

Object-Orientation Programming

Team 14 Report



TEAM MEMBER

20200479 김동영

20200699 박혜인

20200479 정수용

Contents

- 1. Introduce and Overview Our Program**
- 2. Explain Compile and Execution**
- 3. Descript Functionally and How We Implemented Our Program**
- 4. UML Modeling**
- 5. Execution Result**
- 6. How We Applied OOP Concepts**
- 7. Conclusion**

1. Introduce and Overview Our Program

Since the topic was free, all the team members suggested ideas and chose one as a topic. In the process of choosing a topic, it was not easy because we had to consider whether our topic has good use of the characteristics of OOP, whether it was a creative idea compared to other teams, and whether it was too easy or not.

Our topic is "Vending Machine" that we can see everywhere in our daily life.

- This vending machine supports two modes. One is a mode for consumers who purchase products, and the other is a mode for sellers who sell products, that is, those who manage vending machines.

- When running in buyer mode, it will work by inserting a coin first. After inserting a coin, if you select a product, the name, price, and inventory of the product will appear on the screen. After that, if you purchase a product, the amount will be deducted by the price of the product. Finally, if you no longer want to purchase, you can return the change.

- when running in manager mode, The manager can modify the name, price, inventory, etc. of the product. And you can also add or delete new products to sell.

2. Explain Compile and Execution

We created a program using the java language. And we compiled it using Eclipse. To execute the program, you can press Ctrl + F11 (Debug) or F11 button.

3. Descript Functionally and How We Implemented Our Program

First, we divided the program into inner parts and GUI parts. In the inner part, we created VendingMc class that implements the overall function of the vending machine, Goods class that describes products of machine, and Coin class that manages money. As example case, we also made Drink, Snack, and IceCream class that inherit Goods.

- VendingMc Class

This class is the core class of the inner part. It manages File I/O and goods.

```
private ArrayList<Goods> goodsList;  
private ArrayList<String> categoryList;  
  
private String goodsFilePath;  
private File file;  
private FileWriter fw;
```

goodsList is a list that stores the goods that currently exist in the vending machine, and *categoryList* is a list that stores the categories (type) of the goods. The following three variables are for File I/O.

```
VendingMc(String goodsFilePath) throws IOException{
    this.goodsFilePath = goodsFilePath;
    readFile();
}
```

VendingMc basically receives goods data file as an argument.

```
public void readFile() {
    file = new File(goodsFilePath);

    Scanner input = null;
    goodsList = new ArrayList<Goods>();

    try {
        input = new Scanner(file);
    }
    catch(Exception e) {
        System.out.println("파일 입력이 올바르지 않습니다.");
    }
    while(input.hasNext()) {
        String line = input.nextLine();

        if(line.isEmpty()) {
        }
        else {
            String[] info = line.split(";",-1);
            addGoods(info[0].trim(), Integer.parseInt(info[1]), Integer.parseInt(info[2]), info[3].trim(), info[4].trim());
        }
    }
    setArrangelist();
}
```

Then, in `readFile()`, save the file data to the two lists using the `File` class in java.

```
public void addGoods (String name, int price, int count, String className, String imagePath) {
    Goods temp = null;
    try {
        temp = (Goods) Class.forName("proj4." + className).getDeclaredConstructor(String.class, int.class,
            int.class, String.class, String.class).newInstance(name, price, count, className, imagePath);
    } catch (InstantiationException e) {
        e.printStackTrace();
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    } catch (InvocationTargetException e) {
        e.printStackTrace();
    } catch (NoSuchMethodException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
    goodsList.add(temp);
}
```

In `addGoods()`, create a new `Goods` instance with the data of the goods and store it in *goodsList*. At this time, we created an instance suitable for the category class through dynamic binding. For this, we used `Class.forName`, which led to longer try-catch statements.

This method is used to read files and create an initial list, or to add

goods at runtime.

```
public void deleteGoods(int index) {  
    goodsList.remove(index);  
}
```

We also made it possible to delete certain goods from the list through deleteGoods().

```
public void setArrangeList() {  
    categoryList = new ArrayList<String>();  
    Goods g;  
  
    for(int i=0;i<numGoods();i++) {  
        g = goodsList.get(i);  
        if(!categoryList.contains(g.getCategory())) categoryList.add(g.getCategory());  
    }  
}
```

In setArrangeList(), update the *categoryList* from the *goodsList*. This function is used in the process of importing files, and is used whenever there is a change in goods data at runtime.

```
public void saveVMInFile() throws IOException {  
    fw = new FileWriter(file);  
  
    for(int i=0;i<goodsList.size();i++) {  
        fw.write(goodsList.get(i).getGoodsInfo() + "\n");  
    }  
    fw.close();  
}
```

This function is used to store data from the *goodsList* to a file. It is also used whenever there is a change in goods information at runtime.

- Goods Class

This is the base class for vending machine products(goods).

```
private String name;  
private int price;  
private int count;  
private String imagePath;  
private String category;
```

Goods class basically manages product information. It has Name, price, inventory number, image file path, category as variables.

```
public String getName() {return name;}  
public int getPrice() {return price;}  
public int getCount() {return count;}  
public String getCategory() {return category;}  
public String getPath() {return imagePath;}  
public String getType() {return category;}  
  
public void setCount(int i) {count = i;}  
public void setData(String name, int price, int count, String path) {  
    this.name = name;  
    this.price = price;  
    this.count = count;  
    this.imagePath = path;  
}
```

This class also has getters and setters for data management.

Through `description()`, we can get explanations to appear in the info window of the vending machine. We can also customize it and use it in subClass.

```
public boolean buyGoods() {  
    if(Coin.getSum() >= price && count > 0) {  
        Coin.subCoin(price);  
        count--;  
        return true;  
    }  
    else {  
        return false;  
    }  
}
```

In this function, we can customize what the machine will do when the user buy the product. In this function, we can customize what the

machine will do when the user buys the product. Because the Goods class is a base class, it only does basic actions. But we can add various actions needed for the category in subclass. (by overriding). For example, if goods is pizza, you can ask to make pizza with the ingredients given in this function.

- Coin Class

This is a class that manages coins put into vending machines. The variables and methods were declared static and used as class members.

```
private static int sum;  
public static int getSum() {return sum;}
```

Coin class has a variable *sum*, the total amount of coin put into the vending machine. There is also a getter for this.

```
public static int add100() {  
    sum += 100;  
    return sum;  
}  
public static int add500() {  
    sum += 500;  
    return sum;  
}  
public static int add1000() {  
    sum += 1000;  
    return sum;  
}  
public static int add5000() {  
    sum += 5000;  
    return sum;  
}  
public static int add10000() {  
    sum += 10000;  
    return sum;  
}  
public static int add50000() {  
    sum += 50000;  
    return sum;  
}  
  
public static void clearCoin() {  
    sum = 0;  
}  
public static void subCoin(int amount) {  
    sum -= amount;  
}
```

We can add coins through the add function. Our vending machine can only accept coins and bills, so we made add functions according

to the unit of money. Through subCoin(int amount), we can deduct some coins, and through clearCoin(), we can deduct all coins. SubCoin is used when the user buys the product, and clearCoin is used when the purchase is completed.

