```
package HardwareRentalStore;

import Customer.*;
import Rental.*;
import Tools.*;
import java.math.*;
import MyUnitTest.*;
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Random;
import java.util.Iterator;
import java.text.DecimalFormat;

public class HardwareRentalStore {

    public ArrayList<Tool> tools;
    public ArrayList<RentalRecord> activeRentals;
    public ArrayList<RentalRecord> oldRentals;
    public Map <String, Integer> customerHistory;



    public HardwareRentalStore() {
        ArrayList<Tool> tools = new ArrayList<Tool>(20);
        this.tools = tools;

        ArrayList<RentalRecord> activeRentals = new ArrayList<RentalRecord>();
        this.activeRentals = activeRentals;

        ArrayList<RentalRecord> oldRentals = new ArrayList<RentalRecord>();
        this.oldRentals = oldRentals;

        Map <String, Integer> customerHistory = new HashMap<String, Integer>();
        customerHistory.put("Casual", 0);
        customerHistory.put("Regular", 0);
        customerHistory.put("Business", 0);
        customerHistory.put("Total", 0);
        this.customerHistory = customerHistory;


    }


    public  ArrayList<Customer> createCustomers(){

        ArrayList<Customer> customers = new ArrayList<Customer>();

        customers.add( new Casual("Carl"));
        customers.add( new Casual("Cathy"));
```

```
    customers.add( new Casual("Chuckie"));
    customers.add( new Casual("Chris"));
    customers.add( new Casual("Craig"));
    customers.add( new Casual("Chuck"));
    customers.add( new Casual("Claude"));
    customers.add( new Regular("Regina"));
    customers.add( new Regular("Red"));
    customers.add( new Regular("Rhonda"));
    customers.add( new Regular("Russ"));
    customers.add( new Business("Brad"));


    return customers;

}

public ArrayList<Tool> createTools(){
    // SIMPLY FACTORY PATTERN
    SimpleToolFactory toolFactory = new SimpleToolFactory();
    ToolStore toolStore = new ToolStore(toolFactory);


    this.tools.add(toolStore.createTool("Painting", "Paint Tool 1"));
    this.tools.add(toolStore.createTool("Painting", "Paint Tool 2"));
    this.tools.add(toolStore.createTool("Painting", "Paint Tool 3"));
    this.tools.add(toolStore.createTool("Painting", "Paint Tool 4"));
    this.tools.add(toolStore.createTool("Painting", "Paint Tool 5"));
    this.tools.add(toolStore.createTool("Concrete", "Concrete Tool 1"));
    this.tools.add(toolStore.createTool("Concrete", "Concrete Tool 2"));
    this.tools.add(toolStore.createTool("Concrete", "Concrete Tool 3"));
    this.tools.add(toolStore.createTool("Concrete", "Concrete Tool 4"));
    this.tools.add(toolStore.createTool("Concrete", "Concrete Tool 5"));
    this.tools.add(toolStore.createTool("Plumbing", "Plumbing Tool 1"));
    this.tools.add(toolStore.createTool("Plumbing", "Plumbing Tool 2"));
    this.tools.add(toolStore.createTool("Plumbing", "Plumbing Tool 3"));
    this.tools.add(toolStore.createTool("Plumbing", "Plumbing Tool 4"));
    this.tools.add(toolStore.createTool("Woodwork", "Woodwork Tool 1"));
    this.tools.add(toolStore.createTool("Woodwork", "Woodwork Tool 2"));
    this.tools.add(toolStore.createTool("Woodwork", "Woodwork Tool 3"));
    this.tools.add(toolStore.createTool("Woodwork", "Woodwork Tool 4"));
    this.tools.add(toolStore.createTool("Yardwork", "Yardwork Tool 1"));
    this.tools.add(toolStore.createTool("Yardwork", "Yardwork Tool 2"));
    this.tools.add(toolStore.createTool("Yardwork", "Yardwork Tool 3"));
    this.tools.add(toolStore.createTool("Yardwork", "Yardwork Tool 4"));
    this.tools.add(toolStore.createTool("Yardwork", "Yardwork Tool 5"));
    this.tools.add(toolStore.createTool("Yardwork", "Yardwork Tool 6"));

    return this.tools;

