Project - Clothing

# Introduction

This is a C# project to represent a scenario that a person dresses for the temperature before leaving the house.

# General Description

Figure 1 shows all the cloth a person needs to wear. A person wears different cloth based on the different temperature. The blank square shows a person does not need to wear this type of cloth due to the temperature.

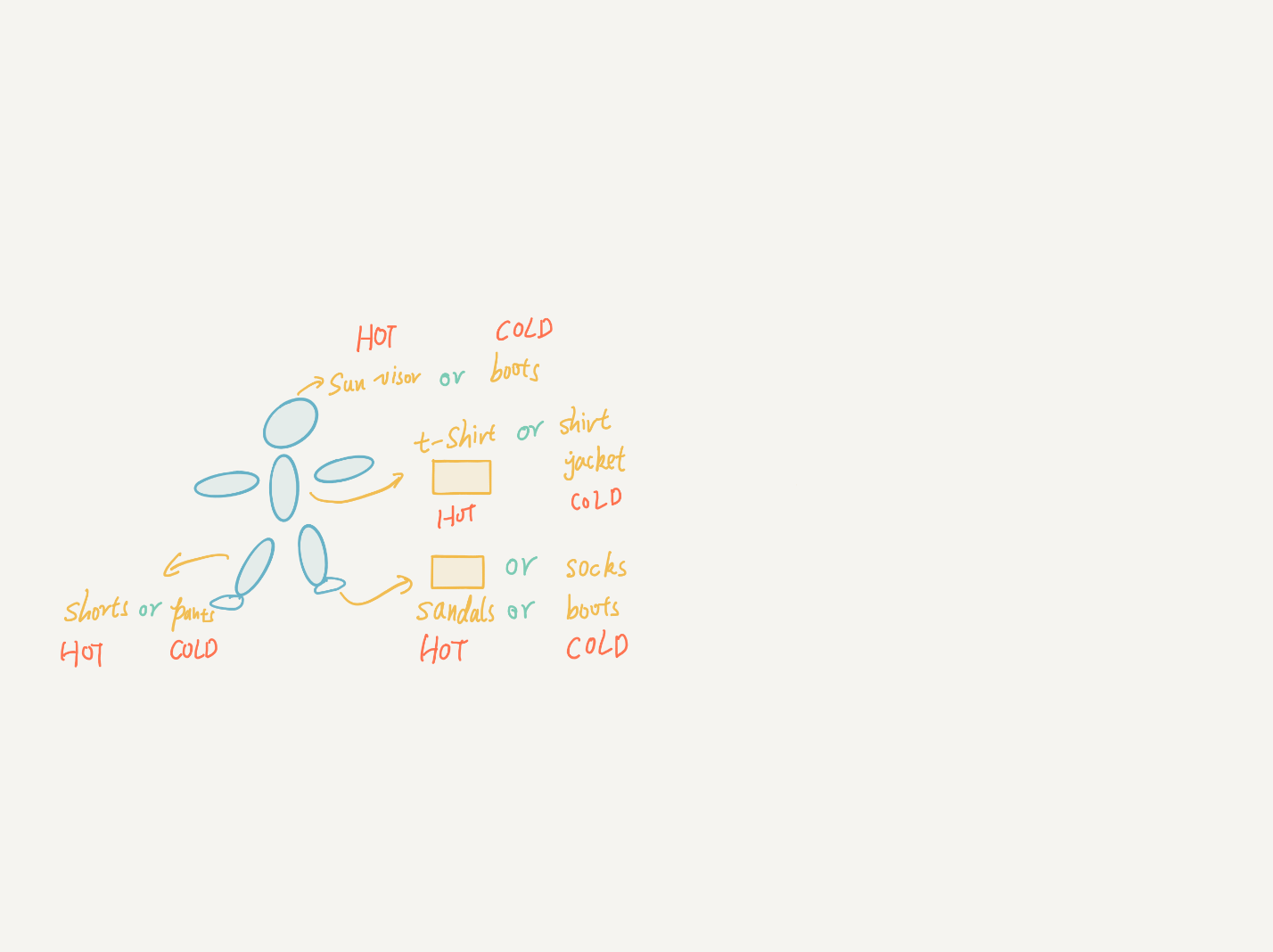


Figure . All the cloth needs to be worn.

