KitchIntel
Control System

A Summary of the Discussion at the Planning Meeting for Sprint 3

Overview of the Sprint

The team is going to implement and test the controllers for various hardware devices.

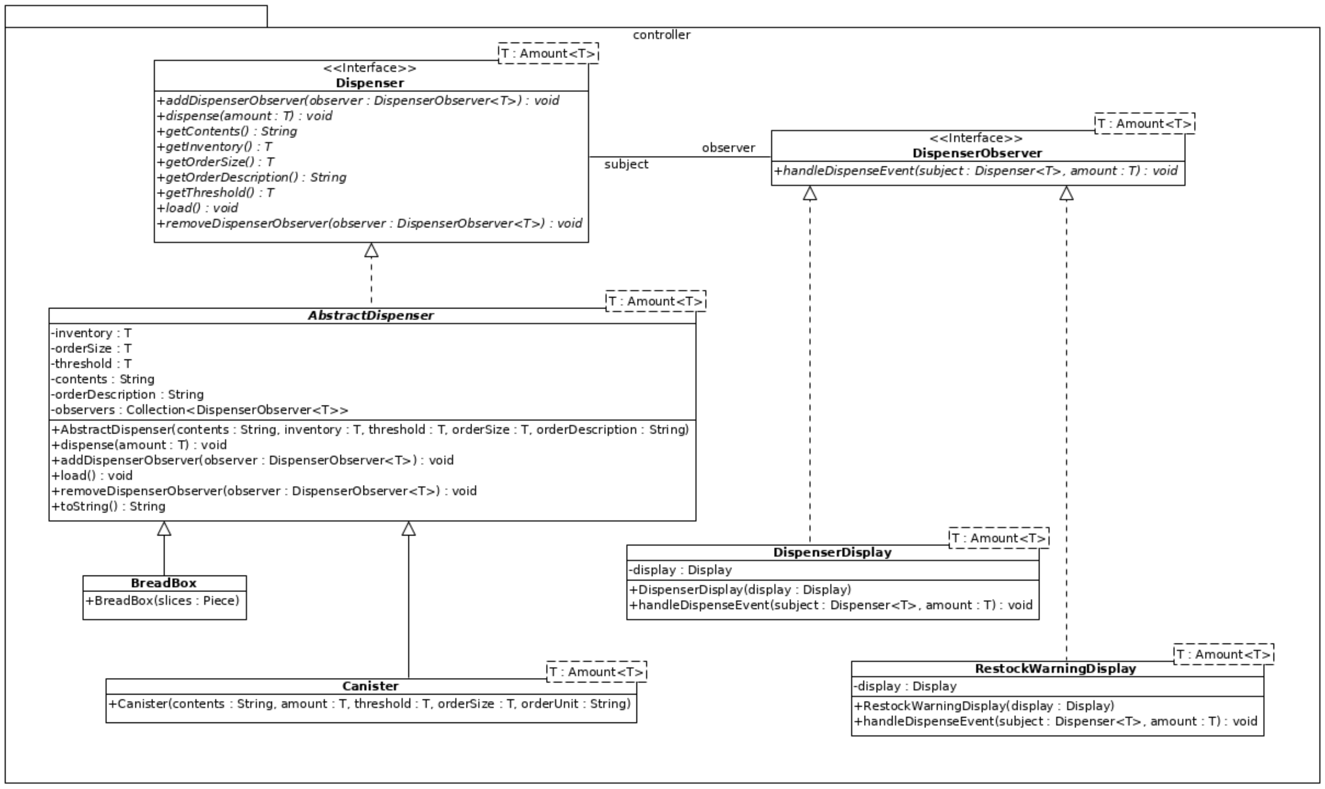
Overview of My Committments

I committed to implementing and testing the controllers for the current increment of the “dispensers” (i.e., the hardware devices that will dispense products to users). In particular, I committed to implementing and testing the controller for a canister (i.e., a cylindrical container for storing and dispensing things like flour, sugar, cookies, etc.) and a bread box.

It’s important for me to remember that dispensers must be able to report when they dispense some of the product they contain (e.g., imagine a vending machine). This must happen when their `dispense()` method is invoked. At the moment, they may need to report this information to the user and/or to a system that automatically places orders when the level of inventory in the dispenser falls below a particular threshold.

