itemPrice: 1000, itemAmount: 10);
Item mintSprite = new Item( itemName: "Mint Sprite", itemID: 2, itemPrice: 1000, itemAmount: 10);
Item coke = new Item( itemName: "Coke", itemID: 3, itemPrice: 1000, itemAmount: 0);
Item mindCoke = new Item( itemName: "Mint Coke", itemID: 4, itemPrice: 1000, itemAmount: 10);
Item water = new Item( itemName: "Water", itemID: 5, itemPrice: 1000, itemAmount: 10);
Item sparkling = new Item( itemName: "Sparkling", itemID: 6, itemPrice: 1000, itemAmount: 0);
Item coffee = new Item( itemName: "Coffee", itemID: 7, itemPrice: 1000, itemAmount: 10);
Item mintCoffee = new Item( itemName: "Mint Coffee", itemID: 8, itemPrice: 1000, itemAmount: 10);
Item milkCoffee = new Item( itemName: "Milk Coffee", itemID: 9, itemPrice: 1000, itemAmount: 10);
Item latte = new Item( itemName: "Latte", itemID: 10, itemPrice: 1000, itemAmount: 10);
myItems.add(sprite);
myItems.add(mintSprite);
myItems.add(coke);
myItems.add(mindCoke);
myItems.add(water);
myItems.add(sparkling);
myItems.add(coffee);
myItems.add(mintCoffee);
myItems.add(milkCoffee);
myItems.add(latte);
Payment mainDVMPayment = new Payment();
DVM mainDVM = new DVM( region: "Main DVM", x: 10, y: 3, totalCash: 500,
        adminID: "admin", adminPW: "1234", myItems, dist: 0, mainDVMPayment);
```

```
// 재고가 떨어지는지 확인
assertEquals( expected: 10, sprite.getItemAmount());
mainDVM.giveItem(sprite);
assertEquals( expected: 9, sprite.getItemAmount());
mainDVM.giveItem(sprite);
assertEquals( expected: 8, sprite.getItemAmount());
```

```
// 0이하로 떨어지지 않는지 확인
for (int i=0; i<100; i++) {
    mainDVM.giveItem(sprite);
}
assertEquals( expected: 0, sprite.getItemAmount());
```

Activity 2055. Write Unit Test Code

- DVMTest: giveCode Tests

Test를 위해 생성한 변수들 및 객체들은 giveItem Tests와 동일함.

```
// 코드 생성확인 + 다른 상품에 대한 코드가 다르게 도출되는지 확인
String code = mainDVM.giveCode(sprite);
boolean validity = mainDVM.codeValidation(code);
assertTrue(validity);
String code2 = mainDVM.giveCode(mintSprite);
boolean validity2 = mainDVM.codeValidation(code);
assertTrue(validity2);
assertFalse( condition: code==code2);
```

```
// 코드 생성확인 + 같은 상품에 대한 코드도 다르게 도출되는지 확인
String code = mainDVM.giveCode(sprite);
boolean validity = mainDVM.codeValidation(code);
assertTrue(validity);
String code2 = mainDVM.giveCode(sprite);
boolean validity2 = mainDVM.codeValidation(code2);
assertTrue(validity2);
assertFalse( condition: code==code2);
```

Activity 2055. Write Unit Test Code

- ItemTest: changeAmount Tests

```
// 양수입력
Item testItem = new Item( itemName: "Test", itemID: 1, itemPrice: 1000, itemAmount: 10);
testItem.changeAmount( itemName: "Test",  newAmount: 5);
assertEquals( expected: 5, testItem.getItemAmount());
```

```
// 음수로 입력할 경우
Item testItem = new Item( itemName: "Test", itemID: 1, itemPrice: 1000, itemAmount: 10);
testItem.changeAmount( itemName: "Test",  newAmount: -1);
assertEquals( expected: 10, testItem.getItemAmount());
```

Activity 2055. Write Unit Test Code

- ItemTest: reduceAmount Tests

```
// 줄어드는지 테스트
Item testItem = new Item( itemName: "Test", itemID: 1, itemPrice: 1000, itemAmount: 10);
testItem.reduceAmount();
assertEquals( expected: 9, testItem.getItemAmount());
```