- Drink / Snack / IceCream Class

These are the subclasses of Goods that we made as examples of Goods. In these three classes, only description() was overridden, but can customize subclasses by adding various variables and functions that suit the category of product.

```
public class Drink extends Goods{
    Drink(String name, int price, int count, String category, String imagePath){
        super(name, price, count, category, imagePath);
    }

    public String description() {
        String str;
        str = "제품명 : " + this.getName() +
            "\n가격 : " + this.getPrice() +
            "\n개수 : " + this.getCount() +
            "\n\n음료 제품";

        return str;
    }
}

public class Snack extends Goods{
    Snack(String name, int price, int count, String category, String imagePath){
        super(name, price, count, category, imagePath);
    }

    public String description() {
        String str;
        str = "제품명 : " + this.getName() +
            "\n가격 : " + this.getPrice() +
            "\n개수 : " + this.getCount() +
            "\n\n과자 제품";

        return str;
    }
}

public class IceCream extends Goods{
    IceCream(String name, int price, int count, String category, String imagePath){
        super(name, price, count, category, imagePath);
    }

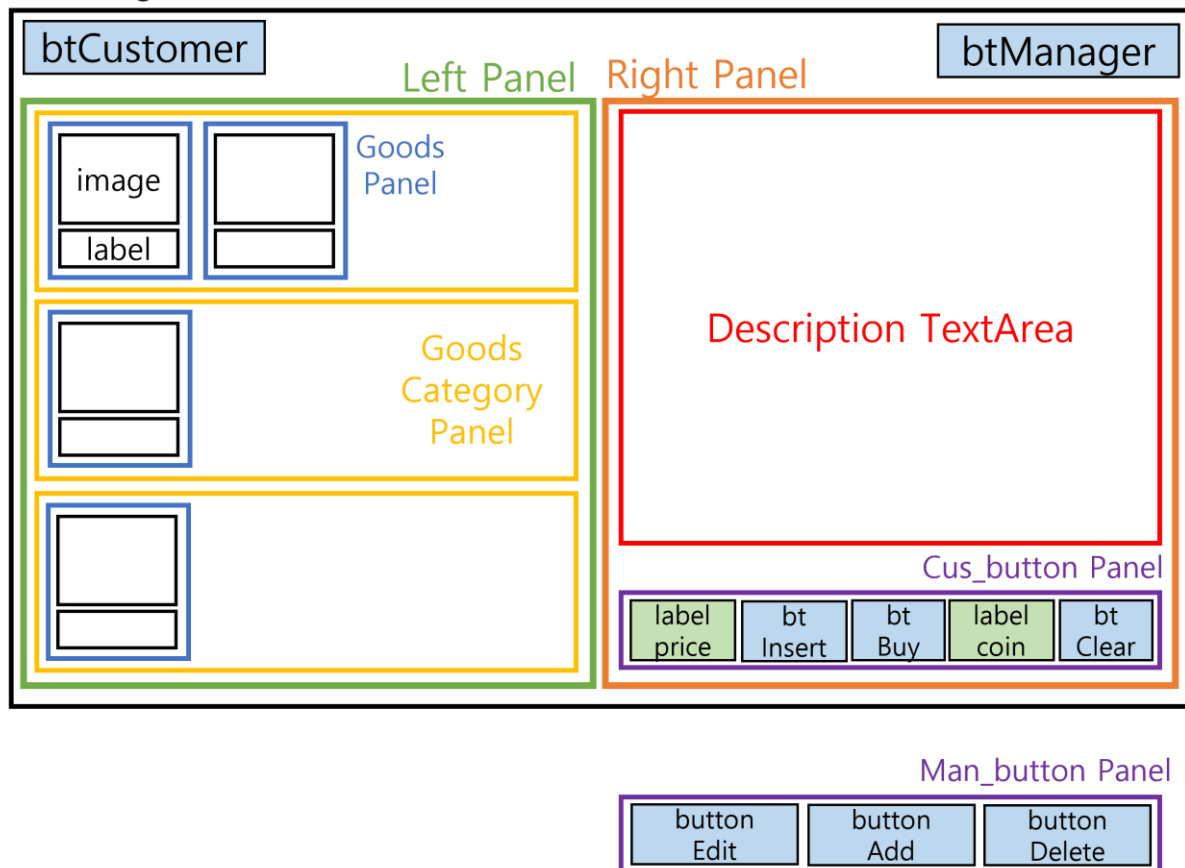
    public String description() {
        String str;
        str = "제품명 : " + this.getName() +
            "\n가격 : " + this.getPrice() +
            "\n개수 : " + this.getCount() +
            "\n\n아이스크림 제품";

        return str;
    }
}
```

- VendingMCFrame Class

This is the main GUI class of the vending machine. Each panel, button, label, and textArea are included as objects.

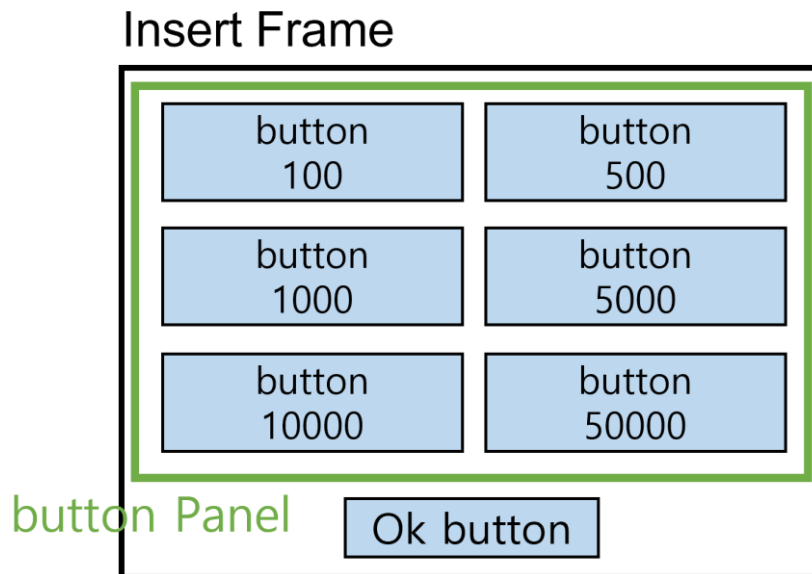
VendingMc Frame



When the customer or manager button is pressed, the button panel below is changed to the panel of the corresponding mode. Method such as insert and buy all work by receiving goods as argument (for polymorphism). If the user presses the insert coin buttons in customer mode, new *InsertFrame* object is created, and if edit, add, and delete buttons in manager mode, new *GoodsInfoFrame* object is created.

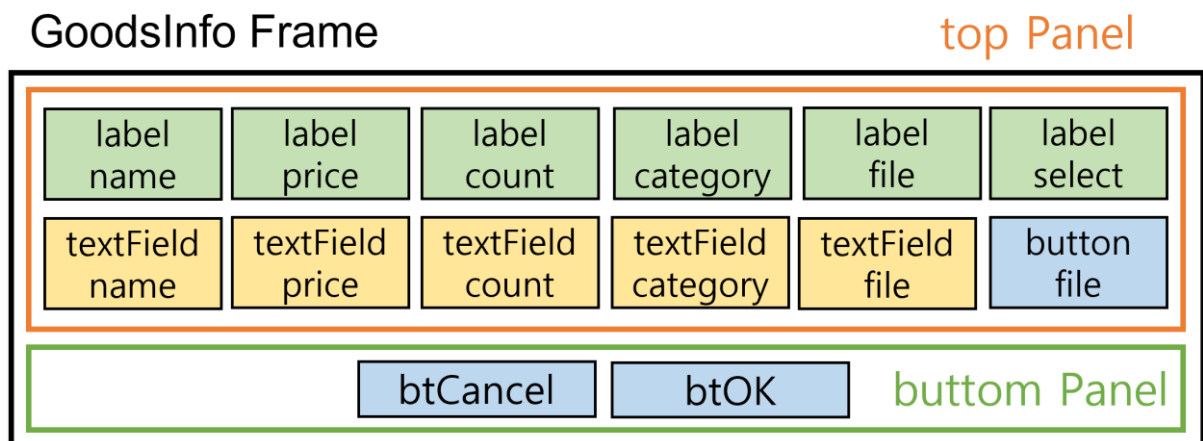
We made GUI just to make it easier to understand our program, so we won't explain the code.

- InsertFrame Class



When the user presses the coin buttons, coin is inserted by executing the add method of the coin class.

