**Assignment 3-- due back on March 5, 2019**

This lab is based on the material studied in weeks 5 and 6 of the course, including the exercises done in class.

This assignment is to be done individually; you are not allowed to work on this assignment with anyone. See also the [Academic Integrity page](http://www-acad.sheridanc.on.ca/~jollymor/acadDishonesty.html) and the [Assignment and Exam Policies](http://www-acad.sheridanc.on.ca/~jollymor/assignPol.html)

**Description**

In assignment 2 you created the Questions Library. In this assignment, you will create a project that programs an ArrayList that contains Question objects from the Questions Library. If you had any problems with your Assignment 2, you'll need to fix those before you can start this assignment.  Following have been included with this assignment specification:

1. Two unfinished source files (ClientApplication.java, MCQuestion.java) included in the sourceFiles folder, which you will complete following the instructions provided here and have also been included in these source files as TODOs.
2. Validator.jar file included in the jarFiles folder to be added to the project as a library to support the client application.
3. A tester file included in the testFiles folder to test the application.

***Class diagram:***

Class diagram shown in Figure 1 below must be implemented. Following steps are strongly recommend:

***Implementation—Part-I***

1. Copy your finished assignment 2 project as assignment 3 and delete Assign2tester.java file from its test package.
2. **Add QuestionType enumeration**, which is very simple and consists of 3 constants and one field. See the QuestionType enum in the class diagram.

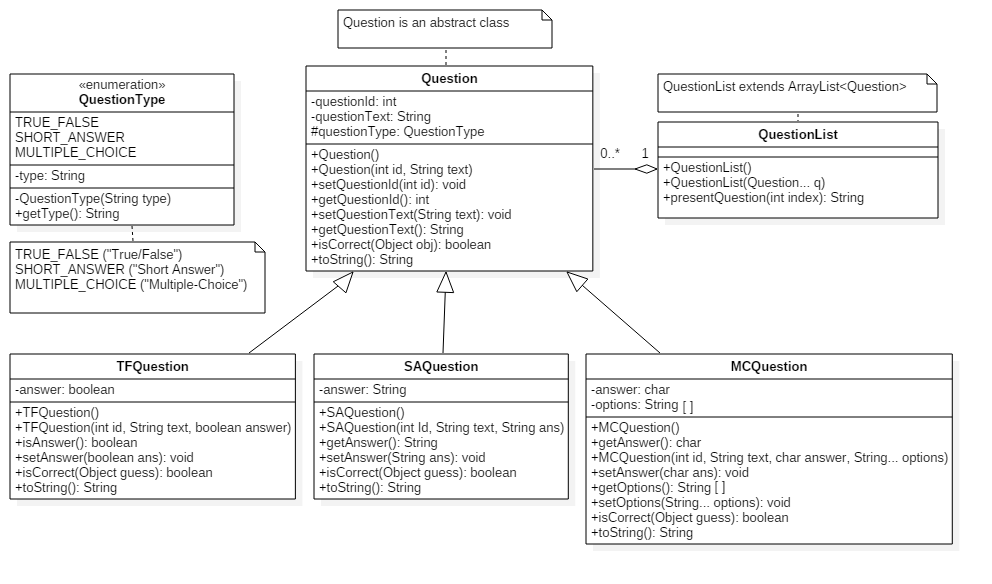


Figure 1 The new class diagram for the Questions library project

The QuestionType enum has the following members:

|  |
| --- |
| QuestionType enumeration |
| + TRUE\_FALSE ("True/False") + SHORT\_ANSWER ("Short Answer")  + MULTIPLE\_CHOICE ("Multiple-Choice") - type : String |
| - QuestionType(type : String) + getType() : String |

The enum constants have fields for the text that is displayed for each type of question. There is the normal private constructor that sets each type field value, plus the public accessor method that allows access to a QuestionType’s type field String.

