Project 3: Bartlebee's Hardware Shack

Alex Costinescu, Ayden Blotnick, and Lucas Sward

Language and Environment:

For this project, we used Java 1.8 and developed our simulation using the Eclipse IDE.

Design and Assumptions

Assumptions:
We assumed that every customer has a store in object in which they own and that they visit to receive tools from, i.e. being the same store object shared between customers. Customers do not keep track of previous rental records, as it is the store's job to do so. Therefore, the store has a list of archivedRentals. Each customer retains its behavior throughout the simulation, for example if you are instantiated as a business customer, this will not change. While rental options are based off the number of tools, since the price is modified via the number of options added, the rental record is what keeps track of the options instead of the tool. Inside the rental record one can view the options associated with the order as well as the number of tools. This allows the responsibility of keeping track of the cost to a single entity.

Project Description and Design:
Our project is designed such that Customers and Stores are contained within a World, which directs their interactions with each other.

The World, a Singleton, has a list of Customers and Stores and keeps track of the next staged day that will be executed upon calling the startNewDay method. This method handles printing the information required at the start of each day, then has the Stores followed by the Customers run through their daily responsibilities. A simulation can be run for any number of days using the runSimulation method, which also prints the relevant totals at the end of the simulation.

For Stores, daily responsibilities mean first checking if there are any active rentals, represented by RentalRecords, that need to be returned that day via the checkRentalRecords method. Stores are observed by Customers, so if they have any due RentalRecords they notify Customers for each RentalRecord that is due.

Customers' first responsibility is returning any due RentalRecords that they possess upon being updated by the Store's notify. Once the World has finished with Stores for the day, it then has each

Customer run through their day as well using the runDay function in Customer. This function determines if a Customer can AND will go to the Store, and if both cases are true then they generate a RentalRecord using the generateRental method. This method implements the Template Pattern, in that it makes calls to howMany and howLong, which are implemented differently by each different type of Customer. Once a RentalRecord has been generated, the Customer passes the record to the Store, which then “executes” the Record, removing the appropriate Tools from inventory and adding the RentalRecord to its list of active RentalRecords. This is an example of the Command Pattern. The Customer then finally adds the RentalRecord to its list of active RentalRecords as well.

The Customer class is an abstract class which is extended by each type of Customer: Business, Casual, and Regular. Each of these must implement the howMany, howLong, canRent, and getType methods, since these vary between different types of Customer. Customer Types are defined as final static integers within a separate class, called CustomerType, to ensure consistency across the program. While this adds additional work when adding a new type of Customer, the ease of use added by the CustomerType class was deemed worth the additional work. This pattern was also used for Rental Options and Tool Categories.

A RentalRecord is generated by a Customer and executed by a Store. It contains all of the information about a given rental including rental length, start day, total cost, the Tools rented, the RentalOptions added, and the Customer who rented the Tools. For performance, the total is calculated once when the RentalRecord is constructed and stored rather than being computed each time. Once a RentalRecord is created, it cannot be modified to prevent Customers and Stores from being able to modify details about current or previous rentals.

The Tool class is an abstract class which is extended by the 5 types of tools: WoodWorkingtool, PlumbingTool, ConcreteTool, YardworkTool, and PaintingTool. It’s attributes include the name of the tool. Each tool has a two getter method, one which returns its daily price as well as one which returns the category of tool that it is (concrete, yardwork, etc.)

The Main class is what instantiates the world, the customers, the tools, and the store. It executes the world’s runSimulation function in order to begin the exchange of tools and rental records between the store and the customers.

JUnit Testing

We used eclipse built in JUnitTest creation tool to create three separate testing interfaces for individual classes: TestWorld, TestRent, and TestStore. TestWorld ensures that the addStore, day incrementation, and runSimulation all work properly. The TestRent ensures that the business, casual and regular customers will each rent tools from the store taking into consideration a full inventory. There is also a test method that ensures that the addTool method is functional and when the customer rents tools, they will actually rent the correct amount based off the project

description. The TestStore ensures that the addTool method and removeTool method correctly increment and decrement the store's tool inventory. It also tests that the total calculated sales is correct. Lastly, it tests to see that the active and archived rentals are properly modified through a startRental and ProcessReturn functions.

UML Class Diagram

The UML diagram is located inside the Project3UML.png file.