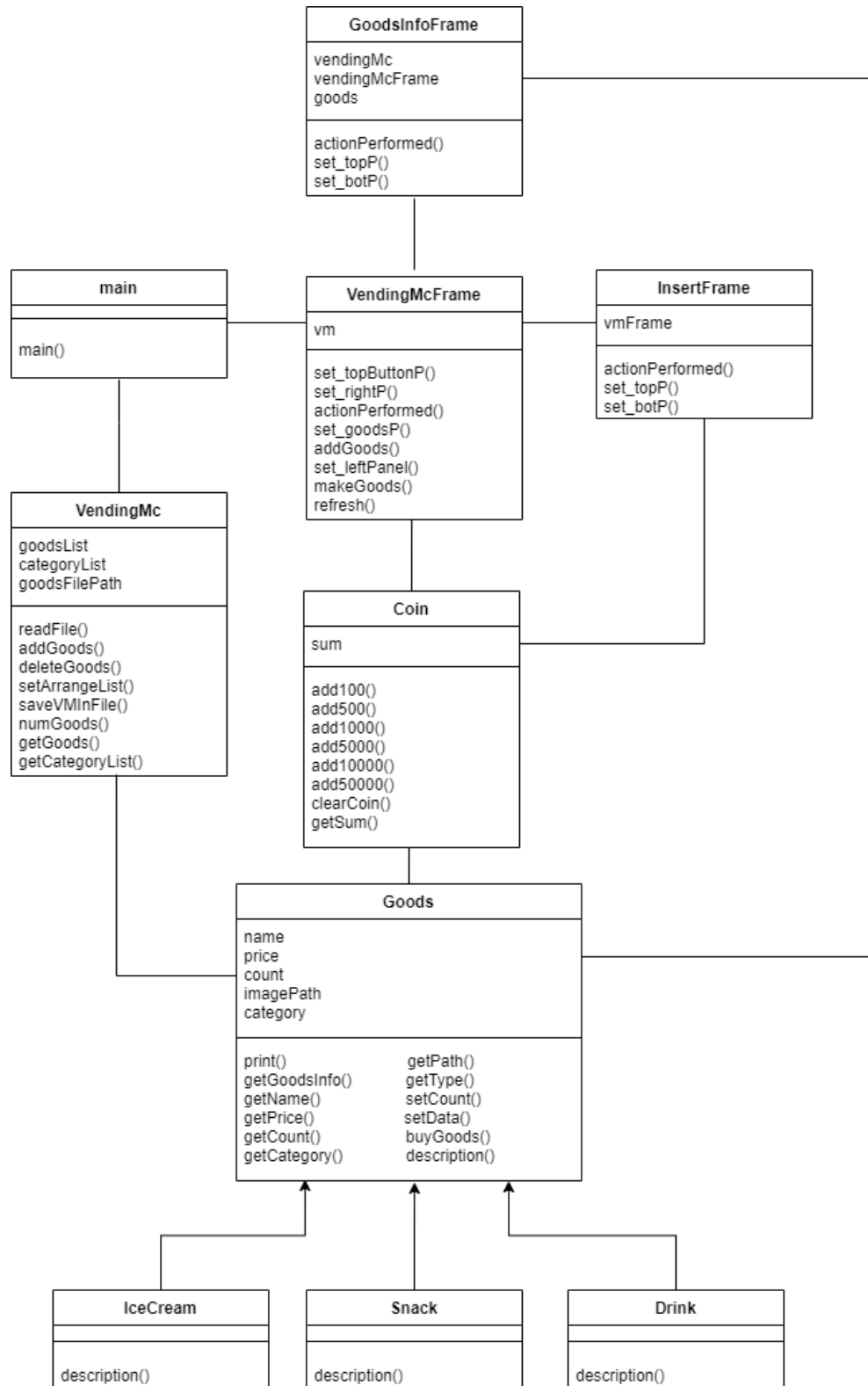
- GoodsInfoFrame Class



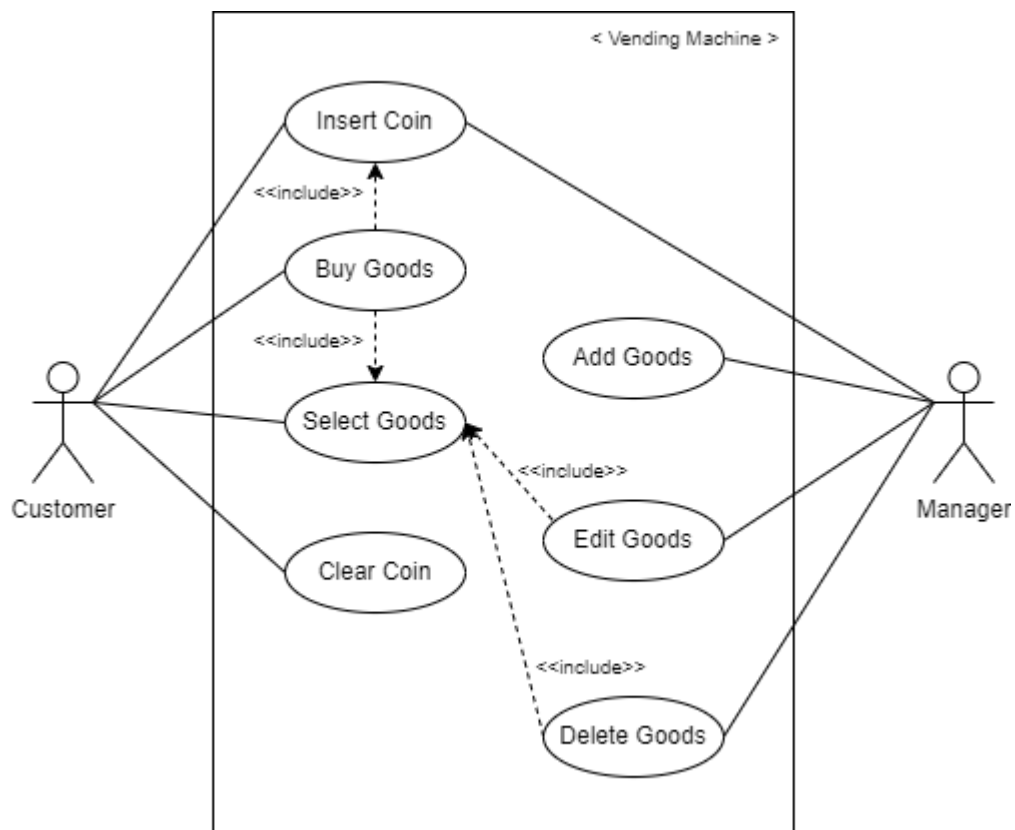
When the user presses the ok button, the data of *Goods* object is updated or added to the value entered in the text field.