1. **Add QuestionType field to Question class.** Each question will now know its own QuestionType. The parent Question class will have a new, **protected** field called questionType. Since we're using an enumeration for this, it requires no validation: it's impossible for this field to receive an invalid value. Hooray for enums.

**What's a protected field?**  
The protected modifier is used on members of a class that you want be accessible to child/sub classes and classes in the same package. In this case, we're making questionType protected so that all of the classes in the Question library can read/write it without using accessor/mutator methods, and the children of Question will still inherit it. Child classes inherit all of the public and protected members of the parent class!  
In UML, we indicate that a member is protected with the **#** symbol. See the updated class diagram up above - you'll notice the new #questionType member in the Question class.  
A protected variable or method is defined just like any other variable or method; just use the protected modifier:  
**protected double radius;**

**protected double calcArea() { ... }**

1. **Assign the appropriate QuestionType:** TFQuestion and SAQuestion inherit the protected questionType member from the Question class. When a new TFQuestion or SAQuestion object is created, we need to assign the appropriate QuestionType in the constructors (both the default constructor and the multi-param constructors. All you need to do is assign questionType one of the QuestionType enums (TRUE\_FALSE for TFQuestion and SHORT\_ANSWER for SAQuestion).
2. Copy **MCQuestion.java** to the source folder of this project. Correct its package name if required and follow the TODO list provided in the MCQuestion.java source file.
3. **Create the QuestionList Class.** The QuestionList Class is also a new class that goes in your Questions class library project.

See the QuestionList class in the class diagram up above. Note that this class extends the ArrayList class. (You could instead just make a regular QuestionList class and include an ArrayList data member, but then you'll have to add a lot more methods than the ones listed below.) The members you need to code for the QuestionList that extends ArrayList include:

* Default constructor: does nothing right now, but it needs to be there.
* Single-param constructor: accepts a variable argument list (remember that this can include a comma-delimited list of items or an array of items) of Question objects. The constructor iterates through each Question object in the list and adds it to the QuestionList. *Please note that since QuestionList is a subclass of ArrayList, ‘this’ operator can be used to refer to an empty ArrayList of Question objects.*
* presentQuestion() method: returns the display String of a specific question in the list. Make sure the index requested is valid before returning the question's String. If the index is invalid, return a null object.

The string returned by presentQuestion() will look like the Question's normal String respresentation with an "Answer: " prompt. For example (this example uses a sample MCQuestion):

**#1: Which of the following is NOT a variety of grape used for white wine?**

**a) Riesling**

**b) Malbec**

**c) Sauvingnon Blanc**

**d) Gewurztraminer**

**e) Chenin Blanc**

**Answer:**

Follow the instructions to add a Junit 4x file as demonstrated in the class and replace the auto-generated file by the tester file provided. Once it is error-free, create an updated .jar file of the project. This completes part-I of the assignment.

***Implementation—Part-II***

***Complete the QuestionsTester project***

The final step is to use your updated question library. This is a separate project and consists only of one Main class, which should be ClientApplication.java file. Make sure you add your updated Question library .jar and Validator.jar file (which is provided) to the project.

Complete the series of TODO tasks given in the ClientApplication.java file.

Once the program is working, you can test it out. Here's a sample run (I stuck in a few errors so you can see the error handling):

