CSE1100 — Introduction to Programming
Assignment 4

September 20, 2024

Preliminaries

Before you start this assignment, create a new project in your workspace. All solution files for this assignment should be placed in this project.

Keep in mind that you have to write your own unit tests for the functionality you develop. These unit tests should be developed in a similar fashion to the unit tests that were provided for the first few assignments. E.g., aim for 1 assert per test, clear name of the test and description for assertions to explain what the expected behaviour is. **Every method should at least have 1 unit test, but aim for more!**

All methods, fields and the class itself should be annotated with JavaDoc documentation. JUnit tests are not required to have JavaDoc.

Assignment

During the next few assignments we will focus on implementing a small application for a local contractor. The contractor has different (small) construction jobs to do at different locations and wants to keep better track of when they need to go where and which tools they need to bring. Over the coming assignments we will slowly build a fully working application, but we will start this assignment with just a few classes.

Assignment 4.1

To start, the application is going to need several classes that can represent information related to the different jobs the contractor has. Before we start working on a class for Jobs, we need to create some classes for "parts" of a job. We will start with a class that can represent an address.
Define the class `Address`. The class `Address` has 4 fields: `String street`, `int number`, `String zipCode` and `String city`.
Define the following methods:

> `public Address(String street, int number, String zipCode, String city)`
> This *constructor* initializes the fields with the arguments provided.
>
> `public String toString()`
> This method returns a human-friendly `String` representation of this object. You can choose the exact format yourself.
>
> `public boolean equals(Object other)`
> Post: If `other` is also an `Address` and has an equivalent `zipCode` and `number`, return `true`. Else return `false`. This means you do not need to check for equality between the `streetName` and `city` fields.
>
> A *getter* method for all the four fields of Address.

As you implement methods for your class `Address`, you should also create a class `AddressTest` and ensure you reach 100% branch coverage for the functionality you implement.

Assignment 4.2

Now that we have the addresses taken care of we need to create a structure for the equipment. We need several classes to represent the types of special equipment the contractor has. In order to make it so that we can easily create and use lists of different pieces of equiment we should apply **inheritance**.

Create an `abstract` class `Equipment`. The class has 1 fields: `String requirements`.
This is a textual description that indicates any special requirements the piece of equipment should meet for this particular job. Define the following method:

> `public Equipment(String requirements)`
> Post: Initializes the fields with the arguments provided.

Next we need 4 different classes that are going to extend from `Equipment`; these classes do not have any additional attributes.

> `ConcreteMixer`
>
> `JackHammer`
>
> `ScaffoldingTower`
>
> `Torch`

Implement a constructor for each of theses classes that takes `String requirements`, and passes this value on to the constructor of the parent (`Equipment`).
Addiontally implement an `equals` and a `toString` method for each of these classes. Don't forget to write tests! Since `Equipment` is an abstract class you cannot create any instances of this class. However you can test the getter of the `Equipment` class via one of the child classes. You only need to test the getter of `Equipment` in one of the four child classes.

Assignment 4.3

There is one final class we need before we can start putting together a class for the jobs. Create a class `Date` that has 3 fields: `int day`, `int month` and `int year`.
Define the following methods, and write tests for them:

> `public Date(int day, int month, int year)`
> This *constructor* initializes the fields with the arguments provided.
>
> `public String toString()`
> This method returns a human-friendly `String` representation of this object. You can choose the exact format yourself.
>
> `public boolean equals(Object other)`
> Post: If `other` is also a `Date` and has an equivalent `day`, `month` and `year`, return `true`. Else return `false`.
>
> A *getter* method for all the three fields of Date.

Assignment 4.4

Finally it's time to set up the `Job` class, and put it all together. The class `Job` has 6 attributes: `int jobNumber`, `Address location`, `String description`, `List<Equipment> requiredEquipment`, `Date plannedDate`. Finally there is `static int jobTotal` which is initialized at `0`.
Define the following methods, and write tests for them:

> `public Job(Address location, String description, List<Equipment> requiredEquipment, Date plannedDate)`
> This *constructor* initializes the fields with the arguments provided. Next to that the constructor increments the value of `jobTotal` by one, and then sets this value as the value for `jobNumber`.
>
> `public String toString()`
> This method returns a human-friendly `String` representation of this object. You can choose the exact format yourself.
>
> `public boolean equals(Object other)`
> Post: If `other` is also a `Job` and its attributes are equivalent, return `true`. Else return `false`. For this equals method, you do not need to compare the values for `jobTotal` and `jobNumber`.
>
> A *getter* method for all six fields of Job.

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
	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)