}

public void updateInventoryRent(ArrayList<Tool> rentedToolsNoDec) {
    // update inventory when a rental transaction has occured
    for (Tool tool : rentedToolsNoDec) {
```

```
            int i = this.tools.indexOf(tool);
            this.tools.get(i).setInStock(false);
        }
    }

    public void updateInventoryReturn(ArrayList<Tool> rentedToolsNoDec) {
        // update inventory when a rental has expired
        for (Tool tool : rentedToolsNoDec) {
            int i = this.tools.indexOf(tool);
            this.tools.get(i).setInStock(true);
        }
    }

    public ArrayList<Tool> availableTools() {
        // iterate through tools and gather an ArrayList
        // of which tools are available to rent
        ArrayList<Tool> availableTools = new ArrayList<Tool>();
        for (Tool tool : this.tools) {
            if (tool.inStock == true) {
                availableTools.add(tool);
            }
        }
        return availableTools;
    }

    public void updateCustomerHistory(RentalRecord rental) {
        // casual index 0 --- reg index 1 -- biz index 2
        if (rental.customer.type == "Casual") {
            this.customerHistory.put("Casual", this.customerHistory.get("Casual") + 1);
        } else if (rental.customer.type == "Regular") {
            this.customerHistory.put("Regular", this.customerHistory.get("Regular") + 1);
        } else {
            this.customerHistory.put("Business", this.customerHistory.get("Business") + 1);
        }
        this.customerHistory.put("Total", this.customerHistory.get("Total") + 1);
    }

    public void printCustomerHistory() {
        System.out.println("Total Rentals for Casual Customers: " +
this.customerHistory.get("Casual"));
        System.out.println("Total Rentals for Regular Customers: " +
this.customerHistory.get("Regular"));
        System.out.println("Total Rentals for Business Customers: " +
this.customerHistory.get("Business"));
        System.out.println("Total Rentals: " + this.customerHistory.get("Total"));
    }

    public double customerSimulation(ArrayList<Customer> customersArr) {

        // to keep track of which rentals returned
        ArrayList<RentalRecord> temp = new ArrayList<RentalRecord>();

        // selecting random number of customers
        Random rand = new Random();
```

```
// to display count of rentals at the end of each day
int active = 0;
int returned = 0;


// ITERATOR DESIGN PATTERN
// used to iterate through every customer for each day
Iterator<Customer> it = customersArr.iterator();
while (it.hasNext()) {
    Customer customer = it.next();
    for (RentalRecord rental : customer.activeRentals) {
        if (rental.numOfDays == 0) {
            temp.add(rental);
            returned += 1;
        } else {
            rental.printRentalDesc("Active");
            rental.decreaseRentalDay();
            active += 1;
        }
    }
}

// ITERATOR DESIGN PATTERN & FACADE
// iterate through rentals to remove expired rentals from active rentals list
// and trigger an expiredRental() which calls a number of different methods
// on different classes to update (FACADE)
Iterator<RentalRecord> itRent = temp.iterator();
while (itRent.hasNext()) {
    RentalRecord rental = itRent.next();
    rental.printRentalDesc("Completed");
    int remove = rental.customer.activeRentals.indexOf(rental);
    RentalRecord rent = rental.customer.activeRentals.get(remove);
    rent.expiredRental();
}

// random number of visitors
int r = (int) (Math.random() * (customersArr.size() - 1)) + 1;

// to keep track of each daily profit
double totalProfit = 0.0;

// If there is inventory and the customer has less than 3 active rentals
// complete a rental transaction. If not, skip.