Figure 2 shows the wearing sequence. The program will produce ‘fail’ when commands do not follow this sequence.

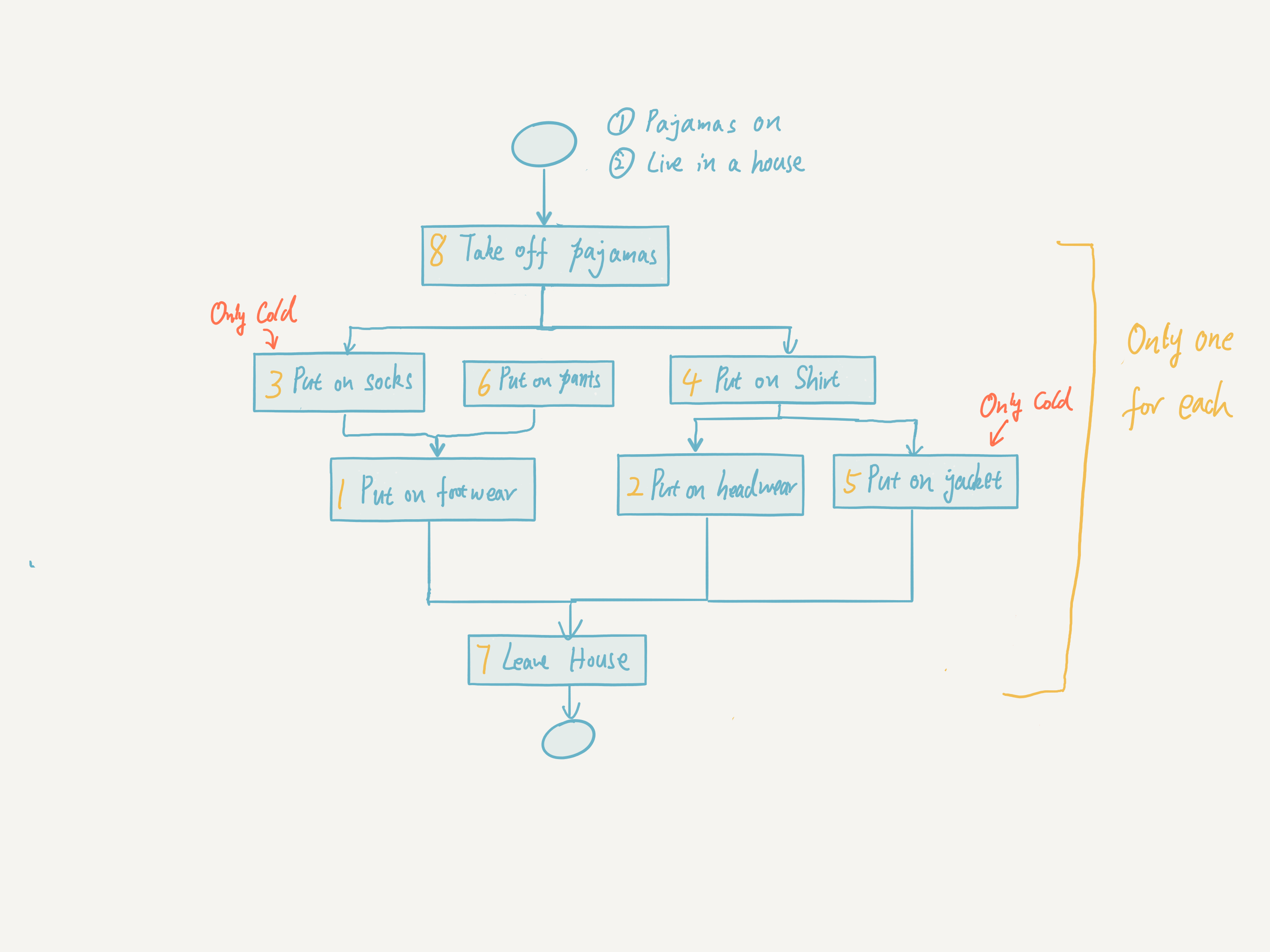


Figure . The activity diagram for both temperatures.

