Vending Machine Project (Prorated to 40 pts) CS205

Overview

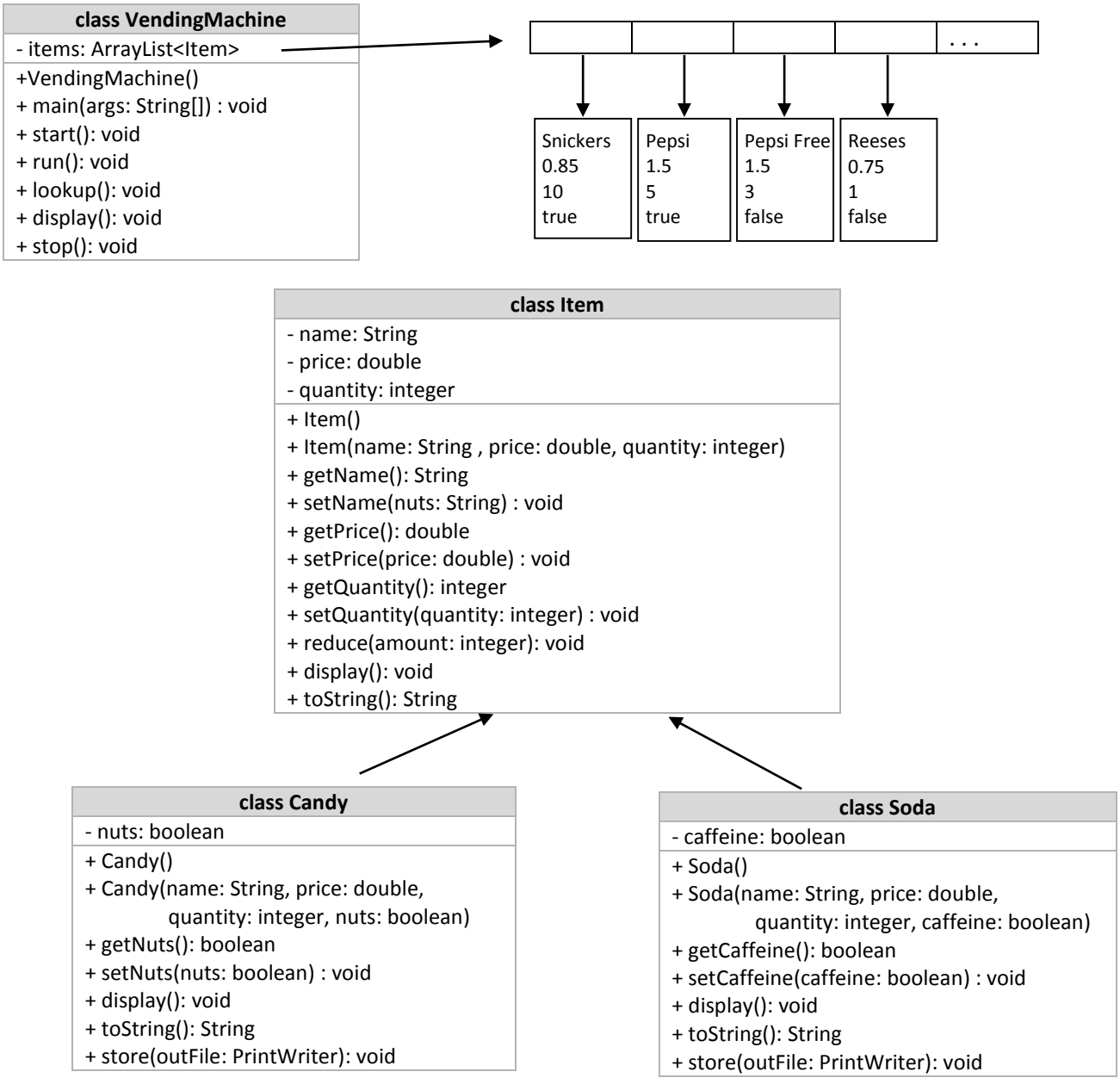
This project will utilize inheritance, polymorphism, file processing, and a dyanmic array to simulate a vending machine.