4. UML Modeling

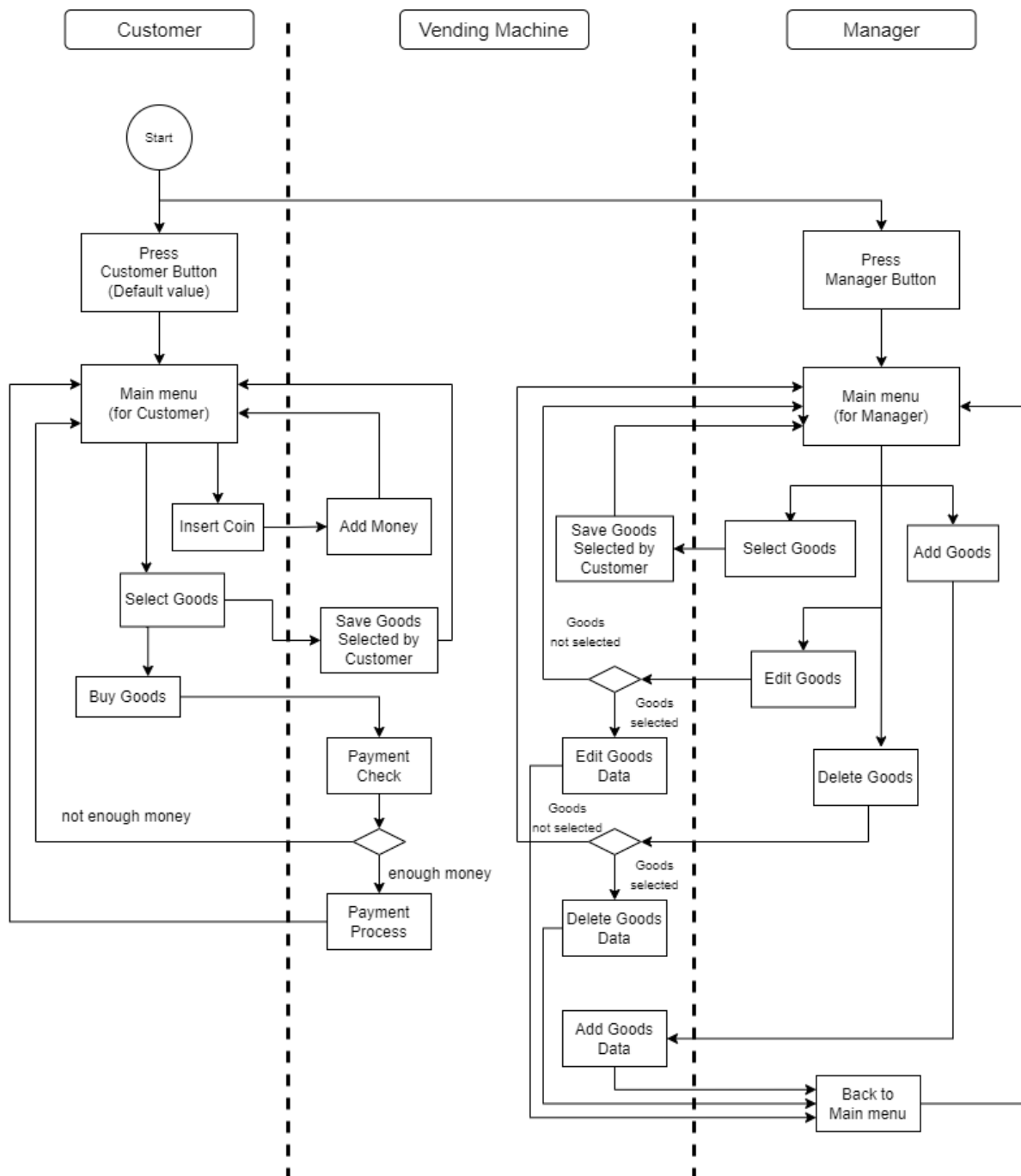
<Class Diagram>



<Use Case Diagram>

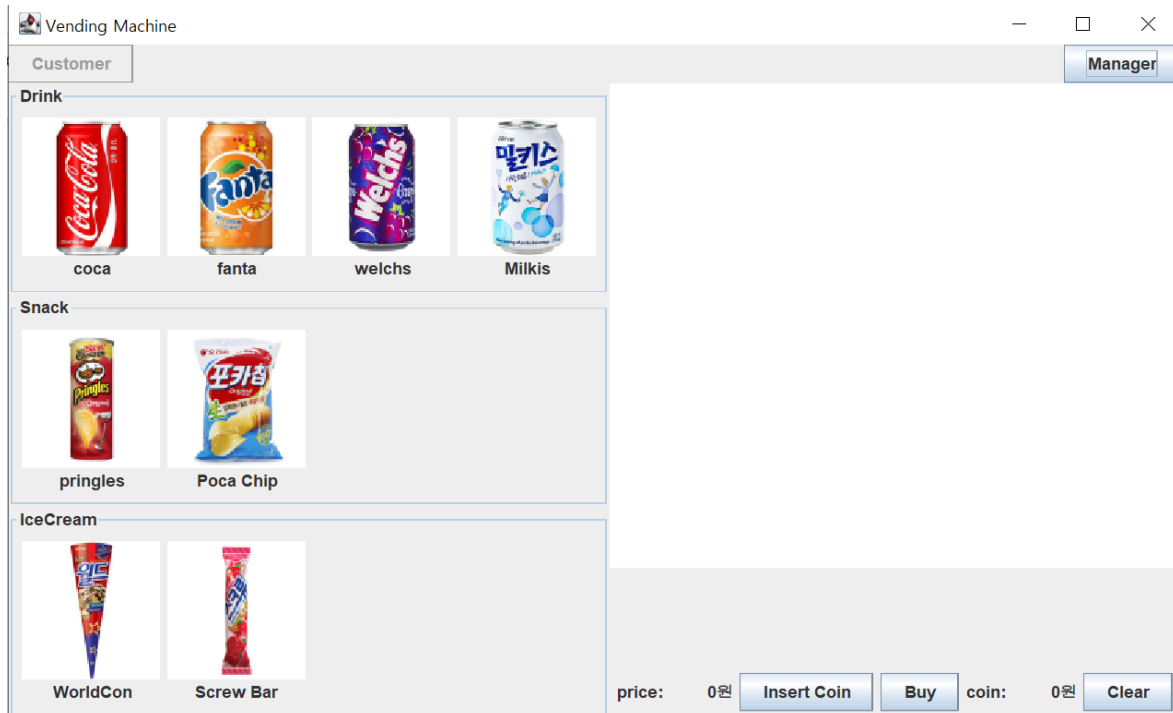


<Activity Diagram>

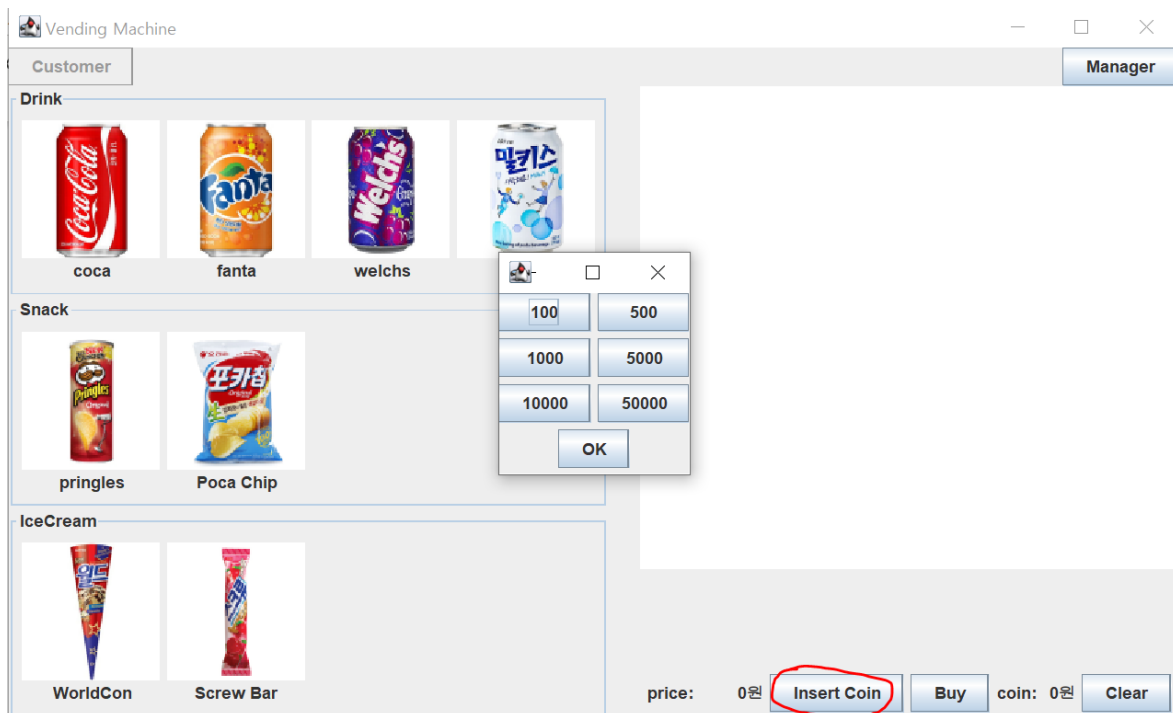


5. Execution Result

