CSE1100 — Introduction to Programming
Assignment 5

September 20, 2024

Assignment

In this assignment we're going to pick up where we left off in the previous week and continue working on the application for the contractor. This week you will create a command line interface application to use the classess you created before. Next to that, you should now start using CheckStyle and JavaDoc to make sure your code of good quality.

Assignment 5.1

First, create a new class `JobCatalog`. This class will be used to manage the list of jobs the contractor has to do.
This class has 1 attribute: `List<Job> jobs`.

Define the following methods, and write tests for them:

> `public JobCatalog()`
> This *constructor* initializes `jobs` as a new concrete `List` type.
>
> `public String toString()`
> This method returns a human-friendly `String` representation of the list of jobs. You can choose the exact format yourself.
>
> `public boolean equals(Object other)`
> Post: If `other` is also a `JobCatalog` and has an equivalent list `jobs`, return `true`. Else return `false`.

Assignment 5.2

To start using the application you need to create a new class: `PlanningApplication`. This class will be used for the interaction with the user. You will create a main method that contains funtionality the user can choose from.

Define the following method:

> `public static void main(String[] args)`
> This *main method* represents the starting point of the program.

In you should offer the user the following options:

> `1 - Show all jobs in the catalog.`
> Print all the jobs and their information in a user friendly way.
>
> `2 - Add a new job.`
> Ask the user to enter all the information about a new job and add this job to this catalog.
>
> `3-6 - To be implemented.`
> These options will be implemented in a later assignment.
>
> `7 - Quit application.`
> The aplication terminates.

Create an empty catalog when the application starts. Work with a `Scanner` to allow the user to make a choice and implement the three features. If the user selects an option other than "Quit application" the menu should be printed again after the action has been completed, and the user should be able to make another choice after that.

Assignment 5.3

The contractor has one other thing they would like the application to take into account. Some of the equipment they have requires a special powersupply to work. For instance: the jackhammer requires an air compressor.

Because some of the equipment has a requirement like this, but other equipment does not, adding an attribute and additional methods to the ***`Equipment`*** *class is not ideal. Furthermore, creating an additional layer of inheritance between the class* ***`Equipment`*** *and the actual pieces of equipment for equipment that needs a powersupply will limit the posibility of creating categories of equipment in the future as more items are added to the application. To handle this issue elegantly you need to create an interface. Pieces of equipment that require a special powersupply can then implement this interface and its methods.*

Create an `Interface PowerSupply`.

Define the following method signature:

```
String getPowerSupply()
```

The class `Jackhammer` and the class `Torch` need to implement this interface. A jackhammer needs an *Air Compressor* and the torch needs *Butane Gas*. Make sure to edit the `toString()` method of these classes to also include information about the required powersupply.

Feedback

You can request help from a teaching assistant while performing the assignments. Also, once you feel your solution to the above assignments is complete, you are strongly encouraged to let a TA sign off your solution. Before you do so, please make sure you adhere to the rules stated in the lab manual (format your code properly, make sure the unit tests run successfully, etc.) Of course, feel free to ask a TA for advice if you need help with this.

Use the following checklist and discuss it with the TA when signing off:

√	Description
	Code formatting is according to the rules of the trade, there are no CheckStyle warnings or errors.
	Each method has comments in Javadoc style.
	Each class in this assignment has an `equals` & a `hashcode` method.
	Each method (apart from the main method) is tested with **100%** line coverage through jUnit tests.
	The number of asserts per unit test method is limited (preferably 1)