```
// 여러번했을때 0이하로 안떨어지는지 테스트
Item testItem = new Item( itemName: "Test", itemID: 1, itemPrice: 1000, itemAmount: 10);
for (int i=0; i<100; i++) {
    testItem.reduceAmount();
}
assertEquals( expected: 0, testItem.getItemAmount());
```

Activity 2055. Write Unit Test Code

- PaymentTest: calculatePriceCash Tests

```
//정상경우
Payment myPayment  = new Payment();
int inputCash = 2000; Item selectedItem = new Item( itemName: "Coke",  itemID: 1234,  itemPrice: 1200,  itemAmount: 3);
int result = myPayment.calculatePriceCash(inputCash, selectedItem);
assertEquals( expected: 800, result);
```

```
//입금이 부족한 경우
Payment myPayment  = new Payment();
int inputCash = 1000; Item selectedItem = new Item( itemName: "Coke",  itemID: 1234,  itemPrice: 1200,  itemAmount: 3);
int result = myPayment.calculatePriceCash(inputCash, selectedItem);
assertTrue( condition: result == -1);
```

```
//itemPrice 가 음수일 경우
Payment myPayment  = new Payment();
int inputCash = 0; Item selectedItem = new Item( itemName: "Coke",  itemID: 1234,  itemPrice: -1200,  itemAmount: 3);
int result = myPayment.calculatePriceCash(inputCash, selectedItem);
assertTrue( condition: result == -2);
```

Activity 2055. Write Unit Test Code

- PaymentTest: isSufficient Tests

```
//정상경우
Payment myPayment  = new Payment();
int limit = 2000; int itemPrice = 1200;
boolean result = myPayment.isSufficient(limit, itemPrice);
assertTrue(result);
```

```
//입금이 부족한 경우
Payment myPayment  = new Payment();
int limit = 2000; int itemPrice = 2200;
boolean result = myPayment.isSufficient(limit, itemPrice);
assertTrue(!result);
```

Activity 2055. Write Unit Test Code

- PaymentTest: calculateChange Test

```
//정상경우
Payment myPayment  = new Payment();
int inputCash = 2000; int itemPrice = 1200;
int result = myPayment.calculateChange(inputCash, itemPrice);
assertEquals( expected: 800,result);
```

Activity 2055. Write Unit Test Code

- PaymentTest: calculatePriceCard Tests

```
//정상 케이스
Payment myPayment  = new Payment();
String inputCardNumber = "1234"; Item selectedItem = new Item( itemName: "Coke",  itemID: 1234,  itemPrice: 1200,  itemAmount: 3);
int result = myPayment.calculatePriceCard(inputCardNumber, selectedItem);
assertEquals( expected: 1,result);
```

```
//블랙카드
Payment myPayment  = new Payment();
String inputCardNumber = "0000"; Item selectedItem = new Item( itemName: "Coke",  itemID: 1234,  itemPrice: 1200,  itemAmount: 3);
int result = myPayment.calculatePriceCard(inputCardNumber, selectedItem);
assertEquals( expected: -1,result);
```

```
//잔액부족
Payment myPayment  = new Payment();
String inputCardNumber = "3572"; Item selectedItem = new Item( itemName: "Coke",  itemID: 1234,  itemPrice: 1200,  itemAmount: 3);
int result = myPayment.calculatePriceCard(inputCardNumber, selectedItem);
assertEquals( expected: 0,result);
```

```
//아이템가격 음수
Payment myPayment  = new Payment();
String inputCardNumber = "3572"; Item selectedItem = new Item( itemName: "Coke",  itemID: 1234,  itemPrice: -1200,  itemAmount: 3);
int result = myPayment.calculatePriceCard(inputCardNumber, selectedItem);
assertEquals( expected: -2,result);
```

Activity 2055. Write Unit Test Code

- PaymentTest: consumeCard Test