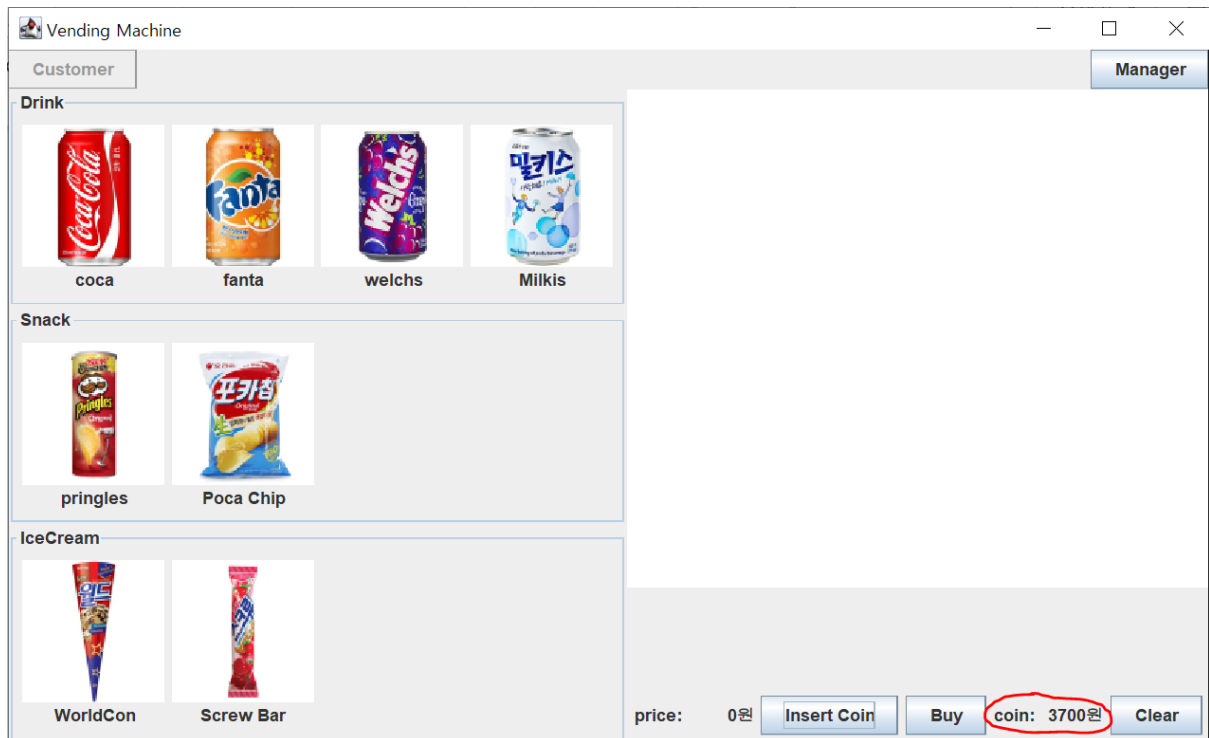
- Customer Mode



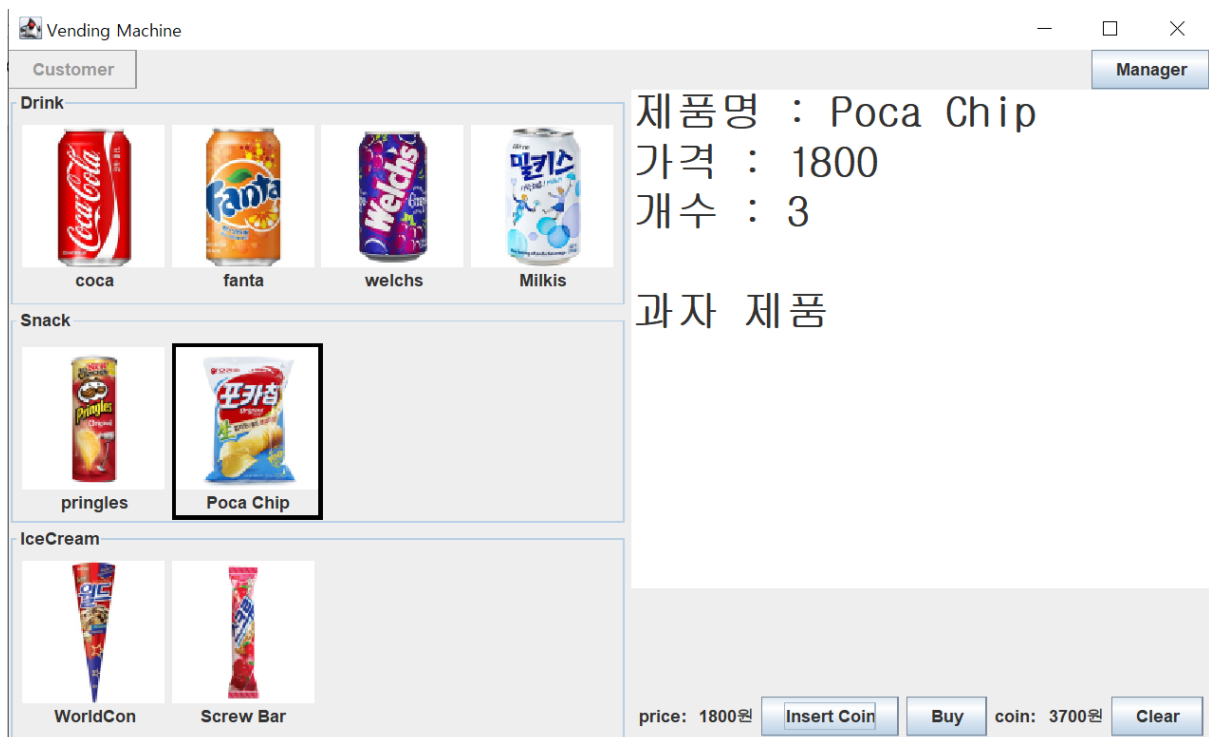
When you run the program, this screen appears.



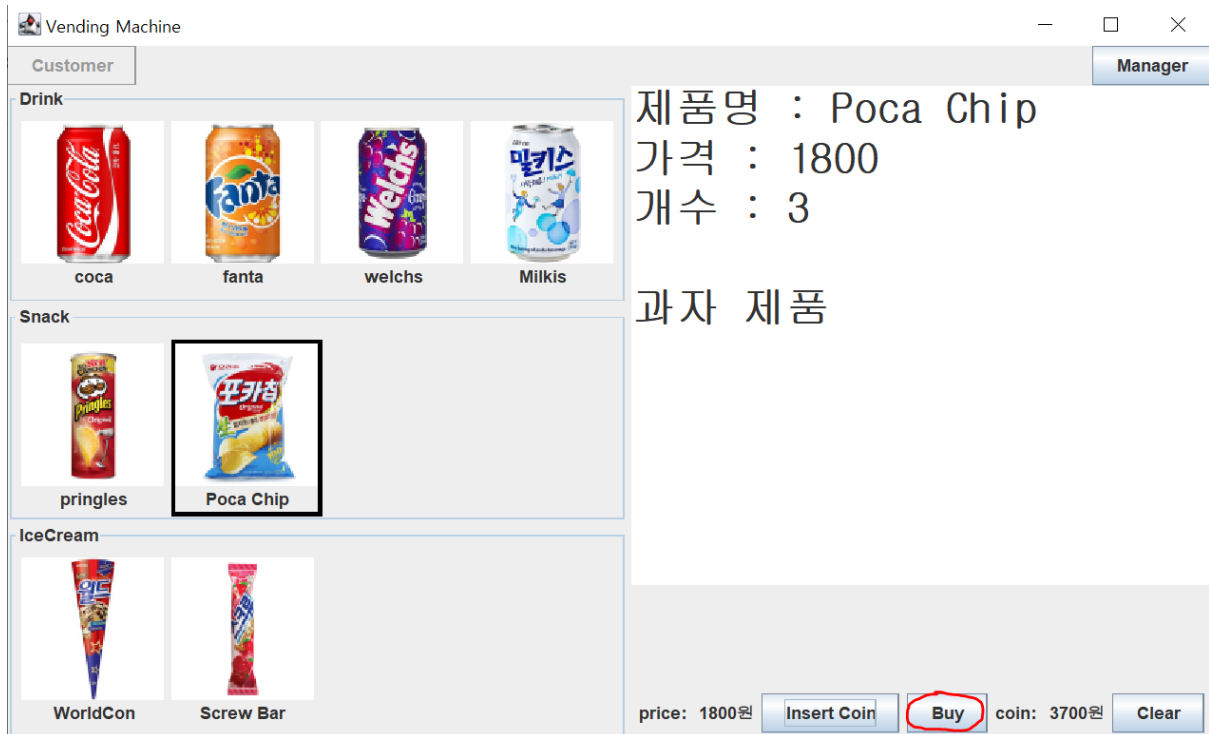
And you can insert coin through pressing "Insert coin" buttons



