Lab 3 [76 marks]

Goals:

* Use Polymorphism
* Design and implement an Interface
* Create and use Enumerations
* Create and use inner/nested classes
* Refactor code
* Read and understand documentation

Submission: [How to Submit CSD221 Labs](https://docs.google.com/document/d/1HBW_otFOBU2tWDPexTCU8VjPpkaxyYVUXZOekEDbqhc/edit?usp=sharing); you will lose marks if you do not submit correctly.

## Overview

This Lab builds on the Quizzard app you started in the previous Lab. You will add to the types of questions that the app can ask, as well as refactoring the code already written in order to make the app more robust and extendable.

Read steps below **carefully** and complete them. It may be helpful to read through the entire lab before starting work on it.

## Setup [10 marks]

* 1. [1 mark] Download the Lab 2 sample solution from LMS (the ‘Quizzard-Lab2-solution.zip’ file under the Lab 2 module) and open it in NetBeans. Take some time to familiarize yourself with the code.
     + NOTE: You must have submitted something to Lab 2 in order to do this step. If you have not submitted something yet, FINISH LAB 2! (You can still get part marks for Lab 2 even if it is submitted late.)
  2. [1 mark] Initialize a new git repository for this project inside the Lab 2 sample solution code you just downloaded. Commit all existing files.
  3. [1 mark] Create a new Project called ‘Lab3’ in your CSD221 GitLab group and push your local repository to this project. (See [Configuring Your GitLab Account for CSD221](https://docs.google.com/document/d/1-Z3TFr1yKYlPEJ9GxWie_mQNTimWOA2KuX9NH0dBT3s/edit?usp=sharing) if you forget how to do this.)
  4. Create a new text file called ‘Lab3Answers.txt’ in the src/main/resources folder of the project
  5. [2 marks] Answer the following question in Lab3Answers.txt (give your answer the heading ‘1.5’ so I can tell it’s your answer for this part of the lab): What is one **advantage** and one **disadvantage** of having the Addition3Question class extend AdditionQuestion?
  6. [2 marks] Answer the following questions in Lab3Answers.txt (with heading ‘1.6’): Why must the answer() method in the AbstractSimpleMathQuestion class be abstract? Why must there be an answer() method in the AbstractSimpleMathQuestion class at all (ie, why not just make the different question classes without the need for a class hierarchy?) Hint: consider how MathQuestionFactory’s makeQuestions method would work without a class hierarchy.
  7. [1 mark] Now download the Lab 3 starter code from LMS (the ‘Quizzard-Lab3-starter.zip’ file under the Lab 3 module) and copy its contents into your project (overwriting files is fine).
  8. Use the git diff command to inspect the differences between the Lab 2 solution and the Lab 3 starter.
     + Note that the question classes now return a Number instead of an Integer in the answer() method.
     + There is also a new promptForNumber input method in InputController that returns a BigDecimal.
  9. Read [this tutorial](https://docs.oracle.com/javase/tutorial/java/data/numberclasses.html) on the Java Number classes, and familiarize yourself with the [BigDecimal](https://docs.oracle.com/javase/8/docs/api/java/math/BigDecimal.html) class.
     + NOTE: Be sure to read carefully the documentation for the BigDecimal(double val) constructor and the difference between the equals and compareTo methods, as they will be important for this lab.
  10. [1 marks] Answer the following question in Lab3Answers.txt (with heading ‘1.10’): What is gained by having answer() return a Number instead of an Integer?
  11. [1 mark] Answer the following question in Lab3Answers.txt (with heading ‘1.11’): What is the advantage of using BigDecimal vs a class like Double?

