Project - 3: Hardware Rental Store

Team Members:
Joshua Khoo
Elizabeth Robinson
Riad Shash (Ray)

Language and environment used for development:
We used Java 8, VS Code for development, and netBeans for building and running.

Program design and specification assumptions:
Our classes include:
Store: We implemented Store as a singleton (only one instance is ever created) and as Observable. When the inventory doesn't have any tools in it, it will **setIsInventory(Boolean)** to false - signifying that there is no inventory. This set function calls **setChanged()** and **notifyObservers()**, both Java Observable methods, which then **update()** (a Java Observer method) the variable in customer that keeps track of whether or not the inventory is empty. These functions are also called if the inventory size goes under 3 - the Business customer has its own update function to set its inventory empty variable if this is less than 3.
SimpleToolFactory: this is a simple factory that we used to create instances of different concrete tool classes based on their types.
Tool: An interface (non-Java term) to make sure all extended Tools are coded according to a contract. We did not implement base class Tool as a Java interface due to automatic static variables that come with Java interfaces.
PaintTool, ConcreteTool, WoodworkTool, PlumbingTool, YardworkTool: concrete classes that inherit from Tool.
StoreOption: StoreOption is an abstract class that provides information and methods for manipulating the "add-ons" a customer can rent when they rent a tool.
ProtectiveGearPackage, ExtensionCord, AccessoryKit: Concrete classes that inherit from StoreOption.
Customer: We implemented an abstract class Customer as an Observer of Store so each customer knows if there are tools to rent.
CasualCustomer, BusinessCustomer, RegularCustomer: these are the concrete classes that inherit from Customer.
Record: This is a class that manipulates and stores any data pertaining to rental records.
storeSimulation: This is a class described below that will run an instance of the 35 days required.

We used ThreadLocalRandom to generate random numbers due to problems with using typical Random Class (Got the same values in the same threads).

When the simulation reaches day 35, the final report prints all the **completed*** rentals (rentals that have been returned to the store, but not any current rentals that have not been returned yet).

The simulation runs by creating an instance of **storeSimulator** class and calling **runSimulation()** method done by our MainApp class in main(). We redirect output to out.txt.

We used the following Software Patterns:

1. **Simple Factory** -> To create Tools needed for the simulation
2. **Singleton** -> To make sure we have one & only one instance of Store
3. **Observer** -> To determine if customers can enter the store

UML Class Diagram Below

Project-3

Hardware Rental Store