```
//정상경우
Payment myPayment  = new Payment();
String cardNumber = "0088"; int itemPrice = 1200;
myPayment.consumeCard(cardNumber, itemPrice);
int remain = Payment.validCardList.get("0088");
assertEquals( expected: 2800,remain);
```

Activity 2055. Write Unit Test Code

- PaymentTest: isValidCard Tests

```
//정상경우
Payment myPayment  = new Payment();
String cardNumber = "1234";
boolean result = myPayment.isValidCard( inputCardNumber: "1234");
assertTrue(result);
```

```
//유효하지 않은 카드인 경우
Payment myPayment  = new Payment();
String cardNumber = "0000";
boolean result = myPayment.isValidCard( inputCardNumber: "0000");
assertTrue(!result);
```

Activity 2061. Unit Testing

Test Summary

54	0	0	0.080s
tests	failures	ignored	duration

100%
successful

Packages **Classes**

Class	Tests	Failures	Ignored	Duration	Success rate
DVMTest	37	0	0	0.077s	100%
ItemTest	4	0	0	0.001s	100%
PaymentTest	13	0	0	0.002s	100%

Generated by Gradle 6.1.1 at May 10, 2021, 4:15:45 PM

Activity 2063. System Testing

Test No.	Test 항목	Test Description	Use Case Name	Pass
1-1	최초 화면 GUI 출력 Test	시스템이 사용자가 상품 구매, 쿠폰 사용, 관리자 로그인 기능 중 하나를 선택할 수 있는 화면을 출력하는지 확인한다.	1. Show Menu	P
2-1	상품 리스트 GUI 출력 Test	사용자가 상품 선택을 활성화시킬 경우 자판기에 등록되어 있는 상품 목록이 출력되는지 확인한다.	2. Choose Item	P
2-2	상품 리스트 GUI 출력 Test	시스템이 등록된 상품들의 이름과 재고 현황을 출력하는지 확인한다.		P
2-3	결제방법 선택 GUI 출력 Test	재고가 있는 상품을 선택했을 때 결제 선택 화면으로 넘어가는지 확인한다.		P
2-4	상품 재고가 없는 경우 안내 GUI 출력 Test	재고가 없는 상품을 선택했을 때 사용자에게 선결제 확인 메시지를 출력하고 확인 버튼을 누르면 재고가 없는 상품에 대한 기능 선택 화면으로 넘어가는지 확인한다.		P

Activity 2063. System Testing

Test No.	Test 항목	Test Description	Use Case Name	Pass