Sample Input File and Format (items.txt)

Candy,Snickers,0.85,10,true Soda,Pepsi,1.5,5,true Soda,Pepsi Free,1.5,3,false Candy,Reeses,0.75,1,false Candy,Almond Joy,0.65,6,true Candy,Rollos,0.75,8,false Soda,Mtn Dew,0.9,8,true	Candy,Item name,price,quantity,true (or false) if contains nuts or Soda,Item name,price,quantity,true (or false) if contains caffeine

items.txt (really a comma separated values file) and itemsOriginal.txt are available online. Copy itemsOriginal.txt over items.txt as needed to start with a fresh file.

Visual



Vending Machine Project (Prorated to 40 pts) CS205

Part I Requirements

Item class

- Add private data fields (instance variables) for the item's name, price, and quantity.
- **public Item()**
  - sets price and quantity to 0 and name to the empty string.
- **public Item(String name, double price, int quantity)**
  - Use the setters (mutators) to set the instance variables to the parameter data.
- Add **public accessors** and **mutators** for name, price, and quantity. Guard against a negative price or quantity by setting the data member to 0.
- **public void reduce(int amount)**
  - decrements the quantity field by amount if amount > 0 and amount <= quantity.
- **public void display()**
  - Displays the candy name, price (with $, at least one digit before decimal pt and 2 after), and quantity (followed by the word " remaining ") on the same line with comma and space separators
    Sample: Pepsi, $1.25, 7 remaining
- **public String toString()**
  - (Overrides the object class toString method)
  - Displays the candy name, price, and quantity with comma separators. There is no additional spacing or price formatting
    Sample: Pepsi,1.5,7

Candy class

- Add a private field (instance variable) for nuts.
- **public Candy()**
  - calls the no-arg constructor of the parent class and then sets nuts to false. This means, it contains no nuts by default.
- **public Candy(String name, double price, int quantity, boolean nuts)**
  - calls the parent constructor. Which parameters must be sent?
  - sets the nuts data field to the value of nuts.
- Add a **public accessor** and **mutator** for nuts.
- **public void display()**
  - (Overrides the parent's display method)
  - Calls the display method of the parent
  - Displays either " (nuts)" or " (no nuts)" depending if the nuts instance variable is true or false.
- **public String toString()**
  - (Overrides the object class toString method)
  - Creates and returns a string that is built from:
    - "Candy," plus
    - the returned result of the parent's toString() method plus
    - a comma and the value stored in nuts.

    Sample return value: "Candy,Snickers,0.85,10,true"
- **public void store(PrintWriter outFile) throws IOException**
  - outFile refers to an opened output file stream
  - There is only one line of code that will print the returned result from the toString() method to the output stream.

Soda class

- Add a private field (instance variable) for caffeine.
- **public Soda()**
  - calls the no-arg constructor of the parent class and then sets caffeine to false. This means, it contains no caffeine by default.

- **public Soda(String name, double price, int quantity, boolean caffeine)**
  - calls the parent constructor. Which parameters must be sent?
  - sets the caffeine data field to the value of caffeine.
- Add a **public accessor** and **mutator** for caffeine.
- **public void display()**
  - (Overrides the parent's display method)
  - Calls the display method of the parent.
  - Displays either " (caffeine)" or " (no caffeine)" depending if the caffeine instance variable is true or false.
- **public String toString()**
  - (Overrides the object class toString method)
  - Creates and returns a string that is built from:
    - "Soda," plus
    - The returned result of the parent's toString() method plus
    - a comma and the value stored in caffeine.

    Sample return value: "Soda,Mtn Dew,0.9,8,true"
- **public void store(PrintWriter outFile) throws IOException**
  - outFile refers to an opened output file stream
  - There is only one line of code that will println out to outFile the returned result from the toString() method.

Part II Requirements

Item class modification

Now, make the Item class into an abstract class. It will never be directly instantiated and will declare a method which its children (class Soda and Candy) must implement in order for them to be instantiated. To accomplish this:

- Change the header of the **Item** class to now be:
  ```
  public abstract class Item
  ```
- Add an abstract declaration for the store method:
  ```
  public abstract void store(PrintWriter outFile) throws IOException;
  ```