// A FACADE is used on the  update() method which calls a number of different methods
// on different classes to update
for (int i = 0; i < r; i++) {
    if (this.availableTools().size() == 0) {
        System.out.println("\n------ Empty Inventory ------ ");
        return 0.0;
    } else {
        // select random customer
        int randomIndex = rand.nextInt(customersArr.size());
        Customer customer = customersArr.get(randomIndex);
```

```
            if ((customer.numRentals < 3)) {
                if (this.availableTools().size() > 0) {
                    RentalRecord rental = new RentalRecord(customer, this);
                    rental.printRentalDesc("New");
                    rental.update();
                    totalProfit += rental.fee;
                }
            }
        }
    }

    // display
    System.out.println("Active Rentals: " + active);
    System.out.println("Returned Rentals: " + returned);
    this.printInventory();
    return totalProfit;
}

public void printInventory() {
    System.out.println("\n########## End of Day Inventory ##########\n");

    for (Tool tool : this.tools) {
        System.out.println(tool.name + " - " + tool.inStock);
    }
    System.out.println("\n##########################################\n");

}

public void daySimulaiton(ArrayList<Customer> customersArr) {
    // formatting purposes
    DecimalFormat df = new DecimalFormat("####.##");
    df.setRoundingMode(RoundingMode.DOWN);

    // cycle through each day
    double monthlyProfit = 0.0;
    for (int i = 1; i <=35; i++) {
        System.out.println("\n\n ********** DAY " + i + " **********\n\n");
        double profit = this.customerSimulation(customersArr);
        monthlyProfit += profit;
        System.out.println("Total Profit for the day: $" + df.format(profit));
    }

    System.out.println("\n\n\n********** End of Month **********\n\n");
    System.out.println("Total Profit for the month: $" + df.format(monthlyProfit));
    this.printCustomerHistory();

}

public static void main(String[] args) {

    // EXTRA CREDIT JUNIT TESTS
    MyUnitTest myUnitTest = new MyUnitTest();
    JUnitCore junit = new JUnitCore();
    Result result = junit.run(MyUnitTest.class);
```

```
        // SET UP
        HardwareRentalStore hardwareRentalStore = new HardwareRentalStore();
        hardwareRentalStore.createTools();
        ArrayList<Customer> customersArr = hardwareRentalStore.createCustomers();

        // SIMULATION
        hardwareRentalStore.daySimulaiton(customersArr);




    }
}
```

```
package Customer;

import Rental.RentalRecord;

public class Business extends Customer{
    public String type;
    public RentalRecord activeRental;

    public Business(String name) {

        super(name, "Business");
    }

    public void setActiveRental(RentalRecord activeRental) {
        this.activeRental = activeRental;
    }

    public  int getValidRentalTime(){
        return 7;
    }

    public  int getValidRentalSize(){
        return 3;
    }
}
```

```
package Customer;

import Rental.RentalRecord;

import java.util.ArrayList;

public class Casual extends Customer{
//    public String type;

    public Casual(String name) {
        super(name, "Casual");
    }
```

```
    public  int getValidRentalTime(){
        int r = (int) (Math.random() * (3 - 1)) + 1;
        return r;
    }

    public  int getValidRentalSize(){
        int r = (int) (Math.random() * (3 - 1)) + 1;
        if (r + this.numRentals > 3) {
            return 1;
        }
        return r;
    }

}
```

```
package Customer;

import Rental.RentalRecord;

import java.util.ArrayList;

public abstract class Customer {

    // abstract class for customer

    public String name;
    public String type;
    public ArrayList<RentalRecord> activeRentals;
    public int numRentals;


    public Customer(String name, String type){

        if (name == "") {
            throw new IllegalArgumentException("No inputted name.");
        }

        this.name = name;
        this.type = type;
        this.activeRentals = new ArrayList<RentalRecord>();
        this.numRentals = 0;

    }

    // subclasses to define valid rental times and
    // rental sizes based on which type of customer they are
    public abstract int getValidRentalTime();
    public abstract int getValidRentalSize();
}
```

```
package Customer;
```

```