3-1	현금 결제 GUI 출력 및 기능 수행 Test	사용자가 지급한 현금이 시스템에 지급되는지 확인한다.	3. Pay by Cash	P
3-2	현금 결제 GUI 출력 및 기능 수행 Test	사용자가 현금 지급을 완료한 후 투입한 현금이 상품들의 값보다 적을 경우 시스템이 현금을 받을 준비를 다시 수행하는지 확인한다.		P
4-1	카드 결제 GUI 출력 및 기능 수행 Test	사용자가 투입한 카드가 시스템에 투입되는지 확인한다.	4. Pay by Card	P
4-2	카드 결제 GUI 출력 및 기능 수행 Test	사용자가 카드를 투입한 후 잔액/한도가 낮으면 시스템이 다른 카드를 투입하라고 하는지 확인한다.		P
5-1	쿠폰 사용 GUI 출력 및 기능 수행 Test	사용자가 코드를 입력할 수 있는지 확인한다.	5. Use code	P
5-2	쿠폰 사용 GUI 출력 및 기능 수행 Test	정상적인 코드가 아니면 다시 입력받는지 확인한다.		P

Activity 2063. System Testing

Test No.	Test 항목	Test Description	Use Case Name	Pass
6-1	현금 결제 기능 수행 Test	시스템이 사용자가 투입한 금액과 상품의 금액을 기반으로 거스름돈을 계산하는지 확인한다.	6. Calculate Price (Cash)	P
6-2	현금 결제 기능 수행 Test	결제가 성공적이면 Choose item or code으로 넘어가는지 확인한다.		P
6-3	현금 결제 기능 수행 Test	현금의 합이 적으면 Pay by Cash로 돌아가는지 확인한다.		P
7-1	카드 결제 기능 수행 Test	시스템이 카드의 잔액/한도를 체크하고 상품의 값과 비교하는지 확인한다.	7. Calculate Price (Card)	P
7-2	카드 결제 기능 수행 Test	결제가 성공적이면 Choose item or code으로 넘어가는지 확인한다.		P
7-3	카드 결제 기능 수행 Test	카드의 잔액/한도가 낮으면 Pay by Card로 돌아가는지 확인한다.		P

Activity 2063. System Testing

Test No.	Test 항목	Test Description	Use Case Name	Pass
8-1	쿠폰 사용 기능 수행 Test	코드의 유효성을 확인하는지 체크한다.	8. Code validation	P
8-2	쿠폰 사용 기능 수행 Test	유효하지 않은 코드를 입력하면 Use code로 돌아가는지 확인한다.		P
8-3	쿠폰 사용 기능 수행 Test	결제가 성공적이면 Give item으로 넘어가는지 확인한다.		P
9-1	현금 결제, 카드 결제 및 쿠폰 사용 기능 수행 Test	시스템이 사용자가 재고가 있는 상품을 선택했는지 없는 상품을 선택했는지 체크하는지 확인한다.	9. Choose item or code	P
9-2	현금 결제, 카드 결제 및 쿠폰 사용 기능 수행 Test	재고가 있는 상품을 선택하면 Give item으로 넘어가는지 확인한다.		P
9-3	현금 결제, 카드 결제 및 쿠폰 사용 기능 수행 Test	재고가 없는 상품을 선택하면 Give code으로 넘어가는지 확인한다.		P

Activity 2063. System Testing