VendingMachine class

Add a Vending class and insert the following skeletal code. Then, follow the bullet points provided below the skeletal code.

```
import java.util.Scanner;
import java.util.ArrayList;
import java.io.*;
import java.text.DecimalFormat;

/**
 * @author Your Name Here
 * class VendingMachine simulates a simple Vending Machine
 * where items may be purchased, and item information is updated
 * and stored.
 */
public class VendingMachine
{
  private ArrayList<Item> items;

  public static void main(String[] args)
  {
    VendingMachine machine = new VendingMachine();
    try
    {
```

```
      machine.start("items.txt");
      machine.run();
      machine.stop("items.txt");
    }
    catch (IOException ex)
    {
      ex.printStackTrace();
    }
  }

  public VendingMachine()
  {
    // Assign a new empty array list to the items field
  }

  /**
   * Method start loads the machine with vending machine data.
   * @param filename The name of the file storing the vending
   *                 machine data.
   * @throws IOException
   */
  public void start(String filename) throws IOException
  {
    // Check if the filename exists. If not, inform the user that the
    // file does not exist and exit with an error code of 1.

    // Open the file for reading

    // While there are more lines in the file, read each vending machine
    // item from file into the items list
    while (/* Insert condition */)
    {
      // Read the next line from the file and split into an
      // array of strings

      // IF the line of data read represents a Soda, build a new
      // Soda object with the name/price/quantity/caffeine information
      // and then add it to items
      // ELSE build a new Candy object with the name/price/quantity/nuts
      // information and add it to items

    }
    // Close the file

  }

  /**
   * Method lookup performs a case-insensitive linear search to determine
   * if a vending machine item with the given name exists.
   * @param itemName The name of the vending machine item to find.
   * @return A reference to the item found or null if not found.
   */
  public Item lookup(String itemName)
  {

  }

  /**
   * Method displayItems displays each item in the machine.
```

```
 */
public void displayItems()
{

}

/**
 * Method run allows a user to purchase items from the machine.
 */
public void run()
{

  // Process transactions until the user quits
  do
  {
    displayItems();

    //Prompt for and retrieve the item into a String variable named choice


    Item item = lookup(choice);

    // If item is found, proceed with the purchase
    if (item != null)
    {
      // If there is at least one item, then process the purchase
      if (/* Insert condition */)
      {
        // Prompt for and retrieve the money

        // Loop until enough money is entered

        // Tell the user to take the product and any remaining change.

        // Reduce the quantity of the item in stock by 1

      }
      else
      {
        System.out.println("Item sold out.");
      }
    }
    else
    {
      System.out.println("Invalid item.");
    }

    // Prompt for and retrieve choice whether the user wants to quit.

  } while (/* choice does not equal an upper or lower case n */);
} // end run

/**
 * Method stop writes the items to file.
 * @param filename The name of the file to which the items are written.
 * @throws IOException
 */
public void stop(String filename) throws IOException
{
```

```
  } // end stop
} // end class
```

- If needed, be sure to realign all code that was copy/pasted.
- Method **main:** completed
- Method **VendingMachine**: Add one line of code as instructed in the comment
- Method **start:** Follow the comments
- Method **lookup**
  - Follow the comments
  - Note that this is a little different from other linear searches we've used in that an index is not returned. Instead, either a reference to an Item object in the list is returned or **null**. **null** is a keyword built into Java.
- Method **displayItems**
  - This method is only two lines of code.
  - You are required to use the "for each" version of the for loop. Inside the loop call the display method for each item.
- Method **run:** Follow the comments.
- Method **stop**
  - Open the file for writing and then use a "for each" loop to store each item. There is only one statement in the loop that calls the **store** method for each item
  - Close the file when finished
  - No error checking on the file is required.

Notes

- All output, including prompts, must be matched. Note that displayed monetary output is always formatted with a $, at least one digit before the decimal point, and 2 digits of precision.
- Give yourself plenty of time to do the project so you can iron out the details as needed.
- All coding guidelines must be followed. Especially, remember to
  - insert internal documentation.
  - insert your name as @author where needed.
  - You are not required to insert additional class or method header documentation unless otherwise directed.