import Rental.RentalRecord;

public class Regular extends Customer{
    public RentalRecord activeRental;

    public Regular(String name) {
        super(name, "Regular");
    }

    public void setActiveRental(RentalRecord activeRental) {
        this.activeRental = activeRental;
    }

    public  int getValidRentalTime(){
        int r = (int) (Math.random() * (6 - 3)) + 3;
        return r;
    }

    public  int getValidRentalSize(){
        int r = (int) (Math.random() * (4 - 1)) + 1;
        int ret;
        if (r +  this.numRentals > 3 ) {
            return 3-this.numRentals;
        }
        return r;
    }
}
```

```
package Rental;

import Customer.*;
import Tools.*;
import HardwareRentalStore.*;
import Tools.ToolDecorator;

import java.math.RoundingMode;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Random;
import java.util.Map;
import java.util.HashMap;


/* Facade in update() and expiredRental()*/

public class RentalRecord {
    public Customer customer;
    public ArrayList<Tool> rentedTools;
    public ArrayList<Tool> rentedToolsNoDec;
    public int numOfDays;
    public int numOfTools;
    public double fee;
    public HardwareRentalStore hardwareRentalStore;
```

```
    public RentalRecord(Customer customer, HardwareRentalStore hardwareRentalStore) {
        this.numOfTools = customer.getValidRentalSize();
        ArrayList<Tool> availableTools = hardwareRentalStore.availableTools();
        Map <Integer, ArrayList<Tool>> tools = getRandomAvailableTools(this.numOfTools,
availableTools);
        this.rentedTools = tools.get(0);
        this.rentedToolsNoDec = tools.get(1);
        this.numOfDays = customer.getValidRentalTime();
        this.fee = getTotalCost(rentedTools);
        this.customer = customer;
        this.hardwareRentalStore = hardwareRentalStore;
    }

    /*
    Facade:
    - when a rental is created, many subsets within the system need to be updated. By
    creating an 'update' method, we are exposing a simple interface that handles all
    of the actions that must exist by the creation of a new rental.
     */

    public void update() {
        this.hardwareRentalStore.updateInventoryRent(this.rentedToolsNoDec);
        this.hardwareRentalStore.updateCustomerHistory(this);
        this.customer.activeRentals.add(this);
        this.customer.numRentals = this.customer.numRentals + this.numOfTools;
        this.numOfDays -= 1;

    }

    /*
    Facade:
    - when a rental is returned, many subsets within the system need to be updated. By
    creating a 'expiredRental' method, we are exposing a simple interface that handles all
    of the actions that must exist by the return of a  rental.
     */

    public void expiredRental() {
        this.hardwareRentalStore.oldRentals.add(this);
        this.hardwareRentalStore.activeRentals.remove(this);
        this.hardwareRentalStore.updateInventoryReturn(this.rentedToolsNoDec);
        this.customer.activeRentals.remove(this);
        this.customer.numRentals = this.customer.numRentals - this.numOfTools;
    }

    public void decreaseRentalDay() {
        this.numOfDays -= 1;
    }

    public void printRentalDesc(String id) {
        DecimalFormat df = new DecimalFormat("####.##");
        df.setRoundingMode(RoundingMode.DOWN);
        System.out.println("\n          ==========" + id + " Rental ==========\n");
        System.out.println("          | Customer: " + customer.name + " -- " + customer.type);
        double totalCost = 0;
```

```
        for (Tool tool : this.rentedTools) {
            System.out.println("          | Tool: " + tool.getDescription());
            totalCost = totalCost + tool.cost();
        }
        System.out.println("          | Days: " + this.numOfDays);
        System.out.println("          | Total Cost: $" + df.format(totalCost));
        System.out.println("\n          ============================\n");
    }

    public double getTotalCost(ArrayList<Tool> tools) {
        double cost = 0;
        for (Tool tool : tools) {
            cost = cost + tool.cost();
        }
        return cost;
    }

    public Map <Integer, ArrayList<Tool>> getRandomAvailableTools(int numOfTools,
ArrayList<Tool> availableTools) {
        ArrayList<Tool> rentedToolsNoDec = new ArrayList<Tool>(numOfTools);
        for (int i = 0; i < numOfTools; i++) {
            if (availableTools.size() == 0) {break;}