# Design and Implementation

## Do More tests

The best way to do more tests on your own is to add more unit tests under ‘ProgramTests’. Figure 3 shows the place ‘ProgramTests’ located.

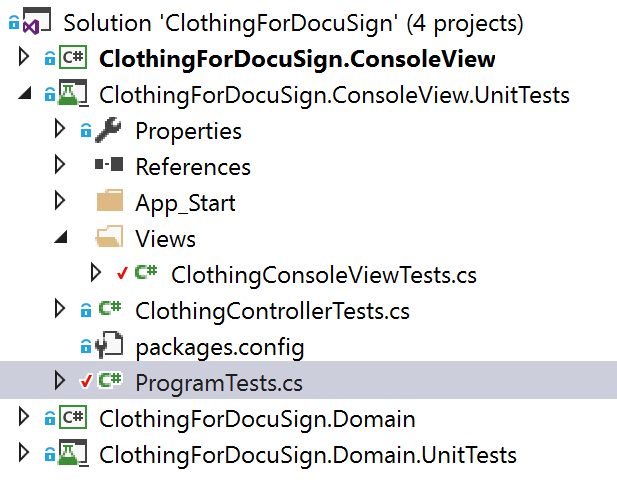


Figure . The location of ProgramTests

## Architecture

I used MVC pattern for the whole solution. ‘Domain’ project shows the model part and ‘ConsoleView’ includes controllers and views (See Figure 4).

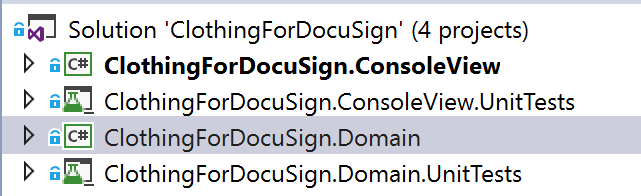


Figure . Projects

## Dependency Injection

I used Ninject as a dependency injection tool. DI can be better for unit tests and be easier to decouple classes. As you can see from Figure 5, I listed all the binding classes.



Figure . DI Configuration

## Unit tests

There are 57 unit tests and I covered 100% blocks for the model part which is under project ‘domain’ and 87.78% blocks for the controller and view part (See Figure 6). I did not test class ‘ConsoleInputAndOutput’ because there is only ‘Console.WriteLine’ and ‘Console.ReadLine’ in it.

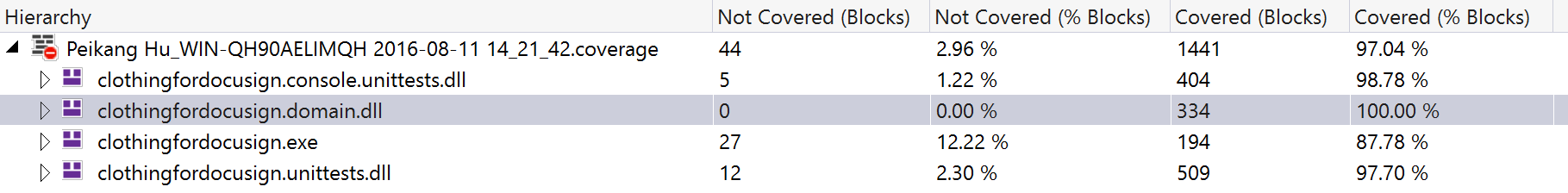


Figure . The Coverage of Unit Tests

I used Moq to mock objects, so each unit test only concentrates on the function itself. Figure 7 shows how I used Moq in the project.

