CIS 162 Project 2
Video Kiosk

Due Date
- at the start of class on Friday, Feb 19

Before Starting the Project
- Read chapter 3
- Read section 4.1 - 4.4
- Read this entire project description before starting

Learning Objectives
After completing this project you should be able to:
- *write* compile and run a simple Java class
- *write* expressions using variables
- *write* methods to meet specific requirements
- *explain* the differences between local variables, instance variables and method parameters
- *write* conditional statements with Boolean expressions

Number Format
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 if necessary.

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

Project Requirements
Create a class, called *VideoKiosk*, which simulates the functionality of a self-service kiosk that rents DVDs. You can do simple things like insert money, make selections and return videos. Simple text messages are displayed on the screen for customer instructions. This limited machine only carries THREE titles but multiple copies of each.

A *rental transaction* includes the follow steps:
1. Make one or more video selections
2. Checkout
3. Insert money (until amount due has been achieved)

A *return transaction* includes the follow:
1. Return one video

User Interaction
You do not use a Scanner in this project for user interaction. Instead, the customer interacts with the Video Kiosk by selecting methods within BlueJ.

Instance Variables
A class should contain several instance variables (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 private instance variables:

- three Strings for the *video titles*
- three integers for the *current inventory* for each title
- a double for the *credit balance* that changes as the customer inserts coins
- a double for the *amount due* for the current customer
- an integer for the current number of *customer selections*, an integer for the total number of *daily transactions*
- a double for *total daily sales* that increase as more videos are rented.
- a **final** double for the *price* of one video in cents (198.0),
- a **final** double for the *sales tax rate* (1.06, i.e. 6%)
- a **final** integer for the *starting inventory* for each title (3).

Constructors (5 pts)
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 VideoKiosk ()` – this constructor sets the instance variables to appropriate starting values including three videos in stock for each title. Choose your own titles for the three videos.
- `public VideoKiosk (int units)` - alternative constructor sets the number of videos in stock for each title to `units` rather than 3.

Accessor Methods (5 pts)
An *accessor* method does not modify class fields. The names for these methods, which simply return the current value of a field, often begin with the prefix 'get'. Refer to section 3.6.

- `public double getPrice ( )` - return the price of one video.
- `public double getCredit ( )` - return the amount of money currently inserted by the customer.
- `public double getAmountDue ( )` - return the amount of money the customer owes for this transaction, including sales tax.
- `public int getTransactions ( )` - return the number of customer transactions completed during the day. A transaction is either a single video return or possibly multiple rentals for a single customer.
- `public int getNumSelections ( )` - return the number of selections for the current customer. Zero is valid if the customer has not yet made a selection.
- `public int getTotalInventory ( )` - return the total number of videos in stock. For example, there might be two of the first title, three of the second title and zero of the third title. Total inventory would be five.

- `public double getDailySales ( )` - return the total sales for the day including sales tax.
- `public int getVideoCount (int id)` – return the number of video in stock for title `id`. Valid `id` includes 1, 2 or 3. If `id` is not valid, return -1.

Mutator Methods (35 pts)
A mutator method performs tasks that may modify class fields. Refer to section 3.6.

- `public void selectVideo (int id)` - Decrement the counter for video `id`, increment the number of selections for the current customer and display an appropriate message (see below). Assume there is at least one video in stock and that `id` is valid.
- `public void insertMoney (int amount)` - add the given number of cents to the credit balance. Display one of two messages (see below): 1) the current credit and amount due if not yet enough money to complete the purchase, or 2) Thank you and amount of change. For option 2, invoke the private `finishTransaction()` described below.
- `public void checkOut ()` – calculate the amount due by multiplying the number of selections by the cost and then by the sales tax rate. Display a message for the customer (see below).
- `public void returnVideo (int id)` – increase the counter for video `id` and provide a confirmation message. Assume the `id` is valid and that the inventory can rise higher than the original number.
- `public void resetKiosk ()` – reset the daily total sales and number of transactions to zero. This method is only invoked by the technician at the end of the day.

Private Helper Methods (10 pts)
Designated as *private*, a helper method is designed to be used by other methods within the class. Good practice is to make methods private unless they need to be public. Refer to section 3.6. Several of the following methods are invoked from within the `printStatement` method and is an effective way to keep it shorter than it otherwise would be.

- `private void finishTransaction ( )` – Update total sales and number of transactions. Reset the instance variables that keep track of amount due, number of selections, current credit. Redisplay the greeting by invoking the private `displayGreeting()`. Note, this **private** method is automatically invoked after the customer has inserted enough money to complete the transaction.
- `private void displayGreeting ( )` - display a brief welcome message and prompt the customer to make a selection. See sample output below for how the messages should appear. Note, this **private** method should be invoked in a few places such as the constructors and within the private `finishTransaction()`.

Preventing Customer Errors (15 pts)
The kiosk is now functional but it is possible the customer will make various errors. Make the following improvements to prevent common customer errors.