            int randomIndex = (int)(Math.random() * availableTools.size());
            rentedToolsNoDec.add(availableTools.get(randomIndex));
            availableTools.remove(randomIndex);
        }

        // add decorator
        int numOfDecorators = (int) (Math.random() * (7 - 1)) + 1;
        Tool tool = rentedToolsNoDec.get(0);
        ArrayList<Tool> rentedTools = (ArrayList<Tool>)rentedToolsNoDec.clone();
        Tool temp = tool;
        for (int i = 0; i <= numOfDecorators; i++) {
            int index = (int) (Math.random() * (4 - 1)) + 1;
            if (index == 1) {
              temp  = new ExtensionCordDecorator(tool);
            }
            if (index == 2) {
                temp = new AccessoryKit(tool);
            }
            else {
                temp = new ProtectiveGearPackage(tool);
            }
            tool = temp;
        }
        rentedTools.set(0,temp);


        Map <Integer, ArrayList<Tool>> ret = new HashMap<Integer, ArrayList<Tool>>();
        ret.put(0, rentedTools);
        ret.put(1, rentedToolsNoDec);

        return ret;
    }
```

```
}
```

```
package Tools;

public class AccessoryKit extends ToolDecorator {
    Tool tool;

    public AccessoryKit(Tool tool) {
        this.tool = tool;
        this.name = "Accessory Kit";
        this.cost = 8.00;
        this.description = "**Addition** | Name: " + this.name + " | ";
        this.inStock = true;

    }

    public double cost() {
        return tool.cost() + this.cost;
    }

    public String getDescription() {
        return tool.getDescription() + this.description;
    }

    public void setInStock(boolean inStock) {
        this.inStock = true;
    }
}
```

```
package Tools;

public class ConcreteTool extends Tool {
    public ConcreteTool(String name) {
        this.name = name;
        this.type = "Concrete";
        this.cost = 5.53;
        this.description = this.type + " | Name: " + this.name + " | ";
        this.inStock = true;

    }

    public double cost() {
        return this.cost;
    }

    public String getDescription() {
        return this.description;
    }

    public void setInStock(boolean inStock) {
        this.inStock = inStock;
    }
```

```
}

package Tools;

public class ExtensionCordDecorator extends ToolDecorator {
    Tool tool;

    public ExtensionCordDecorator(Tool tool) {
        this.tool = tool;
        this.name = "Extension Cord";
        this.cost = 4.04;
        this.description = "**Addition** | Name: " + this.name + " | ";
        this.inStock = true;

    }

    public double cost() {
        return tool.cost() + this.cost;
    }

    public String getDescription() {
        return tool.getDescription() + this.description;
    }

    public void setInStock(boolean inStock) {
        this.inStock = true;
    }
}


package Tools;

public class PaintingTool extends Tool {
    public PaintingTool(String name) {
        this.name = name;
        this.type = "Painting";
        this.cost = 3.57;
        this.description = this.type + " | Name: " + this.name + " | ";
        this.inStock = true;

    }

    public double cost() {
        return this.cost;
    }

    public String getDescription() {
        return this.description;
    }

    public void setInStock(boolean inStock) {
            this.inStock = inStock;
    }
}
```

```
package Tools;

public class PlumbingTool extends Tool {
    public PlumbingTool(String name) {
        this.name = name;
        this.type = "Plumbing";
        this.cost = 7.77;
        this.description = this.type + " | Name: " + this.name + " | ";
        this.inStock = true;

    }

    public double cost() {
        return this.cost;
    }

    public String getDescription() {
        return this.description;
    }

    public void setInStock(boolean inStock) {
        this.inStock = inStock;
    }
}

package Tools;

public class ProtectiveGearPackage extends ToolDecorator {
    Tool tool;

    public ProtectiveGearPackage(Tool tool) {
        this.tool = tool;
        this.name = "Protective Gear Package";
        this.cost = 11.13;
        this.description = "**Addition** | Name: " + this.name + " | ";
        this.inStock = true;

    }

    public double cost() {
        return tool.cost() + this.cost;
    }

    public String getDescription() {
        return tool.getDescription() + this.description;
    }