Test No.	Test 항목	Test Description	Use Case Name	Pass
10-1	상품 재고가 없는 경우의 결제 기능 수행 Test	시스템이 사용가능한 코드를 반환하는지 확인한다.	10. Give code	P
11-1	상품 재고가 있는 경우의 결제 기능 수행 Test	시스템이 사용자가 선택한 상품을 반환하는지 확인한다.	11. Give Item	P
11-2	상품 재고가 없는 경우의 결제 기능 수행 Test	시스템이 제공한 상품에 대해 제공한 수량만큼 시스템 내부 재고에서 차감하는지 확인한다.		P
12-1	상품 재고가 없는 경우 안내 GUI 출력 Test	사용자가 재고가 없는 상품을 보유한 다른 DVM의 위치 보기를 재고가 없는 상품에 대해서만 수행 가능한지 확인한다.	12. Get None Item Location	P
12-2	상품 재고가 없는 경우 안내 GUI 출력 Test	사용자가 재고가 없는 상품을 보유한 다른 DVM의 위치 보기를 선택하면 시스템에 Get Another DVM Info 실행 요청이 이루어지는지 확인한다.		P

Activity 2063. System Testing

Test No.	Test 항목	Test Description	Use Case Name	Pass
13-1	현재 DVM에 재고 없는 물품에 대한 다른 DVM 재고 안내 기능 수행 Test	시스템이 다른 DVM 모두에 요청을 보내는지 확인한다.	13. Get Another DVM Info	P
14-1	현재 DVM에 재고 없는 물품에 대한 다른 DVM 재고 안내 기능 수행 Test	해당 시스템이 다른 DVM으로부터 정보 요청을 받는지 확인한다.	14. Give DVM Info	P
14-2	현재 DVM에 재고 없는 물품에 대한 다른 DVM 재고 안내 기능 수행 Test	해당 시스템이 요청받은 재고 유무를 파악하는지 확인한다.		P
14-3	현재 DVM에 재고 없는 물품에 대한 다른 DVM 재고 안내 기능 수행 Test	해당 시스템에 재고가 있을 시 정보를 요청한 DVM에 자신의 위치정보와 재고현황을 알려주는지 확인한다.		P

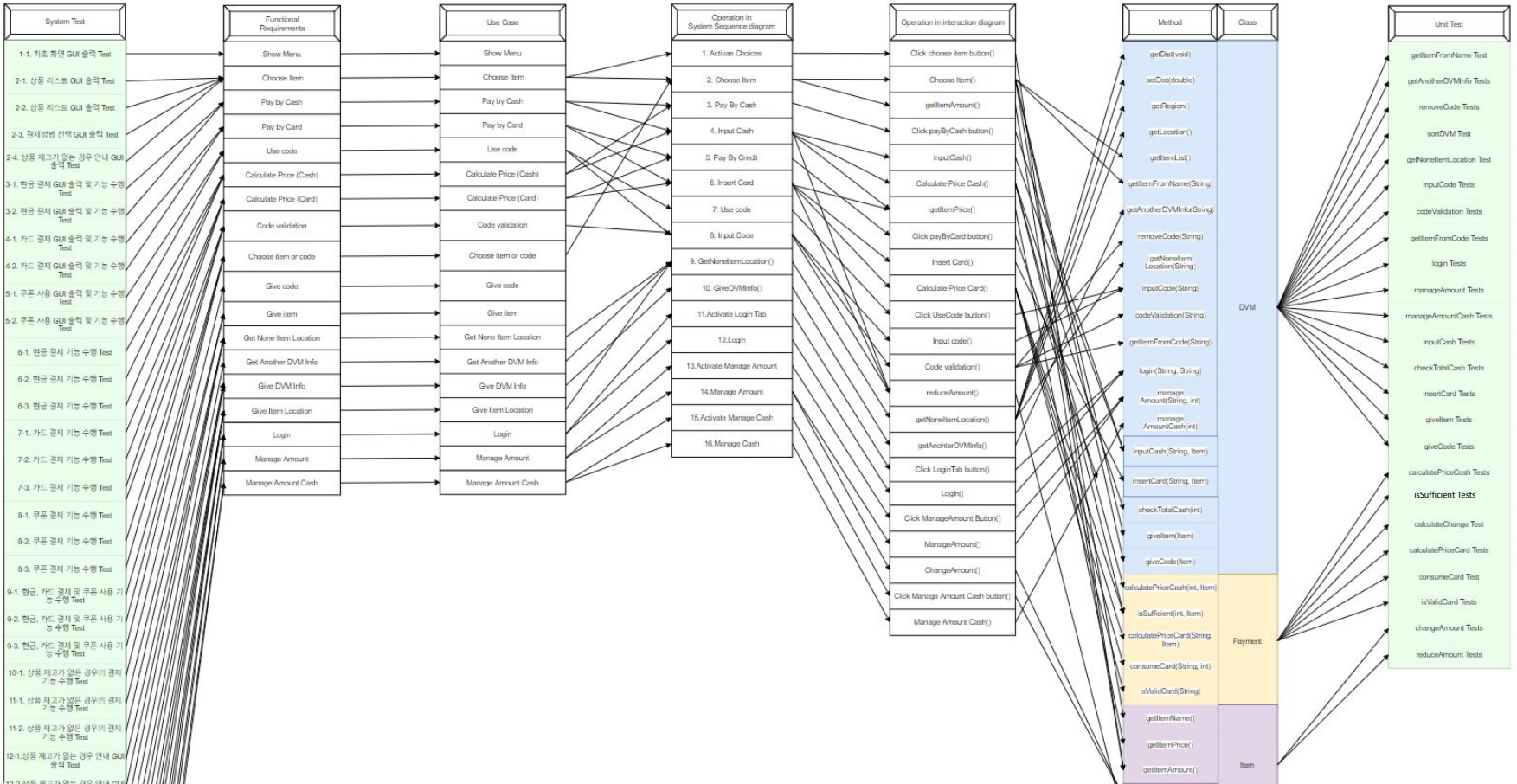
Activity 2063. System Testing

Test No.	Test 항목	Test Description	Use Case Name	Pass
15-1	현재 DVM에 재고 없는 물품에 대한 다른 DVM 재고 안내 기능 수행 Test	시스템이 다른 DVM으로부터 정보를 전달받아 재고가 존재하는 DVM과 재고가 없는 DVM을 구분하는지 확인한다.	15. Give Item Location	P
15-2	현재 DVM에 재고 없는 물품에 대한 다른 DVM 재고 안내 기능 수행 Test	재고가 있는 다른 DVM이 없을 경우 시스템이 재고가 존재하는 자판기가 없음을 반환하는지 확인한다.		P
15-3	현재 DVM에 재고 없는 물품에 대한 다른 DVM 재고 안내 기능 수행 Test	시스템이 재고가 있는 DVM을 시스템에서 가까운 순서대로 정렬하는지 확인한다.		P
15-4	현재 DVM에 재고 없는 물품에 대한 다른 DVM 재고 안내 GUI 출력 Test	시스템이 정렬된 DVM 목록을 출력하는지 확인한다.		P