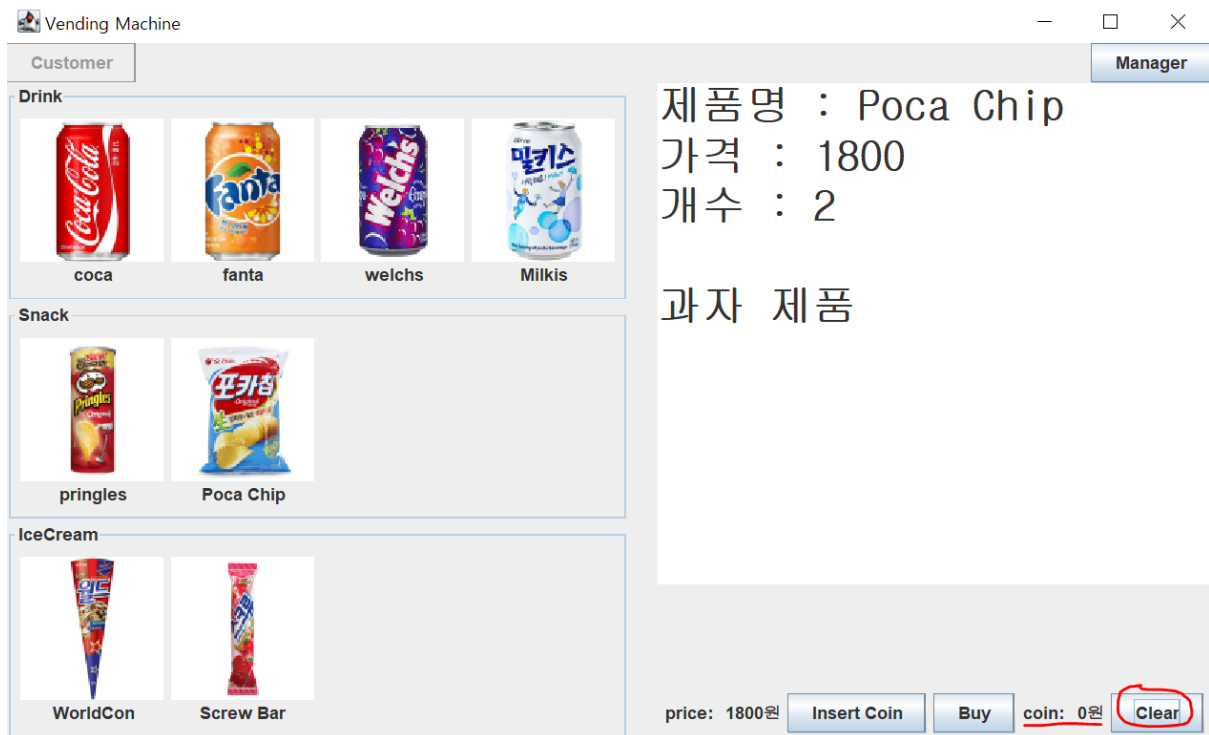
You can see the amount of coin is increased.



And if you choose the good you want, you can see some information on the screen.

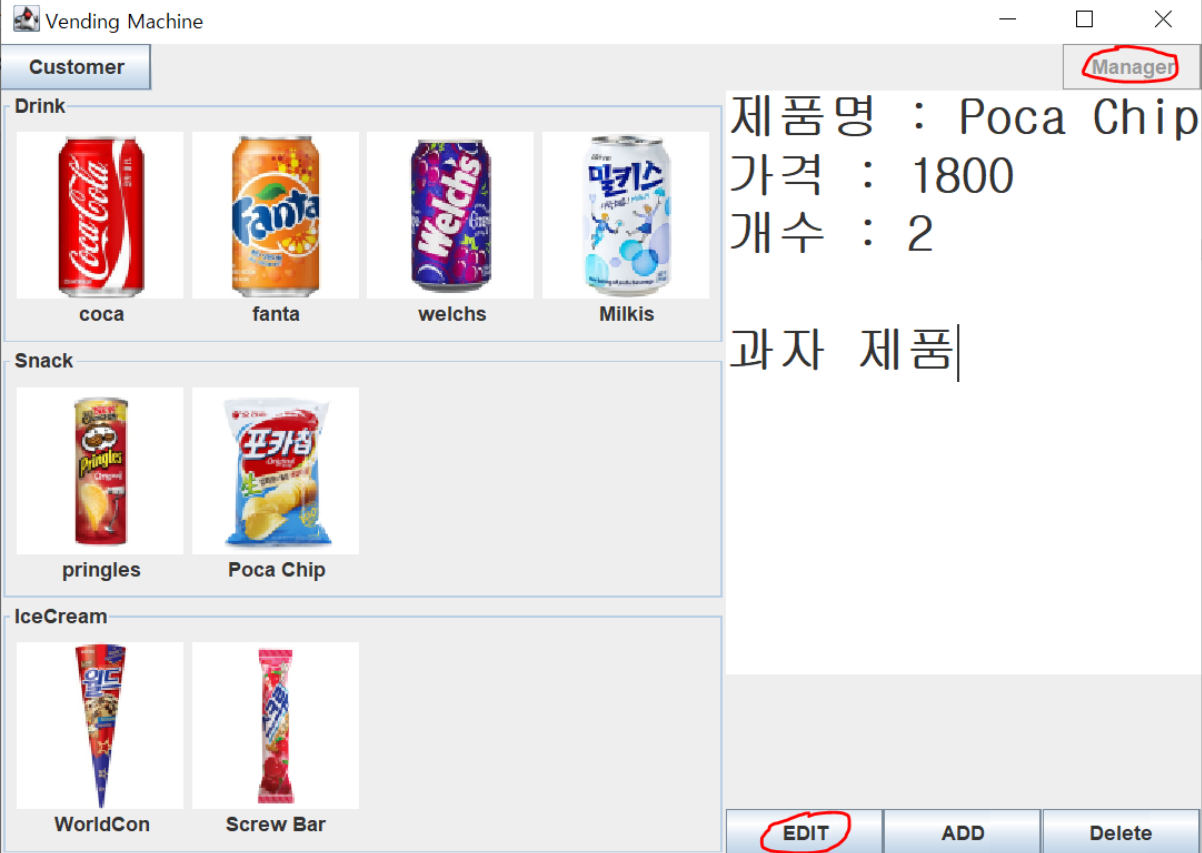


And if you buy "Poca Chip", you can notice your coin is decreased by the price of the product.



If you want to stop buying, you should press the "clear" button to return the remaining coin.