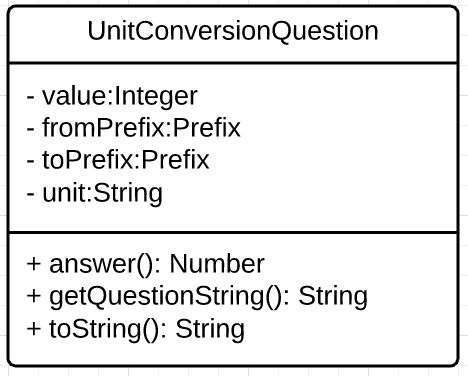
## Implement the QuestionType Enumeration [6 marks]

* 1. Examine the askForQuestionType method in InputController, and makeQuestions method in MathQuestionFactory.
  2. Currently, the different types of questions that Quizzard can ask are represented by Strings. Refactor the code so that an enum called QuestionType is used instead.
  3. [1 mark] Use appropriate enum value names to replace the current Strings.
  4. [1 mark] Place the QuestionType enum in its own file in the Quizzard.Questions package.
  5. [4 marks] When you are finished, askForQuestionType should return a QuestionType, and the switch in makeQuestions should operate on a QuestionType instead of a String.
     + NOTE: You will need to change more files than just askForQuestionType and MathQuestionFactory. Refactor the code as needed to make things work using the QuestionType enum.

## Implement ExponentQuestion [14 marks]

* 1. [1 mark] Add a new ExponentQuestion class to the Quizzard.Questions package that extends the AbstractSimpleMathQuestion class.
  2. [1 mark] ExponentQuestion should have operationSymbol ‘^’
  3. [1 mark] The answer() method should return the value of operand1 to the power of operand2
  4. [2 marks] The toString() method should operate as follows:  
      If operand2 is 2 then return operand1 followed by ‘squared’  
      If operand2 is 3 then return operand1 followed by ‘cubed’  
      Otherwise return the string ‘operand1^operand2’  
      Eg. “4 squared”, “2 cubed”, and “7^1” are all possible results, but  
      “4^2” and “2^3” should not ever be returned from toString()
  5. [2 marks] Add a new QuestionType called ‘Squaring’ to the app, and refactor the code so that the user can request this kind of question.
     + A squaring question should ask questions like ‘What is n squared?’ where n is any number from 0 to 12.
  6. [4 marks] Add a new QuestionType called ‘Exponent’ to the app, and refactor the code so that the user can request this kind of question.
     + An exponent question should ask questions like ‘What is x^y?’ where x can be any number from 0 to 12, and y is:
       - 0 to 2 if x is 4 or higher
       - 0 to 3 if x is 3
       - 0 to 4 if x is 2
       - 0 to 12 if x is 1 or 0
  7. [3 marks] Answer the following question in Lab3Answers.txt (with heading ‘3.7’): How does the current structure of the app help when new question types need to be added? (Hint: talk about what concepts are being used and how those concepts make the app easier to extend and easier to read. You should probably mention the new enum, the Question class hierarchy, and the MathQuestionFactory.)

## Implement UnitConversionQuestion [16 marks]

* 1. NOTE: it will be helpful to read through this whole section before starting so you understand how all the parts are meant to work.  
       
     For this part, you will be adding a new question class called UnitConversionQuestion that asks questions like “What is 21 cm in km?” or “What is 3 g in mg?” or “What is 93 L in kL?”. The class has the following UML:  
       
     
  2. [2 marks] Answer the following question in Lab3Answers.txt (with heading ‘4.2’): Does this class fit well into the AbstractSimpleMathQuestion class hierarchy? Why or why not?
  3. [1 mark] Add a UnitConversionQuestion class to the Quizzard.Questions package.
  4. [2 marks] The class should have a single constructor with the following signature that is used to initialize its private properties: UnitConversionQuestion(Integer value, String unit, Prefix fromPrefix, Prefix toPrefix)
  5. [1 mark] Add a **public static enum** (ie, a ‘nested’ enum) called Prefix **inside** the UnitConversionQuestion class.
  6. [1 mark] Prefix has the following values: k, UNIT, c, and m. (The unit prefixes that the question will ask the user to convert between.)
     + You will refer to these values with code like this:  
       UnitConversionQuestion.Prefix.m
  7. [3 marks] Prefix’s values have a **public BigDecimal** property called multiplier. Make sure that each Prefix value is initialized with its appropriate multiplier:
     + - k.multiplier = 1000
       - UNIT.multiplier = 1
       - c.multiplier = 0.01
       - m.multiplier = 0.001  
           
         NOTE: you will need to declare an enum property, an enum constructor, and use the initialization syntax for enum values to complete this step. See your lecture notes.  
           
