Lab 06
Apple Maker

Objective:

Write a program that creates a class **Apple** and a tester to make sure the Apple class is crisp and delicious.

Lab Solution

Requirements:

- Functionality. (80pts)
  - No Syntax Errors. (80pts*)
    - *Code that cannot be compiled due to syntax errors is nonfunctional code and will receive no points for this entire section.
- Create a class and name it **Apple.** (1pt)
  - Do not include the main method
- Create the following Instance Variables for the class **Apple.** (9pts)
  - Type: A string that describes the apple. The type must not be null and can only be "Red Delicious", "Golden Delicious", "Gala", or "Granny Smith" and its default value is "Gala".
  - Weight: A decimal value representing the apple's weight in kilograms. The weight must be between 0kg and 2kg both inclusive, and its default value is 0.0.
  - Price: The price per apple. This must be a non-negative decimal value and its default value is 0.0.
  - Every scope must be private.
  - All above must apply for full credit.
- Create a Default Constructor for the class **Apple.** (5pts)
  - Each instance variable must be assigned a valid default value.
  - The default values for each instance variable can be found in the section "Create the following Instance Variables…".
- Create a Parameterized Constructor for the class **Apple.** (5pts)
  - Must include a parameter for each instance variable.
  - Parameters must be checked for valid values before they are assigned. Valid values can be found in the section "Create the following Instance Variables…".
- Create Accessors for each instance variable for the class **Apple**. (15pts)
  - Must follow the structure and naming conventions demonstrated in lecture.
  - All above must apply for full credit.
- Create Mutators for each instance variable for the class **Apple**. (15pts)

- Each mutator must check for valid values before assigning. If the parameter value is not correct, then the mutator must set the instance variable to a default value.
- Default and valid values can be found in the section “Create the following Instance Variables…”.
- Must follow the structure and naming conventions demonstrated in lecture.
- All above must apply for full credit.

- Create a “toString” method for the class **Apple**. (10pts)
  - Does not have parameters.
  - Must return a String formatted as,

    Type: <<apple’s name>> Weight <<apple’s weight>> Price <<apple’s price>>

  - Where apple’s name, weight, and price are the instance variable values.
  - All above must apply for full credit.
- Create an “equals” method for the class **Apple**. (10pts)
  - Must have a parameter for another Apple’s instance.
  - The method must return true or false based on if this apple’s instance variables match the other apple’s instance variables.
  - All above must apply for full credit.
- Create a class and name it **AppleTester**. (10pts)
  - Include the “main method”
  - Create (Construct) 3 different instances of Apples.
  - Demonstrate that the Default and Parameterized Constructors are working correctly.
  - Demonstrate that the Accessors and Mutators are working correctly.
  - Demonstrate that the “toString” and “equals” methods are working correctly.
  - All above must apply for full credit.
- Coding Style. (10pts)
  - Code functionality organized within multiple methods other than the main method, and methods organized within multiple classes where appropriate. (5pts)
  - Readable Code. (5pts)
    - Meaningful identifiers for data and methods.
    - Proper indentation that clearly identifies statements within the body of a class, a method, a branching statement, a loop statement, etc.
    - All the above must apply for full credit.
- Comments. (10pts)
  - Your name in every file. (5pts)
  - At least 5 meaningful comments in addition to your name. These must describe the function of the code it is near. (5pts)

Example Dialog:

```
Welcome to the apple tester

Creating a default apple
Printing the default apple's value
Type: Gala Weight: 0.0 Price: 0.0

Creating another apple
Setting the new apple's values to the
following, valid values
"Granny Smith 0.75 0.99"
Printing the new apple's values
Type: Granny Smith Weight: 0.75 Price: 0.99

Creating another default apple
Then setting the new apple's values to the
following, invalid values "iPad 2.5 -200"
Printing the newest apple's values which
should not have changed from the default
values
Type: Gala Weight: 0.0 Price: 0.0

Checking if the first and last apple have
the same values.
True
```

Solution Tests:

1. Is your name written as a comment in all source files?
2. Does the solution compile (no syntax errors)?
3. Have all the requirements for the class Apple been fulfilled?
4. Have all the requirements for the class AppleTester been fulfilled?

Lab Report

1. Create a section named "Problem" and describe this lab's problem in your own words. (10pts).
2. Create a section named "Solution Description" and describe how the code solves the problem in your own words. (10pts).

3. Create a section named “Problems Encountered” and describe the various syntax, run-time, and logic errors that were encountered while implementing the solution. (10pts).
4. In your own words describe what a class is used for.
5. In your own words describe encapsulation as it relates to object-oriented programming.
6. What is the reserved word used to create an object in memory?
7. What is the purpose of a constructor and how is it different from other methods?
8. The code snippet below seems to have an error on the Default Constructor. What is the error (or errors) and how can it be fixed?

```
public class SomeClass
{
    private int someValue;
    public SomClass()
    {
        this.someValue = 0;
    }
}
```

9. The code snippet below is an Accessor for a class, but it does not seem to work. How can this accessor be rewritten to work?

```
private int someValue;

public void getSomeValue(int aValue)
{
    return this.someValue;
}
```

10. The following code snippet keeps resulting in a “NullPointerException” run-time error. How can the code be altered to avoid this kind of error?

```
public boolean equals(AnotherClass aC)
{
    return this.someValue == aC.getSomeValue();
}
```

Finally:
Upload the source code (.JAVA File Extension) and written lab report (.DOC, .DOCX, or .PDF file extension) to the CSCE Dropbox.