**Choose an Option:**

**(A)dd a Question**

**Do a (Q)uiz**

**E(x)it Program**

**> u**

**Invalid option.**

**Choose an Option:**

**(A)dd a Question**

**Do a (Q)uiz**

**E(x)it Program**

**> a**

**Choose a question type by number:**

**1. True/False**

**2. Short Answer**

**3. Multiple-Choice**

**> x**

**Invalid choice. Please try again.**

**>**

**Invalid choice. Please try again.**

**> 2**

**Question ID: 1**

**Question text: Who invented Java?**

**Answer:**

**Error: Field can't be blank.**

**Answer: James Gosling**

**Choose an Option:**

**(A)dd a Question**

**Do a (Q)uiz**

**E(x)it Program**

**> a**

**Choose a question type by number:**

**1. True/False**

**2. Short Answer**

**3. Multiple-Choice**

**> 1**

**Question ID: s**

**Invalid value: please enter only digits.**

**Question ID: 2**

**Question text: Java is Object Oriented.**

**Answer: True**

**Choose an Option:**

**(A)dd a Question**

**Do a (Q)uiz**

**E(x)it Program**

**> a**

**Choose a question type by number:**

**1. True/False**

**2. Short Answer**

**3. Multiple-Choice**

**> 3**

**Question ID: 3**

**Question text:**

**Error: Field can't be blank.**

**Question text: Which of these is not a primitive type in Java?**

**How many options does your question have? 4**

**Enter the Multiple Choice Options:**

**Option a) int**

**Option b) double**

**Option c) boolean**

**Option d) String**

**Enter letter for correct option: D**

**Choose an Option:**

**(A)dd a Question**

**Do a (Q)uiz**

**E(x)it Program**

**> q**

**#1: Who invented Java?**

**Answer: James Gosling**

**#2: Java is Object Oriented. [true or false]?**

**Answer: True**

**#3: Which of these is not a primitive type in Java?**

**a) int**

**b) double**

**c) boolean**

**d) String**

**Answer: d**

**Your score: 3/3**

**Choose an Option:**

**(A)dd a Question**

**Do a (Q)uiz**

**E(x)it Program**

**> x**

**Exiting Program...**

**Submission**

**Follow these instructions carefully!**

Your submission must follow all the submission requirements outlined in the [**Submission Standards**](http://www-acad.sheridanc.on.ca/~jollymor/prog24178/submissions.html).

It is expected that all code will conform to the industry standards outlined in [Java Standards for this Course](http://www-acad.sheridanc.on.ca/~jollymor/standards.html).

You are to submit 3 files:

***1. ZIP of NetBeans Question Class Library***

ZIP your NetBeans class library project into a file called **FirstNameLastName\_LIBa3.zip**

This must be a valid NetBeans project. No other types of projects will be accepted.

***2. The updated JAR file of your Library***

Following the instructions in the notes on [Creating Class Libraries](http://www-acad.sheridanc.on.ca/~jollymor/prog24178/library.html), build a JAR file of your questions library. The name of your JAR file must be

**FirstNameLastName\_questions.jar**

Upload the JAR file to the drop box **in addition to** your project zip file. **DO NOT add it inside your project zip file** - it must be a separate file.

***3. ZIP of the Main Class NetBeans Project***

ZIP the NetBeans project with the main ClientApplication.java program into a file called **FirstNameLastName\_MAINa3.zip**

This must be a valid NetBeans project. No other types of projects will be accepted.

When grading your submission, I will run this program with your questions .jar file.

***Upload All 3 Files***

Upload all three files individually.

Submit your assignment to the Assignment 3 drop box in SLATE. **Failure to follow any of the instructions above will result in penalties or a grade of 0.**

**Evaluation**

It is expected that all code will conform to the industry standards outlined in [Java Standards for this Course](http://www-acad.sheridanc.on.ca/~jollymor/standards.html).

Your submission will be evaluated based on the following criteria:

|  |  |  |
| --- | --- | --- |
| **Assignment 3:** | | |
| **Criteria** | **Mark** | **Comments** |
| **QuestionType Enumeration** | | |
| Constants defined correctly with String value for type description. | /1 |  |
| Constructor - correct name, private modifier, parameter for type field, sets field member to param. | /1.5 |  |
| Type field member with accessor method (correct method header, returns type field value). | /1.5 |  |
| **Total** | **/4** |  |

|  |  |  |
| --- | --- | --- |
| **QuestionType Field in Question Classes** | | |
| **Criteria** | **Mark** | **Comments** |
| Defined with proper modifier in Question | /1 |  |
| Set to correct enum value in all TFQuestion & SAQuestion constructors | /1 |  |
| **Total** | **/2** |  |