Activity 2063. System Testing

Test No.	Test 항목	Test Description	Use Case Name	Pass
16-1	관리자 로그인 GUI 출력 Test	관리자가 로그인 입력부를 활성화시킬 경우 활성화가 진행되는지 확인한다.	16. Login	P
16-2	관리자 로그인 기능 수행 Test	시스템이 관리자의 아이디와 비밀번호를 체크하여 일치하면 관리자 권한을 부여하는지 확인한다.		P
16-3	관리자 로그인 기능 수행 Test	시스템이 관리자의 아이디와 비밀번호를 체크하여 일치하지 않으면 일치하지 않다는 내용을 반환하는지 확인한다.		P
17-1	DVM 물품재고 관리 GUI 출력 Test	관리자가 상품재고 관리 기능을 활성화시킬 경우 활성화가 진행되는지 확인한다.	17. Manage Amount	P
17-2	DVM 물품재고 관리 기능 수행 Test	관리자가 상품의 재고를 변경할 경우 시스템이 해당 정보를 기반으로 상품 재고량을 수정하는지 확인한다.		P
18-1	DVM 현금 관리 GUI 출력 Test	관리자가 현금 관리 기능을 활성화시킬 경우 활성화가 진행되는지 확인한다.	18. Manage Amount Cash	P
18-2	DVM 현금 관리 기능 수행 Test	관리자가 현금을 변경할 경우 시스템이 해당 정보를 기반으로 현금 재고량을 수정하는지 확인한다.		P

Activity 2066. Testing Traceability Analysis




reduceAmount()
13-1.현재 DVM에 재고 없는 물품에 대한 다른 DVM 재고 안내 기능 수행 Test
14-1. 현재 DVM에 재고 없는 물품에 대한 다른 DVM 재고 안내 기능 수행 Test
14-2. 현재 DVM에 재고 없는 물품에 대한 다른 DVM 재고 안내 기능 수행 Test
14-3. 현재 DVM에 재고 없는 물품에 대한 다른 DVM 재고 안내 기능 수행 Test
15-1.현재 DVM에 재고 없는 물품에 대한 다른 DVM 재고 안내 기능 수행 Test
15-2.현재 DVM에 재고 없는 물품에 대한 다른 DVM 재고 안내 기능 수행 Test
15-3.현재 DVM에 재고 없는 물품에 대한 다른 DVM 재고 안내 기능 수행 Test
15-4.현재 DVM에 재고 없는 물품에 대한 다른 DVM 재고 안내 GUI 출력 Test
16-1.관리자 로그인 GUI 출력 Test
16-2.관리자 로그인 기능 수행 Test
16-3.관리자 로그인 기능 수행 Test
17-1. DVM 물품재고 관리 GUI 출력 Test
17-2.DVM 물품 재고 관리 기능 수행 Test
18-1.DVM 현금 관리 GUI 출력 Test
18-2.DVM 현금 관리 기능 수행 Test

Notes. Stubs in Implementation

- 본 프로그램에서 Stub에 해당하는 요소에는 Other DVMs, Valid Code List, Valid Card List가 있음
- Stub들은 실제 네트워크 환경이나 데이터베이스를 구축한 것이 아닌 편의상 해당되는 클래스 내에서 클래스 변수로 선언하여 각각의 DVM Object들이 해당되는 Stub들에 접근할 수 있도록 하였음