Sample Runs

Run 1

```
Snickers, $0.85, 10 remaining (nuts)
Pepsi, $1.50, 5 remaining (caffeine)
Pepsi Free, $1.50, 3 remaining (no caffeine)
Reeses, $0.75, 1 remaining (no nuts)
Almond Joy, $0.65, 6 remaining (nuts)
Rollos, $0.75, 8 remaining (no nuts)
Mtn Dew, $0.90, 8 remaining (caffeine)

Item? Reeses
Enter money: $.75
Please take your Reeses.

====================
Quit? (y or n): n
====================
Snickers, $0.85, 10 remaining (nuts)
Pepsi, $1.50, 5 remaining (caffeine)
Pepsi Free, $1.50, 3 remaining (no caffeine)
```

```
Reeses, $0.75, 0 remaining (no nuts)
Almond Joy, $0.65, 6 remaining (nuts)
Rollos, $0.75, 8 remaining (no nuts)
Mtn Dew, $0.90, 8 remaining (caffeine)

Item? Snikkers
Invalid item.

====================
Quit? (y or n): n
====================
Snickers, $0.85, 10 remaining (nuts)
Pepsi, $1.50, 5 remaining (caffeine)
Pepsi Free, $1.50, 3 remaining (no caffeine)
Reeses, $0.75, 0 remaining (no nuts)
Almond Joy, $0.65, 6 remaining (nuts)
Rollos, $0.75, 8 remaining (no nuts)
Mtn Dew, $0.90, 8 remaining (caffeine)

Item? Reeses
Item sold out.

====================
Quit? (y or n): n
====================
Snickers, $0.85, 10 remaining (nuts)
Pepsi, $1.50, 5 remaining (caffeine)
Pepsi Free, $1.50, 3 remaining (no caffeine)
Reeses, $0.75, 0 remaining (no nuts)
Almond Joy, $0.65, 6 remaining (nuts)
Rollos, $0.75, 8 remaining (no nuts)
Mtn Dew, $0.90, 8 remaining (caffeine)

Item? Pepsi Free
Enter money: $1.25
Enter $0.25 more: $1
Please take your Pepsi Free.
Please take your change ($0.75).

====================
Quit? (y or n): y
====================
```

Run 2 (Note: The vending machine uses the updated data that resulted from Run 1)

```
Snickers, $0.85, 10 remaining (nuts)
Pepsi, $1.50, 5 remaining (caffeine)
Pepsi Free, $1.50, 2 remaining (no caffeine)
Reeses, $0.75, 0 remaining (no nuts)
Almond Joy, $0.65, 6 remaining (nuts)
Rollos, $0.75, 8 remaining (no nuts)
Mtn Dew, $0.90, 8 remaining (caffeine)

Item? PEPsi
Enter money: $1.65
Please take your PEPsi.
Please take your change ($0.15).

====================
Quit? (y or n): N
====================
```

```
Snickers, $0.85, 10 remaining (nuts)
Pepsi, $1.50, 5 remaining (caffeine)
Pepsi Free, $1.50, 2 remaining (no caffeine)
Reeses, $0.75, 0 remaining (no nuts)
Almond Joy, $0.65, 6 remaining (nuts)
Rollos, $0.75, 8 remaining (no nuts)
Mtn Dew, $0.90, 8 remaining (caffeine)

Item? Rollos
Enter money: $.75
Please take your Rollos.

====================
Quit? (y or n): N
====================
Snickers, $0.85, 10 remaining (nuts)
Pepsi, $1.50, 5 remaining (caffeine)
Pepsi Free, $1.50, 2 remaining (no caffeine)
Reeses, $0.75, 0 remaining (no nuts)
Almond Joy, $0.65, 6 remaining (nuts)
Rollos, $0.75, 7 remaining (no nuts)
Mtn Dew, $0.90, 8 remaining (caffeine)

Item? almond joy
Enter money: $.50
Enter $0.15 more: $.15
Please take your almond joy.

====================
Quit? (y or n): N
====================
Snickers, $0.85, 10 remaining (nuts)
Pepsi, $1.50, 5 remaining (caffeine)
Pepsi Free, $1.50, 2 remaining (no caffeine)
Reeses, $0.75, 0 remaining (no nuts)
Almond Joy, $0.65, 6 remaining (nuts)
Rollos, $0.75, 8 remaining (no nuts)
Mtn Dew, $0.90, 8 remaining (caffeine)

Item? Mtn Dew
Enter money: $.50
Enter $0.40 more: $.20
Enter $0.20 more: $.05
Enter $0.15 more: $.10
Enter $0.05 more: $.05
Please take your Mtn Dew.

====================
Quit? (y or n): Y
====================
```

Submission

- When finished with parts 1 and 2, upload all source files to D2L
- Print and staple the following files together, preferably in the following order:
    1. Item.java
    2. Candy.java
    3. Soda.java
    4. VendingMachine.java