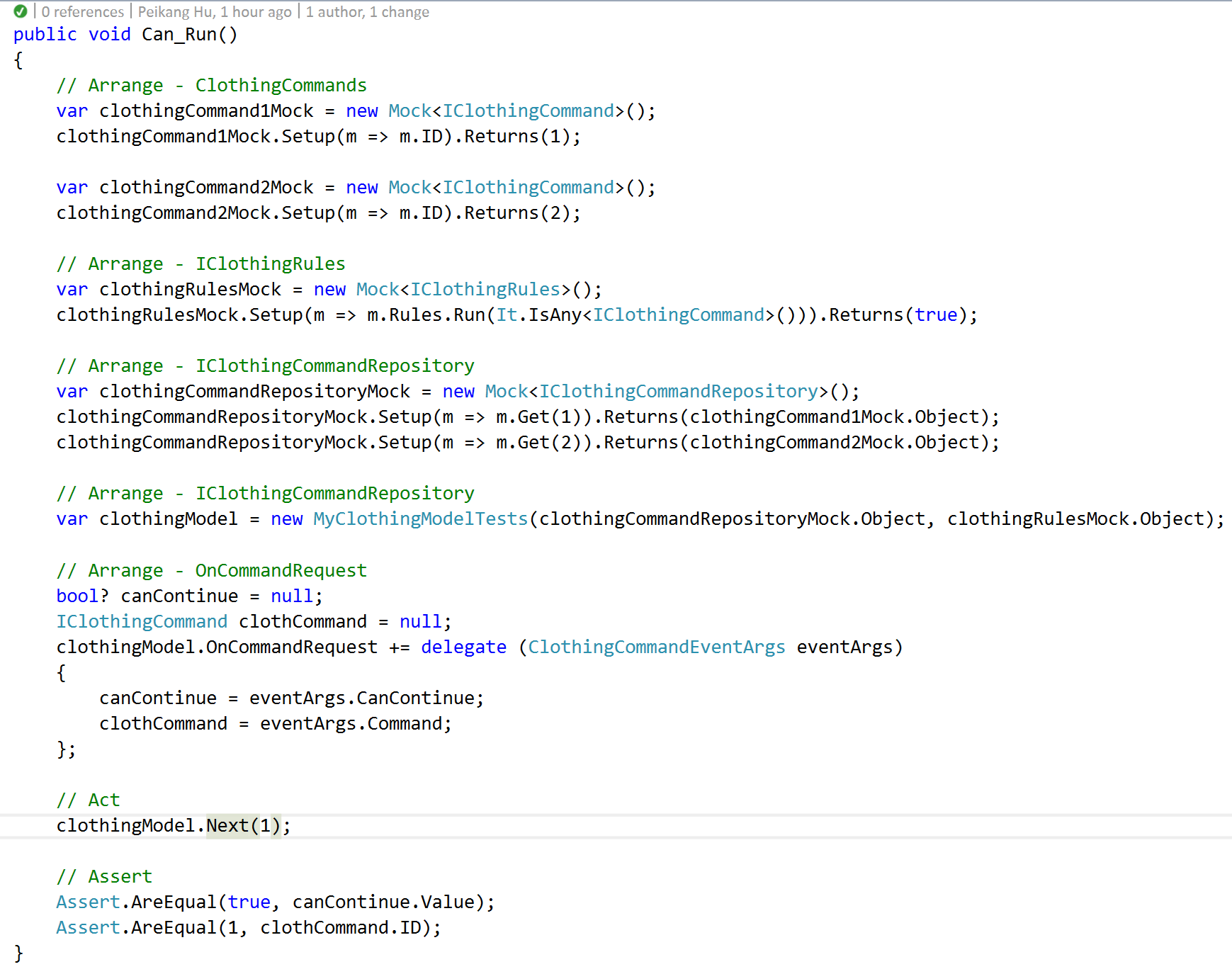


Figure . A Demo Unit Test

## How to follow the wearing sequences

I used the method which is similar to workflow. Firstly, I generated two graphs for HOT and COLD respectively. Following the tasks run, the in degree for each task will be changed. If the in degree is decreased to 0, which means this task can be run, otherwise the program will return fail. Please see more detail at Figure 8.

