CIS 162 Project 2
Self-Pay Kiosk
Section 04 (Kurmas)

Due Date

- at the start of lab on 8 October

Project Summary

Create a class to simulate the functionality of a self-serve kiosk at a grocery store. You can do simple things like scan items, scan coupons, make payments, and report daily sales. Text messages are displayed on the screen for customer feedback.

Before Starting the Project

- Read sections 5.1 – 5.2
- Read this entire project description before starting

Learning Objectives

After completing this project you should be able to:

- *write* methods to meet specific requirements
- *write* conditional statements with boolean expressions
- *write* loops to solve problems
- *explain* the differences between local variables, instance variables (class fields) and method parameters

Class Specification

Create a class with the following class fields and methods. Do not create additional methods or make any changes to the following requirements. All method names must match exactly with these instructions for your code to pass the automated testing.

Class Name: `SelfPayKiosk`

Class Fields

A class usually contains several class fields (section 3.1). Some instance members are not expected to change after given an initial value. It is good practice to define these as *final* and use ALL CAPS for the names (section 7.1). Provide appropriate names and data types for each of the class fields:

- a double for the *amount due* that increases as the customer scans items
- a double for the *total sales* that increases throughout the day
- an integer for the *number of customers*

- a String for the *store name*
- a **final** double for the *sales tax rate* (0.06).
- Create an instance variable of type `NumberFormat` and instantiate it in the constructor. Use this object throughout all methods to properly display currency amounts.

  The `NumberFormat` class can be used to display currency numbers as text/strings. It is not explained in the book but you can find more information in the Java API. The following example may be enough for you. Feel free to Google more information if needed.

  You must `import java.util.Locale;`
  You must `import java.text.NumberFormat;`

Code fragment	output

`NumberFormat fmt = NumberFormat.getCurrencyInstance(Locale.US);`
`double amount = 3.0111111;`
`System.out.println("Cost: " + fmt.format(amount));`
	`Cost:$3.01`

Constructors
A *constructor* is a special method with the same name as the class and generally initializes the fields to appropriate starting values. Refer to sections 3.7 and 3.8.

- `public SelfPayKiosk (String name)` - initialize the instance variables to zero and sets the store name to parameter `name`. Displays a welcome message that includes the store name.

```
Welcome to Quick N Easy!
```

- `public SelfPayKiosk ()` - initialize the instance variables to zero and sets the store name to "GVSU Corner Store". Displays a welcome message that includes the store name.

Accessor Methods
It is good practice to provide *accessor* methods for most instance members (section 3.6). Some of these methods are informally called *getter* methods and allow access to the state of the object.

- `public double getTotalSales()` - return the total sales for the day.
- `public double getAmountDue()` - return the current amount due from the current customer (one simple line of code).
- `public int getNumCustomers()` - return the number of customers.
- `public void reportSales()` – display the store name, number of customers, total sales for the day **and average sales for the day**. Warning: be careful when calculating the average if no sales were made.

```
Scott's Corner Store
Number of Customers: 2
Daily Sales: $107.06
Average Sales: $53.53
```

Mutator Methods

A *mutator* method performs tasks that may modify class fields. Refer to section 3.6.

- `public void scanItem (double price)` – update the amount due only if parameter `price` is above zero. Print of one of two messages: 1) price of the item or 2) an error message. Use the `NumberFormat` object to correctly display currency values.

```
1) Price: $24.99
2) Scanning error.  Please try again.
```

- `public void checkout ()` – After the customer has scanned the last item, calculate the sales tax and add to the amount due. Display the sales tax and amount due.

```
Sales Tax: $1.20
Amount Due: $24.99
```

- `public void cancelSession ()` – simulates the customer clicking on a Cancel button. Set the amount due to zero and display an appropriate message.

```
Session cancelled
```

- `public void scanCoupon (double value)` – a valid coupon must be between $0 and $2.00. Only if valid, reduce the amount due by the parameter `value`. You <u>do not</u> need to plan for the amount due going negative (but you can if you want to). Print of one of two messages:

```
1) Credit: $1.99
2) Coupon not valid
```

- `public void resetKiosk(int id)` – An employee must provide a special ID to reset the kiosk (4567). Total sales, amount due and number of customers are reset to zero. Display one of two messages:

```
1) Kiosk reset
2) ID not valid
```

- `public void makePayment (double pmt)` – simulates customer making a payment of the amount in parameter `pmt`.
  - If the payment is sufficient: 1) add the amount due to total sales, 2) increase customer count, 3) reset amount due to zero and 4) display the next welcome message with a blank line before to visually separate transactions from one customer to the next.
  - However, if the payment is not enough, the amount due is reduced and the updated amount due is displayed. Total sales is also increased by parameter `pmt`. The customer can then make a second payment to complete the transaction.
  - Display one of four messages.