- Modify the `insertMoney (int amount)` method to ignore all values that are inserted other than 5, 10, 25, 100 and 500. Provide a warning that the kiosk only accepts certain denominations (see below).
- Modify the `selectVideo (int id)` – only update the instance variables if the `id` is valid and there is at least one video in stock. Otherwise, display a warning message (see below).
- Modify the `returnVideo (int id)` – only update the instance variables if the `id` is valid. Otherwise, display a warning message (see below).

Advanced Features (10 pts)
The following should only be attempted after all of the other requirements have been completed.

- You will note that the monetary amounts display with extra decimal places and no dollar sign. Use the `NumberFormat` class (described above) to display all monetary figures in currency format. Create a single instance variable that is used throughout.
- `public void createDailyReport ( )` – Display a four line message showing the inventory, daily transactions and daily sales including tax (see below). Invoke `resetKiosk()` since this report is only generated at the end of the day.

Coding Style (10 pts)
Good programming practice includes writing elegant source code for the human reader. Follow the GVSU Java Style Guide.

Sample Output

The following messages are provided as additional examples. Your messages can be more creative as long as they convey the correct information.

Display Customer Greeting

```
Welcome to Your Cute Company Name
(1) Star Wars: The Force Awakens: 2 available
(2) It's a Wonderful Life: 3 available
(3) Divergent: 3 available

Please make a selection:
```

After customer makes a selection

```
Star Wars: The Force Awakens selected...
```

After customer selects one video and invokes Checkout

```
Amount due: $2.10
Please insert your money
```

After customer selects two videos and invokes Checkout

```
Amount due: $4.20
Please insert your money
```

After customer selects one video and inserts $1.00

```
Credit: $1.00

Amount due: $1.10
```

After customer attempts to insert 13 cents

```
Insert nickels, dimes, quarters, $1 or $5
```

After customer selects one video and inserts $3.00

```
Thank You. Enjoy the Show!

Your change: $0.90
```

Daily sales report after three customers each rented one video

```
Today's Sales Report
DVDs in stock: 6
Daily Transactions: 3
Daily Sales (including tax): $6.30
```

Software Testing (10 pts)

Software developers must carefully test their solution. BlueJ allows you to instantiate objects and then invoke individual methods to manipulate the fields (instance variables) within the object. 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 a main() method
Another approach is to write a `main` method that instantiates an object and then calls the various class methods in a carefully designed sequence. Refer to section 4.11.

For this project, write a main method in a new class called `KioskTest` that instantiates a kiosk 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.

This is a brief and incomplete example. **Your solution will be longer.**

```
public static void main(String [] args){
      VideoKiosk redbox = new VideoKiosk();
      redbox.selectVideo(3);
      redbox.checkOut();
      redbox.insertMoney(500);
}
```

Will result in the following output to the terminal window:

```
Welcome to Your Cute Company Name
(1) Star Wars: The Force Awakens: 3 available
(2) It's a Wonderful Life: 3 available
(3) Divergent: 3 available

Please make a selection:

Divergent selected...

Amount due: $2.10
Please insert your money

Thank You. Enjoy the Show!
Your change: $2.90

Welcome to Your Cute Company Name
(1) Star Wars: The Force Awakens: 2 available
(2) It's a Wonderful Life: 3 available
(3) Divergent: 3 available

Please make a selection:
```

JUnit Testing

JUnit is a Java library that helps to automate software testing. A JUnit test file `JunitKiosk.java` has been provided on BlackBoard. Follow these instructions to use the test file.

1. Name your class `VideoKiosk`. **Exact spelling is required**. Using a different name (videoKiosk, videokiosk, VidKisk, …) will cause the tests to fail.
2. Complete ALL requirements described above or some of the unit tests will fail.
3. There are multiple ways to install the Junit test file

Alternative #1

1. Create a new class in your BlueJ project that is of type **Unit Test**. Call it `JunitKiosk` and delete all of the provided code. Leave the window open while you continue with step #2.
2. Click on the `JunitKiosk.java` link provided in BlackBoard. This should open the code in a new Window.
3. Cut and paste **all** of the java source code into the newly created class in BlueJ.
4. Compile the class and it should work!

Alternative #2

1. Download `JunitKiosk.java` from Blackboard and make note where you save it. In BlueJ, select from the menu <Edit> <Add class from file…>, select the file you downloaded and press <Add>.
2. The file should compile with no errors and the class icon will be green.

Running Junit Test

After the Unit Test has compiled, right click on the green icon and select Test All. A number of automated tests will run. Those that pass will have a green check mark. Those that fail will have a gray X. Click on any of the test names for clues as to what might have gone wrong with your solution. Fix the code and try again!

Grading Criteria

There is a 50% penalty on programming projects if your solution does not compile.

- Stapled cover page with your name and signed pledge. (-5 pts if missing)
- Project requirements as specified above. (90 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 Terminal window after running the main method that shows a variety of the printed messages. You can copy and paste into the Word document that contains your cover page.
- Source code - a printout of your elegant source code printed from BlueJ (with your name). Provide `VideoKiosk` and `KioskTest`.