```
public class DVM {
    static HashMap<String,String> codeTable = new HashMap<>();
    static ArrayList<DVM> DVMList  = new ArrayList<>();
```

```
public class Payment {
    static HashMap<String, Integer> validCardList = new HashMap<>() {{...}};
```

Notes. User Manual

- User Manual의 자세한 사항은
  과목 홈페이지 내 업로드된 자료를 참조

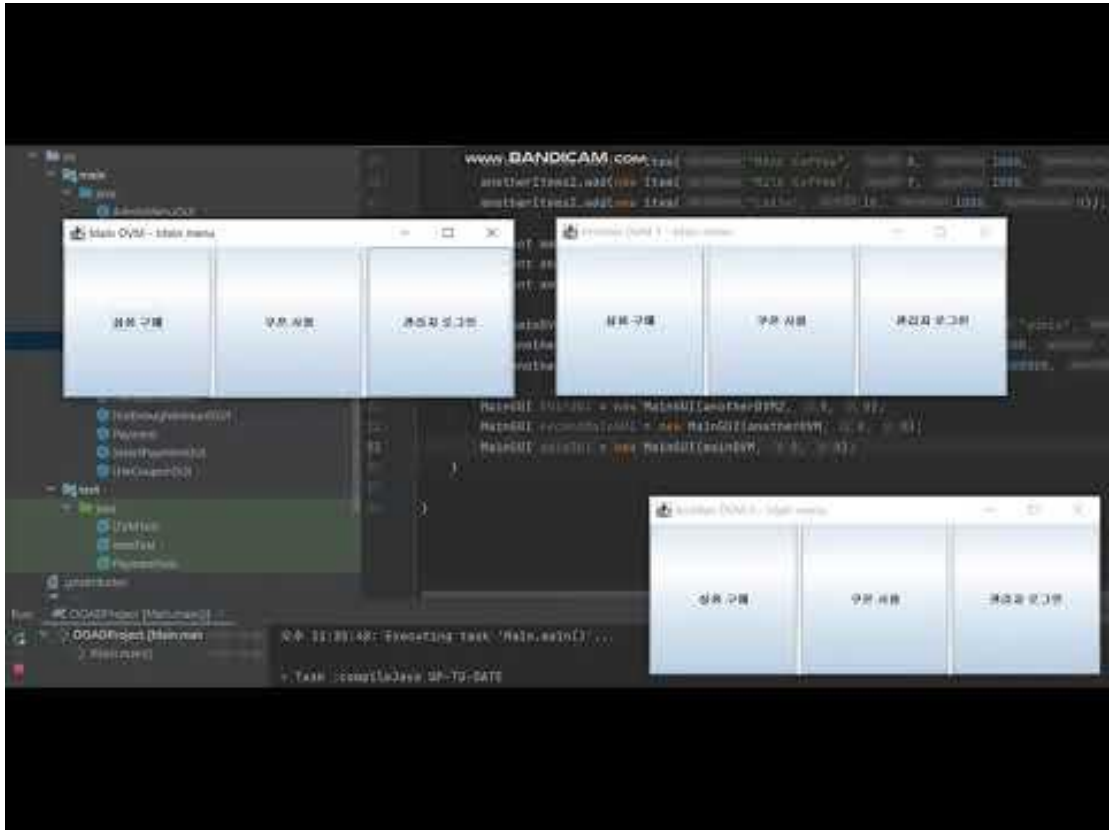
본 DVM 프로그램은 크게 3가지 기능으로 구성되어 있습니다. 각 기능은 마우스로 해당되는 버튼을 클릭하거나 입력란에 적절한 값을 입력한 뒤 적용 버튼을 누르는 식으로 사용할 수 있습니다.

본 프로그램을 실행했을 때 화면에 출력되는 최초 화면은 다음과 같습니다.

1. 상품 구매
   1.1. 상품 리스트 확인

- 최초 화면에서 '상품 구매' 버튼을 누르면 DVM에 등록되어 있는 제품들을 선택할 수 있는 버튼들과 해당 제품의 가격, 재고 내용이 차례대로 출력됩니다.
- 취소 버튼을 누르면 최초 화면으로 되돌아갑니다.

Demonstration Video



Thank You.