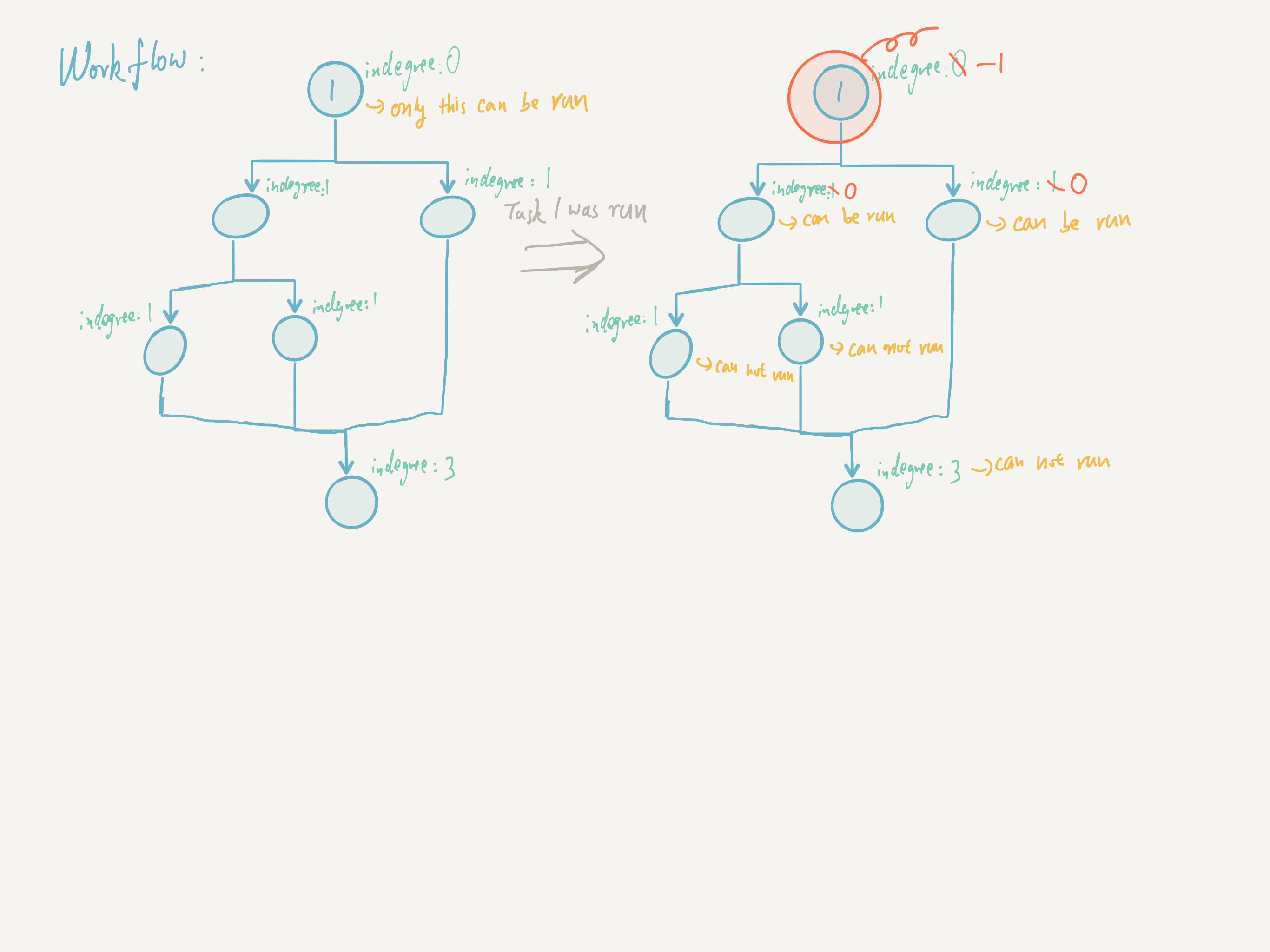


Figure . Workflow

# Extensibility

## MVC

The Model and the View were totally decoupled by MVC. So we can replace any parts to implement the new feature, even for the totally different business logic.