|  |  |  |
| --- | --- | --- |
| **MCQuestion Class** | | |
| **Criteria** | **Mark** | **Comments** |
| Task 1.1 & 1.2: questionType field assigned an enum constant | /1 |  |
| Task 2: setOptions() method   * Correct method header * Check for no elements in parameter (throws IAE with concise error message) * Check that no list parameter elements are empty (throws IAE with concise error message) * arrayCopy() valid list parameter to options member | /4.5 |  |
| Task 3: getOptions() method   * Correct method header * Returns copy of options array (not reference to options array - uses arraycopy()) | /2 |  |
| Task 4: isCorrect() override   * Correct method header * Checks that param is String or Character class * Compares guess parameter as char to answer member * Efficient and doesn't use unnecessary if's or comparisons * Returns correct boolean values in all branches | /2.5 |  |
| Task 5: toString() override   * Correct method header * Uses parent toString() for re-usability * Checks for null options array and adds required string to return value * Returns formatted multiple-choice question as specified in the comments/requirements | /3 |  |
|  | **/13** |  |

|  |  |  |
| --- | --- | --- |
| **QuestionList Class** | | |
| **Criteria** | **Mark** | **Comments** | |
| Class header: child of ArrayList, uses concrete Question type | /1 |  | |
| Constructors:   * Default constructor * Single-param constructor has var-arg parameter, adds each object in param to array list * Constructor method headers are correct | /2.5 |  | |
| presentQuestion() method:   * Check for valid index, returns null object if invalid * Returns string as specified in comments/requirements * Re-uses toString() | /2.5 |  | |
| **Total** | **/6** |  | |

|  |  |  |
| --- | --- | --- |
| **QuestionTester Program** | | |
| **Criteria** | **Mark** | **Comments** |
| Task 1.4: displayTypes() method:   * Iterates through enum values() * Display menu using enums as specifed in comments/requirements * Re-uses toString() | /2 |  |
| Task 1.5: returns a QuestionType enum for the user's choice | /1 |  |
| Task 2.2: getQuestion() method:   * Switch/selection to examine type; efficient * Gets answer of appropriate type from user; ensures user can't enter a blank answer * For MC: invokes getMcOptions() method to get options array before getting question answer * Constructs appropriate Question object | /4.5 |  |
| Task 2.4: create the array variable to contain the list of options | /1 |  |
| Task 2.5: gets a valid, non-empty string and add as array element | /1 |  |
| Task 2.6: return array of options | /.5 |  |
| Task 3.0: add new question object to QuestionList | /.5 |  |
| Task 4.1: serveQuiz() method:   * Check for empty list, concise error message * Iteration through QuestionList * Invoke presentQuestion() and get user's guess * Determine if guess is correct or incorrect * Keep track of correct answers * Output score in a professional way * Code is efficient | /4.5 |  |
| **Total** | **/15** |  |

|  |  |  |
| --- | --- | --- |
| **General (applies to entire assignment)** | | |
| **Criteria** | **Mark** | **Comments** |
| **Programming Style/Standards:** Code meets all the industry standards laid out in the [Submission Standards for PROG24178](http://www-acad.sheridanc.on.ca/~jollymor/submissions.html), including, but not limited to:   * Code is fully documented, including required programmer ID documentation, javadocs where appropriate (OOP only), and full internal program documentation. * Code uses proper indentation and spacing, line length, tab sizes, consistent brace style, etc. * Package name meets industry standard or the course package name is used. | /5 |  |
| **Misc.** Penalties will be given for various items, including, but not limited to:   * Submission instructions not followed (0 grade), not a valid NetBeans project (-50% of assignment value), missing components, etc. * Techniques discussed in class have been not been used, failure to meet program requirements. * Program crashes while being tested by professor (-50% of assignment value). | -- |  |
| Late Penalty: -10% per day, up to 3 days |  |  |
| **Assignment Total:** | **/45** |  |