    public void setInStock(boolean inStock) {
        this.inStock = true;
    }
}

package Tools;
```

```
public class SimpleToolFactory {

	// encapsulate the creation of a tool

	public Tool createTool(String type, String name) {
	Tool tool = null;
	if (type.equals("Painting")) {
	   tool = new PaintingTool(name);
	}
	else if (type.equals("Concrete")) {
	   tool = new ConcreteTool(name);
	}
	else if (type.equals("Plumbing")) {
	   tool = new PlumbingTool(name);
	}
	else if (type.equals("Woodwork")) {
	   tool = new WoodworkTool(name);
	}
	else if (type.equals("Yardwork")) {
	      tool = new YardworkTool(name);
	}
	else {
	   throw new IllegalArgumentException("Invalid Input.");
	}
	return tool;
   }
}


package Tools;

import java.util.ArrayList;
import java.util.Random;

public abstract class Tool {

   // abstract class for tool

   public String name;
   public String type;
   public double cost;
   public String description;
   public boolean inStock;


   // subclasses are to define methods
   // based on which type of tool they are
   public abstract double cost();
   public abstract String getDescription();

   public abstract void setInStock(boolean inStock);

}
```

```
package Tools;

public abstract class ToolDecorator extends Tool {

    // DECORATOR DESIGN PATTERN


    public String name;
    public double cost;
    public String description;
    public abstract double cost();
    public abstract String getDescription();
}

package Tools;

public  class ToolStore {

    SimpleToolFactory factory;

    public ToolStore(SimpleToolFactory factory) {
        this.factory = factory;
    }
    public Tool createTool(String type, String name) {
        Tool tool;
        tool = factory.createTool(type, name);
        return tool;
    }

}


package Tools;

public class WoodworkTool extends Tool {
    public WoodworkTool(String name) {
        this.name = name;
        this.type = "Woodwork";
        this.cost = 9.99;
        this.description = this.type + " | Name: " + this.name + " | ";
        this.inStock = true;

    }

    public double cost() {
        return this.cost;
    }

    public String getDescription() {
        return this.description;
    }
```

```
    public void setInStock(boolean inStock) {
        this.inStock = inStock;
    }
}
```

```
package Tools;

public class YardworkTool extends Tool {
    public YardworkTool(String name) {
        this.name = name;
        this.type = "Yardwork";
        this.cost = 7.75;
        this.description = this.type + " | Name: " + this.name + " | ";
        this.inStock = true;

    }

    public double cost() {
        return this.cost;
    }

    public String getDescription() {
        return this.description;
    }

    public void setInStock(boolean inStock) {
        this.inStock = inStock;
    }
}
```

```
package MyUnitTest;

import Customer.*;
import HardwareRentalStore.HardwareRentalStore;
import Rental.RentalRecord;
import Tools.ExtensionCordDecorator;
import Tools.SimpleToolFactory;
import Tools.Tool;
import Tools.ToolStore;
import org.junit.Test;

import java.util.ArrayList;

import static org.junit.Assert.*;

public class MyUnitTest {
```

```
@Test
public void testCreateTool() {
    SimpleToolFactory toolFactory = new SimpleToolFactory();
    ToolStore toolStore = new ToolStore(toolFactory);

    Tool tool = toolStore.createTool("Painting", "Test Name");


    try {
        assertEquals("Test Name", tool.name);       }
    catch (AssertionError e) {
        System.out.println("Test 1 - Failed");
        throw e;
    }
    System.out.println("Test 1 - Passed");

}

@Test (expected = RuntimeException.class)
public void testCreateTool2()
{
    try
    {
        SimpleToolFactory toolFactory = new SimpleToolFactory();
        ToolStore toolStore = new ToolStore(toolFactory);

        Tool tool = toolStore.createTool("Nonsense", "Test Name");       }
    catch(RuntimeException re)
    {
        System.out.println("Test 2 - Passed");
        System.out.println("       Exception caught ");
        String message = "Invalid Input.";
        assertEquals(message, re.getMessage());
        throw re;
    }
    fail("Invalid input exception did not throw!");
}

@Test
public void testCasualCustomer() {
    Customer customer = new Casual("Cate");
    int time = customer.getValidRentalTime();
    int size = customer.getValidRentalSize();

    try {