1) If customer attempts to pays a negative amount

```
Payment: ($5.00)
Credit card declined
```

2) If customer pays exact amount

```
Payment: $5.35
Thank You.  Have a nice day!

Welcome to Quick N Easy!
```

3) If customer pays too much

```
Payment: $5.00
Thank You.  Your change is $1.45


Welcome to Quick N Easy!
```

4) If customer pays too little

```
Payment: $10.00
Remaining Amount Due: $4.19
```

Simulation Methods

The following methods allow a programmer to quickly simulate multiple method calls. They are used for testing and demonstrate an ability to use loops.

- `public void simulateCustomer(int items, double price, double incr)` – Use a loop to simulate a customer who scans multiple items, checks out and makes a payment. The number of items is specified in parameter `items`. Price of the first scanned item is parameter `price`. Each subsequent item increases in price by parameter `incr`. For example, the first item might cost $5.00 and the second might be $5.50 followed by $6.00.

  This method can be written in approximately five lines by invoking several methods within a loop: `scanItem()`, `checkout()` and `makePayment()`.
- `public void simulateManyCustomers(int customers, int items)` - Use a loop to simulate multiple customers throughout the day. Number of customers is represented by parameter `customers`. The first customer purchases a certain number of items (parameter `items`). Each subsequent customer purchases one more item than the prior customer. For example, the first customer might purchase ten items. The second customer will purchase eleven items. The third customer will purchase twelve items and so on.

  All customers scan their first item at $3.00 and subsequent items increase by $0.35.

  This method should be written in approximately four lines by invoking `simulateCustomer()` with appropriate parameters within a loop.

Coding Style (10 pts)

Good programming practice includes writing elegant source code for the human reader. Follow the GVSU Java Style Guide.

Software Testing

Software developers must plan from the beginning that their solution is correct. BlueJ allows you to instantiate objects and invoke individual methods. You can carefully check each method and compare actual results with expected results. However, this gets tedious and cannot be automated.

Testing Your Class using the main() method

Another approach is to write a `main` method that calls all the other methods in a carefully designed sequence. See section 3.9 in the zyBook.

For this project, write a main method in a new class called `KioskTest` that instantiates at least two kiosks for different stores and invokes each of the methods with a variety of parameter values to test each method. Provide multiple print statements and if statements to test each method along with error messages as needed. It takes careful consideration to anticipate and test every possibility.

A brief and incomplete example is provided below. Your test method should be much longer and instantiate two kiosks for different stores.

```
public static void main(String args[]){
    int errors = 0;
    SelfPayKiosk kiosk2 = new SelfPayKiosk();

    // scan first item
    kiosk2.scanItem(10.0);
    if(kiosk2.getAmountDue() != 10.0){
        errors++;
        System.out.println("    ERROR: amount due should be 10.0");
    }

    // finish checking out
    kiosk2.checkout();
    kiosk2.makePayment(11);

    if(kiosk2.getAmountDue() != 0.0){
        errors++;
        System.out.println("    ERROR: amount due should be 0.0");
    }
    if(kiosk2.getNumCustomers() != 1){
        errors++;
        System.out.println("    ERROR: number of customers should be 1");
    }

    System.out.println("Testing Complete. Number of errors: " + errors);
}
```

Sample Output

The sample main method above will create the following output to the terminal window.

```
Welcome to GVSU Corner Store!
Price: $10.00

Sales Tax: $0.60
Amount Due: $10.60

Payment: $11.00
Thank You! Your change is $0.40
Welcome to GVSU Corner Store!

Testing Complete. Number of errors: 0
```

zyLab Testing

After your solution passes ALL of your tests, it is time to compare against our tests. Copy your code in `SelfPayKiosk.java` to the appropriate Ch 16 zyLab. You may have to repair a few errors until your solution passes all of our tests!

Grading Criteria

- Stapled cover page with your name and signed pledge (-5 pts if missing).
- Project requirements as specified above (100 pts).

Late Policy

Projects are due at the START of the class period. However, you are encouraged to complete a project even if you must turn it in late.

- The first 24 hours (-20 pts)
- Each subsequent weekday is an additional -10 pts
- Weekends and university holidays are free days.

Turn In

A professional document **is stapled** with an attractive cover page. Do not expect the lab to have a working stapler!

- Cover page - Provide a cover page that includes your name, a title, and an appropriate picture or clip art for the project
- Signed Pledge – The cover page must include the following signed pledge: "I pledge that this work is entirely mine, and mine alone (except for any code provided by my instructor). " In addition, provide names of any people you helped or received help from. Under no circumstances do you exchange code electronically. You are responsible for understanding and adhering to the School of CIS Guidelines for Academic Honesty.
- Time Card – The cover page must also include a brief statement of how much time you spent on the project. For example, "I spent 7 hours on this project from January 22-27 reading the book, designing a solution, writing code, fixing errors and putting together the printed document."
- Sample Output – a printout of the BlueJ Terminal window after running the main method that shows a variety of the printed messages. You can cut and paste into the Word document that contains your cover page.
- Source code - a printout of your elegant source code printed from BlueJ with line numbers (with your name):

  ```
  SelfPayKiosk.java
  KioskTest.java
  ```

- Demo – be prepared to demo your project on a lab computer or your laptop. Your instructor will ask you to perform a variety of tasks using BlueJ. You will also be asked to show your code passing the tests in zyLab.