Homework 04 - Willy Wonka's Chocolate Factory

Authors: Hussain Miyaziwala, Jack Kelly, Andrew Chafos, Vince Li, Davis Williams, Shishir Bhat

Problem Description

It is 1970 in Munich, Germany and you just found a golden ticket in a Wonka Bar. Elated for a chocolatey change of pace from your usual vocation as a chimney sweep, you grab your chaperone and head to Willy Wonka's Chocolate Factory. You and and the four other winners sign contracts before heading into the fabled factory. Unfortunately for you, 10 minutes in you get sucked into a chocolate whirlpool and eventually you find yourself unceremoniously fished out by some oompa loompas. When you regain consciousness you realize you're chained to a metal desk with only a laptop, a copy of *Head First Java Edition 2* and a pile of stale saltine crackers. Turns out that in the fine print of your contract, font size 2 to be exact, it stated that any failure to complete the tour would result in your immediate and mandatory employment in the Wonka IT department for the fair wage of 6 saltines an hour.

Your first task will be to create a class that represents a simple rectuangular chocolate bar.

Solution Description

Your solution will consist of one Java class: `Bar.java`.

This Bar class should be flexible and will be used to represent Wonka Bars of various flavors (even those that haven't been created yet) and sizes. It will also provide important methods that will print specifications of the bar to be used by other programmers and oompa loompas.

The `Bar` class should contain of the following fields:

- `private String chocolateType`. This will contain a String of any case that describes the chocolate bar's flavor. *Some* possible values include: **"milk", "Dark", "toffee", or "Cookies'n'Cream"**.
- `private int barLength`. This will contain the length of the bar as an integer.
- `private int barWidth`. This will contain the width of the bar as an integer.

The `Bar` class should contain the following methods:

- `public int getPerimeter()` that returns the the perimeter of the bar as an integer.
- `public int getArea()` that returns the area of the bar as an integer.
- `public boolean isSquare()` that returns True only if the bar of chocolate is a square.
- `public double calculateCost(double chocolateCost, double trimCost)` This method returns the cost of making this bar given the cost per unit area of chocolate and the cost of the trim that will go around the bar's perimeter.

To be specific, this method should calculate and return the following: **area * chocolateCost + perimeter * trimCost**

- `public String getDetails()` This method should return a String that describes the bar in the following format: **"`Length` by `Width` bar of `chocolateType` chocolate"**. However, if the chocolate bar is a square, this method should print the following instead: **"Square `getArea()` piece bar of `chocolateType` chocolate"**
- `public void drawBar()` This should print out a text drawing of the chocolate bar with the correct dimensions using the capitalized first letter of the `chocolateType`. For example, the following should be the console output if called on a 6 (length) by 4 (width) "milk" chocolate bar:

```
MMMM
MMMM
MMMM
MMMM
MMMM
MMMM
```

Note: The x-dimension is the width, and the y-dimension is the length.

Finally, create the following **constructor** that initializes fields `chocolateType`, `barLength`, and `barWidth` respectively:

- `public Bar(String chocolateType, int barLength, int barWidth)`

You are **required** to use the "this" keyword when referencing instance variables in your constructor.

Testing

We encourage you to write your own tests for `Bar.java`. Create a seperate Test class with a `main()` method and instantiate and inspect various instances of the Bar class.

Rubric

- [18] **Fields**
  - [18] `chocolateType`, `barLength`, `barWidth`
    - [6] Return types
    - [6] Access modifiers
    - [6] Names
- [82] **Methods**
  - [30] `getPerimeter`, `getArea`, `isSquare`
    - [6] Method signatures
    - [6] Return types
    - [6] Access modifiers
    - [12] Correct output
  - [13] `calculateCost`
    - [4] Method signature
    - [2] Return type
    - [2] Access modifier
    - [5] Correct output
  - [14] `getDetails`
    - [2] Method signature
    - [2] Return type
    - [2] Access modifier
    - [8] Correct output
  - [14] `drawBar`
    - [2] Method signatures
    - [2] Return types
    - [2] Access modifier
    - [8] Correct output
  - [11] **constructor**
    - [6] Constructor signature, access modifier
    - [2] Uses "this" keyword
    - [3] Correctly initializes fields

Allowed Imports

You may not import anything for this homework assignment.

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- var (the reserved keyword)
- System.exit

Javadocs

For this assignment, you will be commenting your code with Javadocs. Javadocs are a clean and useful way to document your code's functionality. For more information on what Javadocs are and why they are awesome, the online overview for them is extremely detailed and helpful.

You can generate the javadocs for your code using the command below, which will put all the files into a folder called javadoc:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to include are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```
import java.util.Scanner;

/**
 * This class represents a Dog object.
 * @author George P. Burdell
 * @version 1.0
 */
public class Dog {

    /**
     * Creates an awesome dog (NOT a dawg!)
     */
    public Dog() {
    ...
    }

    /**
     * This method takes in two ints and returns their sum
     * @param a first number
     * @param b second number
     * @return sum of a and b
     */
    public int add(int a, int b) {
    ...
    }

}
```

A more thorough tutorial for Javadocs can be found here.

Take note of a few things:

1. Javadocs are begun with `/**` and ended with `*/`.
2. Every class you write must be Javadoc'd and the `@author` and `@verion` tag included. The comments for a class should start with a brief description of the role of the class in your program.
3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an `@param` tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method has a non-void return type, include the `@return` tag which should have a simple description of what the method returns, semantically.

Checkstyle can check for Javadocs using the -a flag, as described in the next section. Just like Checkstyle, **one point will be deducted for each Javadoc error detected by running checkstyle**. These deductions will also be limited by the checkstyle cap, which applies to the sum of checkstyle and javadoc errors.

Checkstyle

For each of your homework assignments we will run checkstyle and deduct one point for every checkstyle error. For this homework the checkstyle cap is 15, meaning you can lose a maximum of 15 points on this assignment due to style errors. This limit will increase with each homework.

If you encounter trouble running checkstyle, check Piazza for a solution and/or ask a TA as soon as you can! You can run checkstyle on your code by using the jar file found on Canvas (under Files/Resources). To check the style of your code, place the checkstyle jar file in the same folder as your java files, then run `java -jar checkstyle-6.2.2.jar -a *.java`. You will be responsible for running checkstyle on ALL of your code.

Collaboration

Collaboration Statement

To ensure that you acknowledge collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment at the top of at least one java file that you submit**. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials.

or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Recall that comments are special lines in Java that begin with `//`.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- Bar.java

Make sure you see the message stating "HW04 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your last submission: be sure to **submit every file each time you resubmit**.

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

1. Prevent upload mistakes (e.g. forgetting checkstyle, non-compiling code)
2. Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files.
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- Check on Piazza for all official clarifications