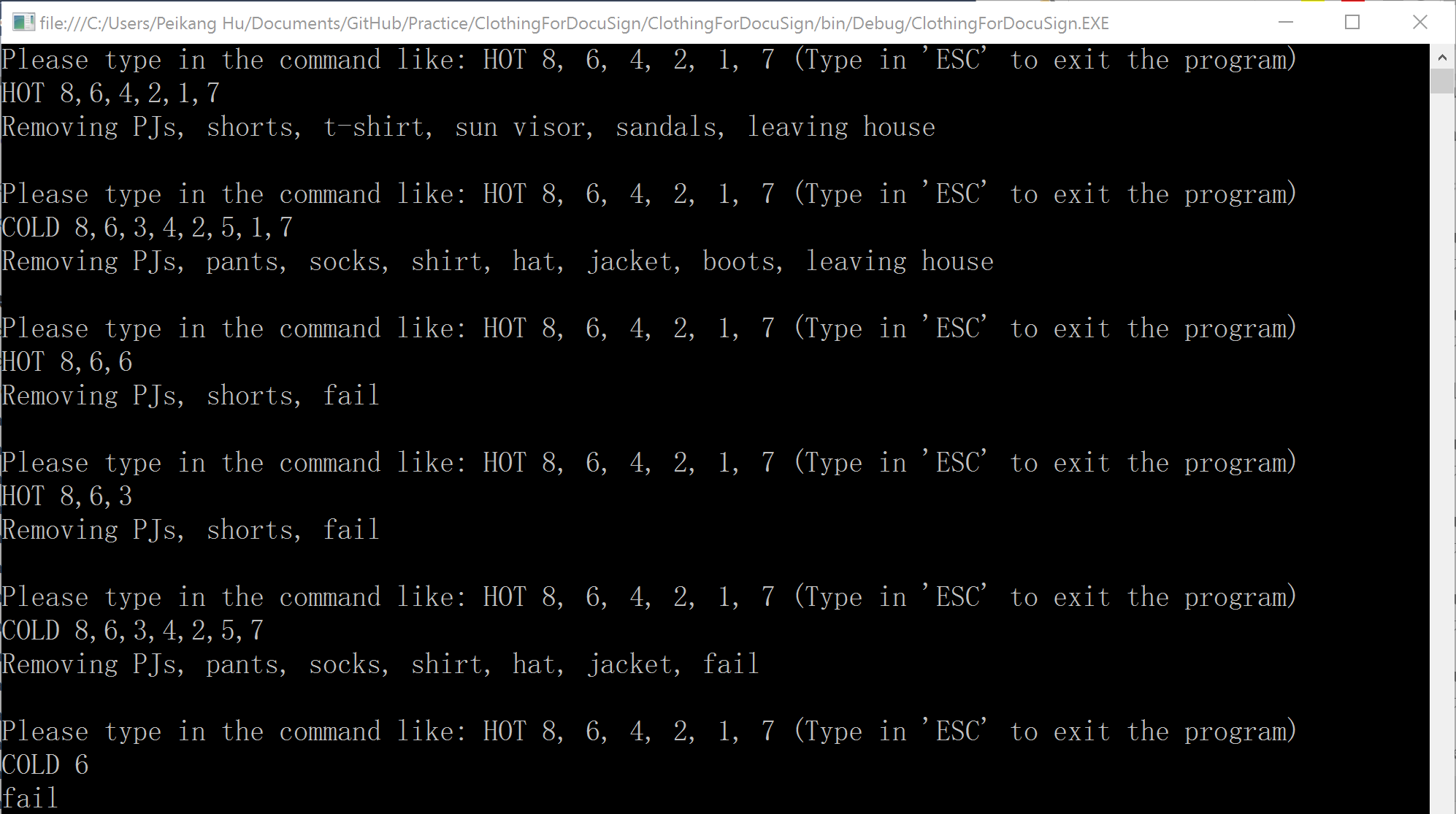
## Dependency Injection

The way to store, input and output data can be changed due to the use of dependency injection. There are ‘IInputAndOutput’, ‘IClothingCommandRepository’ and ‘IClothingRules’.

## Wearing Sequence

The wearing sequence can be easily changed. They are located in ‘ClothingColdRules’ and ‘ClothingHotRules’.

# Screen Shots



# Future Plan

Currently, the program only returns ‘fail’, when the program goes wrong. It can be improved to show different messages to inform users what is happened.