- Manager Mode

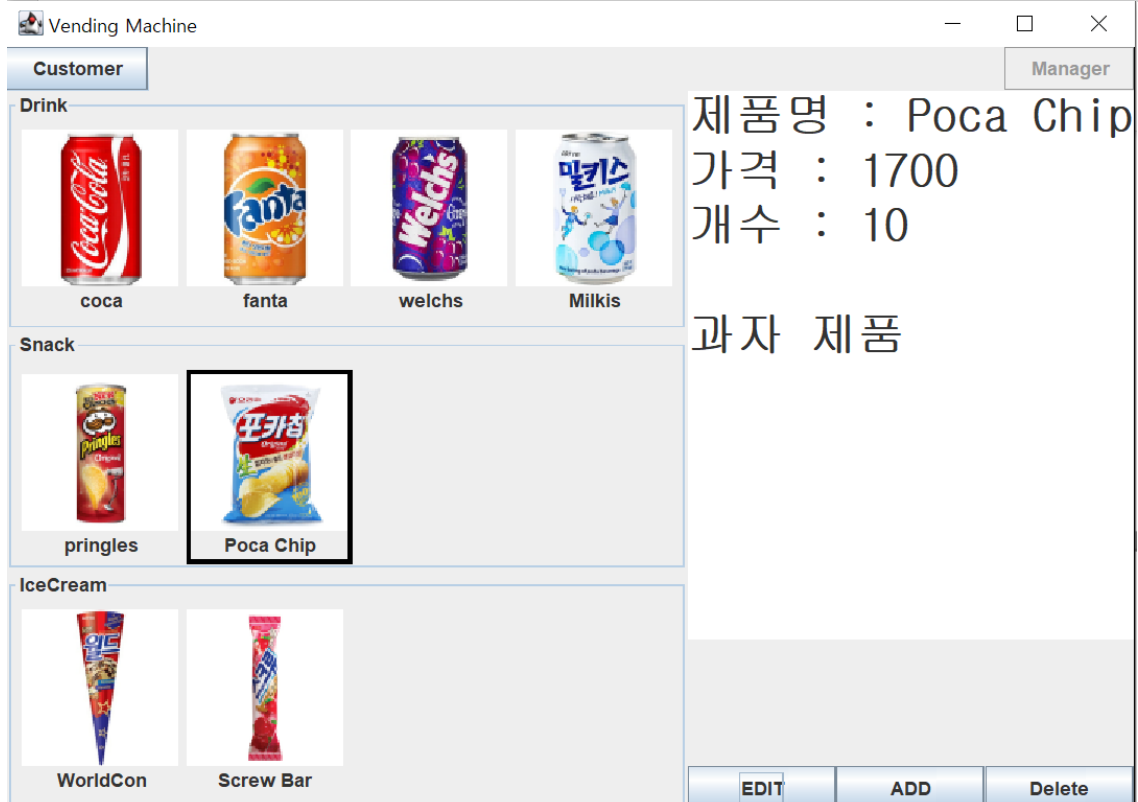
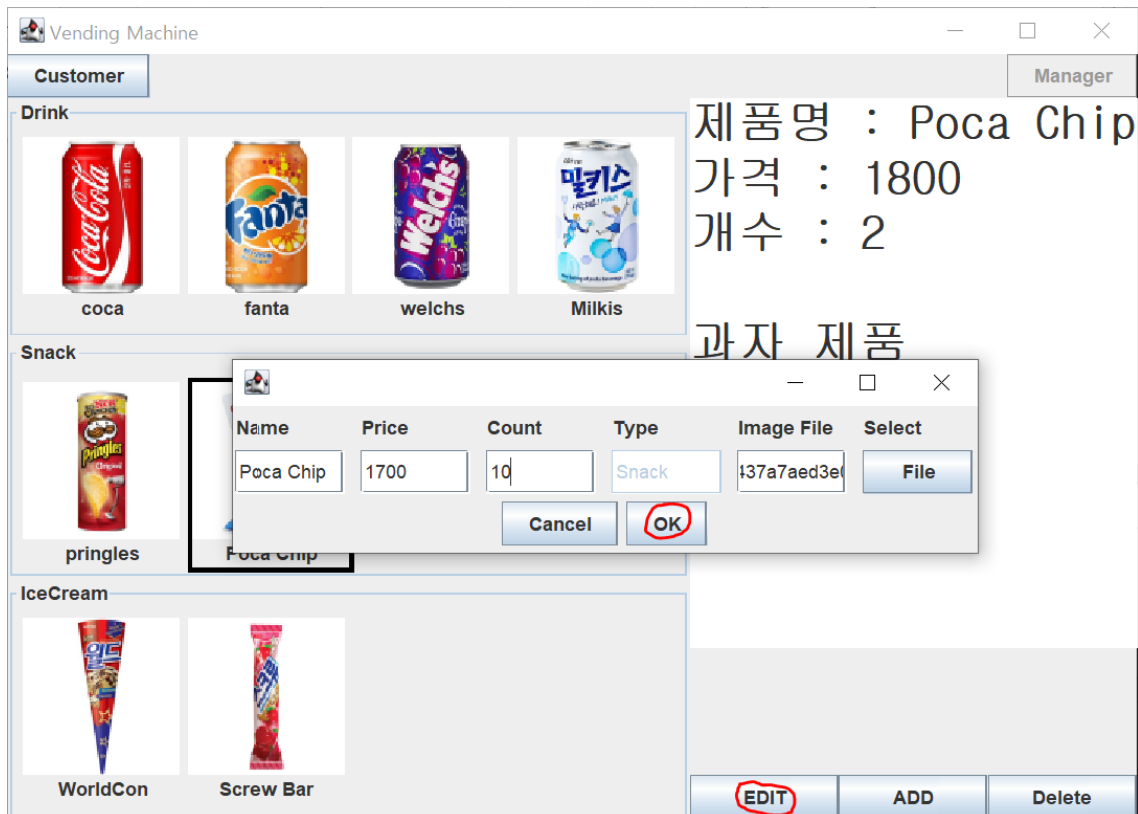


The image shows a software interface for a vending machine. At the top, there's a title bar 'Vending Machine' with standard window controls. Below it, a tabbed interface has 'Customer' and 'Manager' tabs. The 'Manager' tab is active and circled in red. The main area is divided into three sections: 'Drink' with four items (Coca-Cola, Fanta, Welch's, Milkis), 'Snack' with two items (Pringles, Poca Chip), and 'IceCream' with two items (WorldCon, Screw Bar). Each item has a small image and a label below it. On the right side, there's a text area showing details for 'Poca Chip': '제품명 : Poca Chip', '가격 : 1800', and '개수 : 2'. Below this, the text '과자 제품' is visible. At the bottom right, there are three buttons: 'EDIT' (circled in red), 'ADD', and 'Delete'.

