CSC 232: Object-Oriented Software Development
Homework #1: A VendingMachine Class
Due: Monday, September 28th (start of class)

This homework is intended for you to re-familiarize yourself with Java and practice creating a Java class, object, and methods for a VendingMachine type. You may work on this homework with up to 1 partner (no more) but you must email me your partner's name by Monday (September 21st) – be sure to CC your partner on the email that you send. If you do not send me an email but work with someone and both of you submit the same code, unfortunately I will have to treat it as an Academic Dishonesty Violation and file a report to the University.

A VendingMachine Class

- Begin by creating a **VendingMachine** class in order to be able to create a VendingMachine object that mimics how a simple vending machine (shown above) works in our real world
- Each VendingMachine should store the following:
  - A count of the number of **quarters** currently in the vending machine
  - A count of the number of **dimes** currently in the vending machine
  - A count of the number of **nickels** currently in the vending machine
  - The total number of cents that have been inserted by the customer for the current item (e.g., 75 cents). Important: This should be an **integer** field – not a floating-point type.
- Your VendingMachine class should have a **default constructor** that initializes the vending machine to have 20 quarters, 35 dimes, and 10 nickels.
- Your VendingMachine class should also have a **constructor** that allows the user to initialize (specify) the number of quarters, dimes, and nickels in the vending machine. For example, I should be able to call this constructor and provide the number of quarters, dimes, and nickels that **I** would like my VendingMachine object to have.
- Add a method named **insertQuarter**( ) that does not have any formal parameters and mimics a customer adding a quarter to the vending machine by incrementing the number of quarters in the vending machine as well as updating the total number of cents that the customer has inserted into the machine for the current item
- Add a method named **insertDime**( ) that does not have any formal parameters and mimics a customer adding a dime to the vending machine by incrementing the number of dimes in the vending machine as well as updating the total number of cents that the customer has inserted into the machine for the current item
- Add a method named **insertNickel**( ) that does not have any formal parameters and mimics a customer adding a nickel to the vending machine by incrementing the number of nickels in the vending machine as well as updating the total number of cents that the customer has inserted into the machine for the current item

- Add a method named **getTotalNumberOfCents**( ) that does not have any formal parameters and that returns the total number of cents that is in the vending machine (i.e., the total number of cents based upon the number of quarters, dimes, and nickels in the machine). Note: When I say 'return' I do not mean print to the screen, your method should return a value.
- Add a method named **canDispense**( ) that takes a single integer formal parameter for the "number of cents" that we would like to determine if the VendingMachine can dispense to the customer or not. This method should return whether or not the VendingMachine can dispense the requested number of cents based upon the total number of cents that it currently has in quarters, dimes, and nickels. Note: You should call method(s) that you have implemented so far to help you write this method rather than re-implement the same logic in this method. Note: When I say 'return' I do not mean print to the console, your method should return
- Add a method named **dispenseChange**( ) that takes a single integer formal parameter for the "number of cents" (e.g., dispenseChange(**40**) represents 40 cents to dispense) that the VendingMachine should dispense to the customer. This method should **not** return a value but instead it should **print** out the number of quarters, dimes, and nickels that equals the "amount of change" requested. This method must dispense the least amount of change possible, given the number of quarters, dimes, and nickels in the machine. For example, a call to
  dispenseChange(**40**) should print "**1 quarters, 1 dimes, 1 nickels**" (if the machine has it) rather than "4 dimes". You should **not** use loops (e.g., for-loop, while-loop, etc.) to implement this method in order to receive full credit – you can use simple math operations to determine the number of quarters, dimes, and nickels that should be dispensed/printed. Your method should update the number of quarters, dimes, and nickels appropriately as dispensed. If there is not enough change in the vending machine to dispense the requested amount, then your method should print "**Out of order**"
- Add a method named **selectItem**( ) that takes a single formal parameter for the item's number (e.g., **1**, **2**, **3**, etc.) and does not return a value. Assume that the vending machine is stocked with an unlimited supply of the following items (so you do not have to worry about the item not being available):
  - Item 1 (Snickers bar) = **45 cents**
  - Item 2 (M&M candy) = **60 cents**
  - Item 3 (York peppermint) = **15 cents**
  - Item 4 (BBQ chips) = **80 cents**

  Your selectItem( ) should determine if the customer has inserted enough money into the vending machine to purchase the item number that they selected. If they have not inserted enough money, your vending machine should simply print "**Insufficient amount**". Otherwise, if the customer has inserted enough money, your vending machine should print the name of the item they selected (e.g., "BBQ chips") and dispense the correct amount of change (i.e., it should print out the number of quarters, dimes, and nickels returned). As an example of one testing scenario:

  - The customer insertQuarter( )
  - The customer insertDime( )
  - The customer insertNickel( )
  - The customer insertNickel( ) again
  - The customer selectItem(**4**)
  - The vending machine prints out "Insufficient amount" (*because there is only 45 cents in the machine and the BBQ chips cost 80 cents to purchase*)
  - The customer then selectItem(**3**)
  - The vending machine prints out "York peppermint" and "1 quarters, 1 nickels" (*provided that the vending machine has 1 quarter and 1 nickel to dispense*) and the number of quarters and nickels in the machine is updated accordingly.

Tips:

- You should use your Driver class to test your methods **thoroughly** to ensure that they are working correctly, as this is a critical skill for a programmer to know how to do
- Be sure to incorporate good object-oriented design principles ("rules of thumb") that we have discussed so far this semester
- I will not grade your Driver file – the only code that I will grade will be what you put in your VendingMachine.java class
- Start early – do not wait until a few days before the deadline to start this homework (**no extensions will be granted – see late penalty policy in syllabus**)
- If you have questions relating to what I am asking for, I am more than happy to clarify. However, I will not look at your source code nor will I give "hints" about whether you are on the right track or how to solve the problem. This is a graded assignment and I am assessing your ability to design and implement a solution.

Submission:

- You should develop your Homework 01 project in Visual Studio Code
- After you have thoroughly tested your methods to ensure they are working correctly and are ready to submit your code, you should first ensure all changes have been saved to your project's files (recall: if there is a solid circle next to the name of your file on its Visual Studio Code tab, then it has not been fully saved yet). Then, **right-click** on your project's folder in your Windows/Mac file system. If you are a a Windows user, then select **Send To ...** >> **Compressed (zipped) folder**. A new file with "zipped folder" icon should appear that has your project's same name. If you are a Mac user, then selec the **Compress ...** option and a new file with .zip extension should appear alongside your project's folder. Open up a web browser, navigate to our DePauw Moodle page. In our course's Moodle page, you should see a link for **Homework 01**. When you select this item, it will present you with an option to upload your submission. Upload the compressed (zipped) folder that you created in the step above and press Submit.