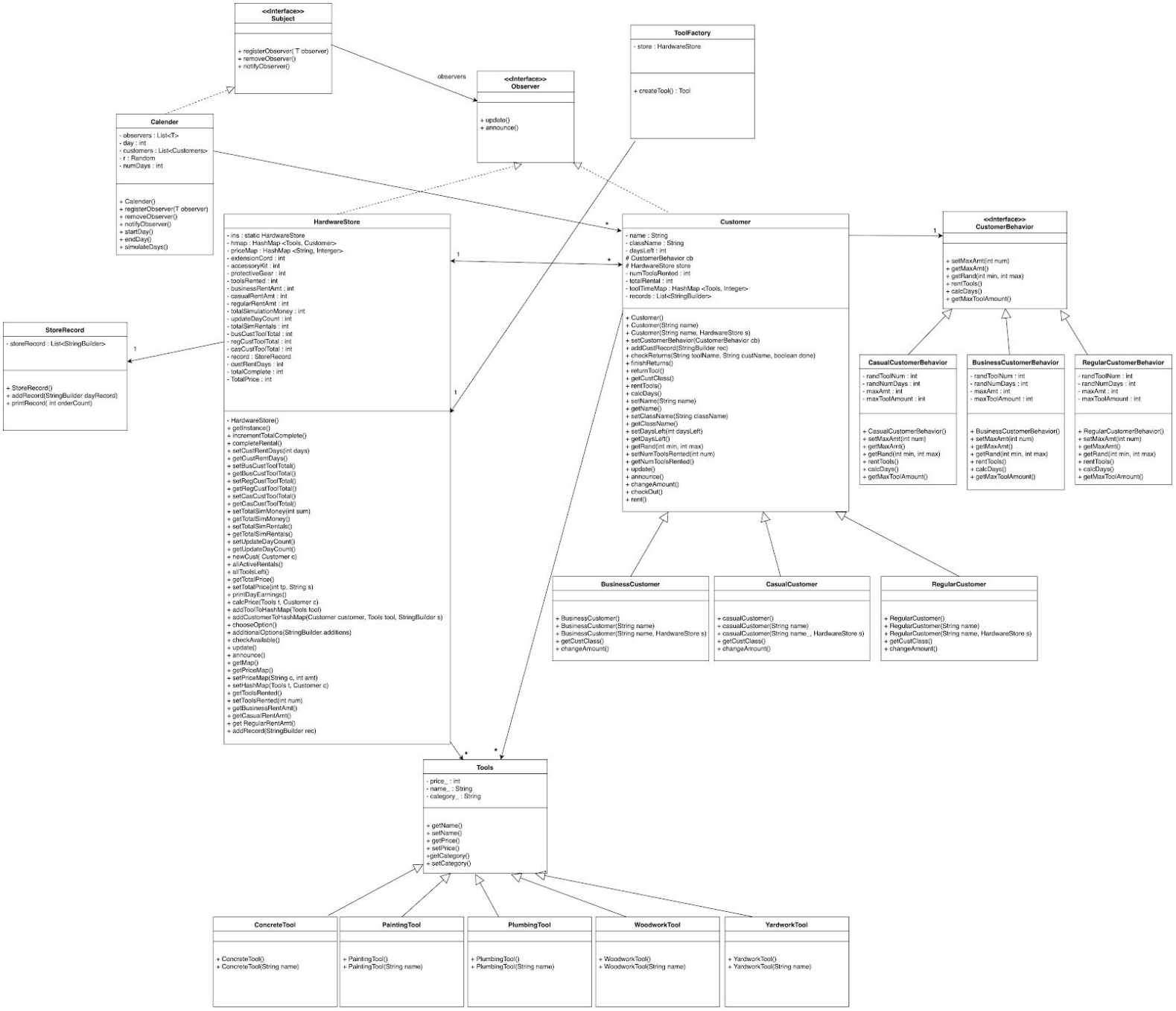
Team: Kyle Schultz, Aaron Steiner, Pari Bhetwal
Project: OOAD Project 3
Language and Environment: Java 8, Sublime/IntelliJ

UML Class Diagram:
-If you're having trouble viewing this, let us know and we can pull it up for you.



Text Description: We used the observer pattern to facilitate the calendar class and update the days within the simulation. We used the strategy pattern to break the customer behaviors into their respective behaviors regarding tool rentals and the number of days that they rent tools. A factory pattern was used to create the tools. We used the singleton pattern to make sure only one instance of HardwareStore and Calendar were created. The basic flow of the program is that tools are created, customers are created. At the beginning of the day the day number, the list of active rentals, the number of tools left in the store, a list of tools left in the store, the total number of completed rentals, the list of completed rentals, and the amount of money they made each day are being printed out. During the day a random number of customers come into the store to rent a number of tools based on for a number of days based on their behavior which are stored in separate hash maps. Before checking out the tool, the customer will add a random number from 0 to 6 of additional options. Each day the store will see how many tools it has so it knows if customers can come in to rent any. Calendar will increment the days by one at the end of each day and update HardwareStore and Customer with that information. When customers return tools their respective assignment to the tool via the hashmap is cleared and the tool is set to null in the hashmap. When a tool is returned the Rental Record is updated with the Customer's information of which tool they rented, the amount of days they rented it, the options they added, and their total at checkout. This list is printed each day. The totals for the end of the simulation are incremented in the getters and setters for those numbers. So, at the end of each day, the store total was added to the overall store simulation total. We implemented 10 JUnit tests to verify various aspects of the program to make sure it performed as expected. Those instructions are included in the README file on GitHub.

Design Assumptions: We assumed that the customer can rent on the last day, even though the simulation will not run past day 35. We assumed the tools the customer is renting, options and total fees should be printed on the day they are renting them. We wrote this as 'Customer <Customer> is adding tool <Tool> to their shopping cart. They are renting to tool for <int> days. They are adding option <String> to their shopping cart before checkout. Customer <Customer> is checking out and paying <int>.' and each sentence is printed on a new line. We assumed the list of completed rentals to be printed includes all of the completed rentals up to this day. The total number of rentals for each customer type (i.e. Regular Customer) is the number of tools rented and **not** number of completed orders. We are assuming the completed number of rentals is the number of individual tools rented and not the number of customers that are renting.

Citations: We learned how to use an iterator for going through hashmaps from
-https://www.geeksforgeeks.org/iterate-map-java/