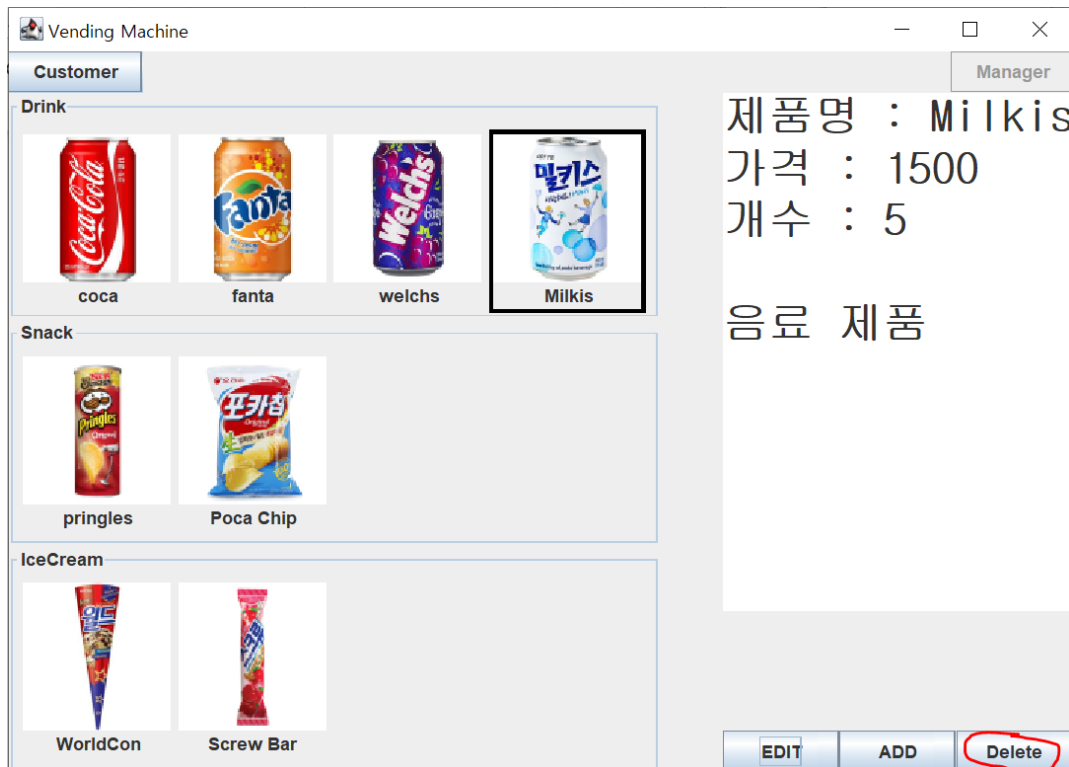
제품명 : Poca Chip
가격 : 1800
개수 : 2

과자 제품

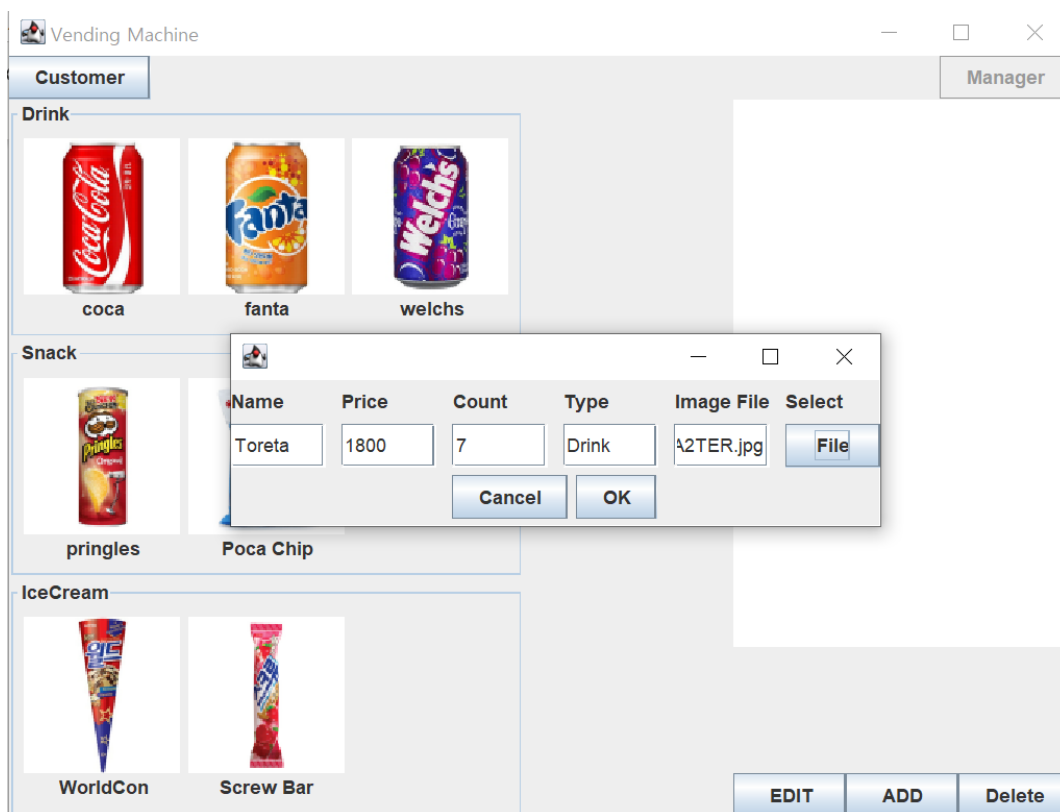
And you can also do manager mode to manage the goods.



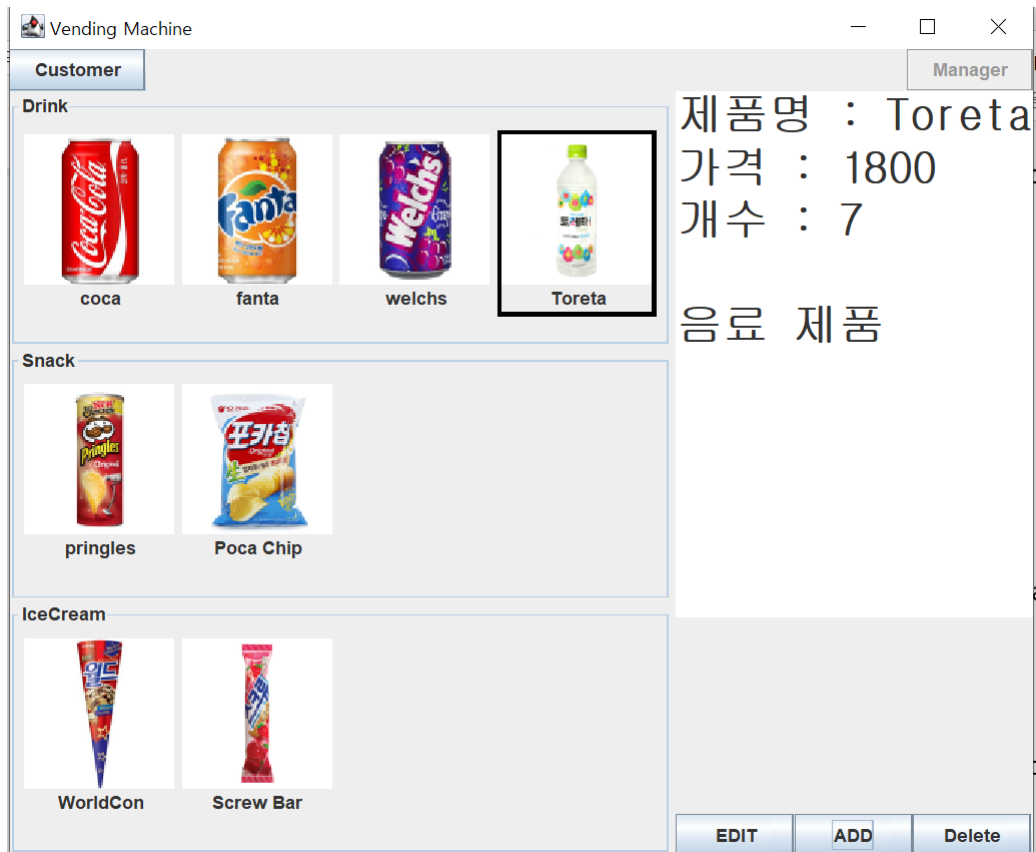
If you click "edit" button, you can edit the name, price, count, and image of goods. I edited the price and inventory of "Poca Chip", you can see the change of them.



And manager can delete the goods. I deleted "milkis".



On the contrary, manager can add the new products through "ADD" button. I added the "Toreta".



And then, "Toreta" is added again.

6. How We Applied OOP Concepts

Various OOP concepts were used because we used java, a language with OOP characteristics. But above all, polymorphism and dynamic binding were used as the most important concepts in our program.

So, we extended the Goods class to Drink, Ice Cream, and Snack class. So, it increases the reusability of the code. In addition, through the "ArrayList<Goods> goodsList" in the "VendingMc" class, we increase flexibility of program by allowing the binding occurring during execution time or change during execution.

7. Conclusion

Through this project, we were able to draw up our ideas in situations where no topic was given, and write the ideas as real programs. In this process, we were able to develop our skills by applying the concept of OOP to our program and solving some errors. In a way, we think this experience has become a valuable experience for all of our team members.