The Design

The team agreed to the following design.


controller
T : Amount<T>
<<Interface>>
Dispenser
+addDispenserObserver(observer : DispenserObserver<T>) : void
+dispense(amount : T) : void
+getContents() : String
+getInventory() : T
+getOrderSize() : T
+getOrderDescription() : String
+getThreshold() : T
+load() : void
+removeDispenserObserver(observer : DispenserObserver<T>) : void
subject
observer
T : Amount<T>
<<Interface>>
DispenserObserver
+handleDispenseEvent(subject : Dispenser<T>, amount : T) : void
T : Amount<T>
AbstractDispenser
-inventory : T
-orderSize : T
-threshold : T
-contents : String
-orderDescription : String
-observers : Collection<DispenserObserver<T>>
+AbstractDispenser(contents : String, inventory : T, threshold : T, orderSize : T, orderDescription : String)
+dispense(amount : T) : void
+addDispenserObserver(observer : DispenserObserver<T>) : void
+load() : void
+removeDispenserObserver(observer : DispenserObserver<T>) : void
+toString() : String
BreadBox
+BreadBox(slices : Piece)
T : Amount<T>
DispenserDisplay
-display : Display
+DispenserDisplay(display : Display)
+handleDispenseEvent(subject : Dispenser<T>, amount : T) : void
T : Amount<T>
Canister
+Canister(contents : String, amount : T, threshold : T, orderSize : T, orderUnit : String)
T : Amount<T>
RestockWarningDisplay
-display : Display
+RestockWarningDisplay(display : Display)
+handleDispenseEvent(subject : Dispenser<T>, amount : T) : void

Details

`Dispenser` Interface
The contents is a description of the product in the dispenser (e.g., “Flour”).
The inventory is the amount of the product currently in the dispenser.
If the inventory falls below the threshold then an order will be placed.
The order size is the amount of the product that will be ordered when an order is placed.
The order description is a textual description of the order.
The `dispense()` method is invoked when the user “removes” some of the product. The amount passed to this method is always negative.
The `load()` method is invoked when the dispenser is “re-stocked”. The amount that is added to the dispenser is always the order size.

`BreadBox` Class
The constructor is passed the initial inventory level (which may not be the same as the order size).
The contents must be “Bread”.
The threshold must be 4 slices.
The order quantity must be 20 slices.
The order description must be the word “Load”.

`DispenserDisplay` Class
Must display relevant information every time something is dispensed.

`RestockWarningDisplay` Class
Must display relevant information only when an order needs to be placed.

Classes for Testing

At the meeting, the team cobbled together the following interfaces/classes that will be useful when testing. They must be in the `devicesimulator` package.

```java
package devicesimulator;

/**
 * The requirements of a Display device.
 */
public interface Display
{
  /**
   * Set the text on the Display.
   *
   * @param text The text to display
   */
  public abstract void setText(String text);
}
```

```java
package devicesimulator;

import java.awt.*;
import javax.swing.*;

/**
 * A simple, poorly-designed, poorly-implemented and not-thread-safe
 * GUI component that can be used to test various classes in the
 * controller package.
 *
 * THIS CLASS MUST NEVER BE USED IN PRODUCTION.
 */
public class GUIDisplay extends JFrame implements Display
{
  private static final long serialVersionUID = 1L;
  private JTextArea textArea;

  /**
   * Construct a Display.
   *
   * @param title  The title of the Display
   */
  public GUIDisplay(String title)
  {
    super("KitchIntel");
    JPanel contentPane = (JPanel)getContentPane();
    contentPane.setLayout(new BorderLayout());
    contentPane.add(new JLabel(title), BorderLayout.NORTH);
    textArea = new JTextArea();
    textArea.setWrapStyleWord(true);
    textArea.setLineWrap(true);
    JScrollPane scrollPane = new JScrollPane(textArea);
    contentPane.add(scrollPane, BorderLayout.CENTER);
    setSize(400,400);
    setVisible(true);
  }

  /**
   * Set the text on the Display.
   *
   * @param text  The text to display.
   */
  @Override
  public void setText(String text)
  {
    textArea.setText(text);
  }
}
```

```java
package devicesimulator;

/**
 * A realization of the Display interface that uses the terminal.
 * It is not useful for manual integration testing because it is hard
 * to see which messages came from which controllers. However, it is useful for
 * automated unit testing.controller package.
 *
 * THIS CLASS MUST NEVER BE USED IN PRODUCTION.
 */
public class CLIDisplay implements Display
{
  /**
   * Set the text on the Display.
   *
   * @param text  The text to display.
   */
  @Override
  public void setText(String text)
  {
    System.out.println(text);
  }
}
```