         WARNING: since multiplier is a BigDecimal, be careful how you initialize these values. See the note in 1.9 above.
  8. [2 marks] The Prefix enum should override the toString method so that the default toString behaviour is used for all values **except** UNIT, in which case toString should return the empty String.
     + Eg.   
         
       System.out.println("3" + UnitConversionQuestion.Prefix.k + "g");  
       should yield the string “3kg”, and  
         
       System.out.println("2" + UnitConversionQuestion.Prefix.UNIT + "g");  
       should yield the string “2g”, NOT “2UNITg” as is the default
  9. [4 marks] Implement the toString(), getQuestionString(), and answer() methods so that they correspond with the following examples:
     + If value = 2, fromPrefix = k, toPrefix = c, unit = “m”, then
       - toString() returns “2 km in cm”
       - getQuestionString() returns “What is 2 km in cm?”
       - answer() returns 200000 as a BigDecimal
     + If value = 31, fromPrefix = UNIT, toPrefix = k, unit = ‘g’, then
       - toString() returns “31 g in kg”
       - getQuestionString() returns “What is 31 g in kg?”
       - answer() returns 0.031 as a BigDecimal
     + If value = 5, fromPrefix = k, toPrefix = UNIT, unit = ‘L’, then
       - toString() returns “5 kL in L”
       - getQuestionString() returns “What is 5 kL in L?”
       - answer() returns 5000 as a BigDecimal

## Incorporate UnitConversionQuestion [20 marks]

* 1. Analyze the UnitConversionQuestion in comparison with the AbstractSimpleMathQuestion class hierarchy. Design an interface that incorporates the common methods of these different question classes.
  2. [4 marks] Create a UML Class Diagram that shows the relationship between your new Interface, the UnitConversionQuestion, and **all** the classes in the AbstractSimpleMathQuestion class hierarchy (including your new ExponentQuestion class).
  3. [1 mark] Save the UML diagram as an IMAGE or PDF (NOT a VISIO file)
     + Name it **Lab3ClassDiagram**
     + Save it in the **src/main/resources** folder of your project
  4. [8 marks] Create your new interface inside the Quizzard.Questions package, and refactor your code to use it.
  5. [5 marks] Add a new QuestionType called ‘UnitConversion’ to your app, and refactor the code so that the user can request this type of question.
     + A UnitConversion question should ask questions like “What is 5 kL in mL?” or “What is 23 cm in km?”.
     + Values should be Integers chosen randomly from 1 to 100
     + The base unit should be chosen randomly from any reasonable set (Acceptable examples include “m”, “g”, “L”, etc.)
     + fromPrefix and toPrefix values should be chosen randomly from the UnitConversionQuestion.Prefix enum you created.
  6. [2 marks] Answer the following question in Lab3Answers.txt (with heading ‘5.6’): How does your new interface help make the Quizzard app more extendable compared with the AbstractSimpleMathQuestion class hierarchy you started with at the beginning of this lab?

## Documentation & Formatting [10 marks]

* 1. Make sure you have documented you code well using Javadoc where appropriate
  2. Use meaningful names for classes, attributes, methods, and variables that you create
  3. Add inline comments to explain non-obvious bits of code

## Submit your Lab

* 1. See [How to Submit CSD221 Labs](https://docs.google.com/document/d/1HBW_otFOBU2tWDPexTCU8VjPpkaxyYVUXZOekEDbqhc/edit?usp=sharing)
  2. You will lose marks if you do not submit correctly.

## 

## Sample Quizzard Command Line Session