        assertEquals("Cate", customer.name);}
    catch (AssertionError e) {
        System.out.println("Test 3 - Failed");
        throw e;
    }
    System.out.println("Test 3 - Passed");
```

```
}


@Test
public void testBusinessCustomer() {
    Customer customer = new Business("Bob");
    int time = customer.getValidRentalTime();
    int size = customer.getValidRentalSize();

    try {
        assertEquals("Bob", customer.name);}
    catch (AssertionError e) {
        System.out.println("Test 4 - Failed");
        throw e;
    }
    System.out.println("Test 4 - Passed");


}

@Test
public void testRegularCustomer() {
    Customer customer = new Regular("Reggie");
    int time = customer.getValidRentalTime();
    int size = customer.getValidRentalSize();

    try {
        assertEquals("Reggie", customer.name);}
    catch (AssertionError e) {
        System.out.println("Test 5 - Failed");
        throw e;
    }
    System.out.println("Test 5 - Passed");


}

@Test
public void testGetAvailableTools() {
    HardwareRentalStore store = new HardwareRentalStore();
    SimpleToolFactory toolFactory = new SimpleToolFactory();
    ToolStore toolStore = new ToolStore(toolFactory);


    store.tools.add(toolStore.createTool("Painting", "Paint Tool 1"));
    store.tools.add(toolStore.createTool("Concrete", "Concrete Tool 1"));
    store.tools.add(toolStore.createTool("Yardwork", "Yardwork Tool 1"));


    Customer customer = new Business("Bob");
    RentalRecord rental = new RentalRecord(customer, store);
    rental.update();

    ArrayList<Tool> availableTools = new ArrayList<Tool>();
```

```
    availableTools = store.availableTools();


    try {
        assertEquals(0, availableTools.size());}
    catch (AssertionError e) {
        System.out.println("Test 6 - Failed");
        throw e;
    }
    System.out.println("Test 6 - Passed");
}

@Test
public void testGetAvailableTools2() {
    HardwareRentalStore store = new HardwareRentalStore();
    SimpleToolFactory toolFactory = new SimpleToolFactory();
    ToolStore toolStore = new ToolStore(toolFactory);


    store.tools.add(toolStore.createTool("Painting", "Paint Tool 1"));
    store.tools.add(toolStore.createTool("Concrete", "Concrete Tool 1"));



    ArrayList<Tool> availableTools = new ArrayList<Tool>();

    availableTools = store.availableTools();


    try {
        assertEquals(2, availableTools.size());}
    catch (AssertionError e) {
        System.out.println("Test 7 - Failed");
        throw e;
    }
    System.out.println("Test 7 - Passed");
}

@Test
public void testCostNoDecorator() {
    HardwareRentalStore store = new HardwareRentalStore();
    SimpleToolFactory toolFactory = new SimpleToolFactory();
    ToolStore toolStore = new ToolStore(toolFactory);


    store.tools.add(toolStore.createTool("Painting", "Paint Tool 1"));
    store.tools.add(toolStore.createTool("Concrete", "Concrete Tool 1"));


    Customer customer = new Business("Bob");
    RentalRecord rental = new RentalRecord(customer, store);

    ArrayList<Tool> availableTools = new ArrayList<Tool>();
```

```
    availableTools = store.availableTools();

    double cost = rental.getTotalCost(availableTools);


    try {
        assertEquals(9.1, cost, 1);}
    catch (AssertionError e) {
        System.out.println("Test 8 - Failed");
        throw e;
    }
    System.out.println("Test 8 - Passed");
}

@Test
public void testCostWithDecorator() {
    SimpleToolFactory toolFactory = new SimpleToolFactory();
    ToolStore toolStore = new ToolStore(toolFactory);


    Tool tool = toolStore.createTool("Painting", "Paint Tool 1");
    Tool toolWithDecorator = new ExtensionCordDecorator(tool);

    double cost = toolWithDecorator.cost();




    try {
        assertEquals(7.61, cost, 1);}
    catch (AssertionError e) {
        System.out.println("Test 9 - Failed");
        throw e;
    }
    System.out.println("Test 9 - Passed");
}

@Test (expected = RuntimeException.class)
public void testEmptyName2()
{
    try
    {
        Customer customer = new Business("");
    }
    catch(RuntimeException re)
    {
        System.out.println("Test 10 - Passed");
        System.out.println("       Exception caught ");
        String message = "No inputted name.";
        assertEquals(message, re.getMessage());
        throw re;
    }
    fail("No inputted name exception did not throw!");
```

```
    }
}
```