CodeCrunch

CS2030 Practice Assessment #1 (ungraded)

Tags & Categories

Tags:

Categories: CodeCrunch

Related Tutorials

Task Content

CS2030 Practice Assessment #1

A quiz comprises questions of various forms such as multiple-choice questions (MCQ), true-false questions (TFQ), multiple-response questions (MRQ), fill-in-the-blanks, etc. After a quiz has been attempted by a student, the answers are locked-in for eventual grading (either automatically or by the examiner).

Task

In this task, we shall implement various types of quiz questions and how they are answered as well as graded.

Take note of the following

- You are NOT allowed to use any java libraries, other than those from `java.lang`, `java.util.List` and `java.util.Comparator`.
- Ensure that there are NO cyclic dependencies in your implementation.
- Write each `class`/`interface`/`enum` in its own file.
- Ensure that ALL object properties and class constants are declared `private final`.
- You are NOT allowed to use Java reflection, i.e. `Object::getClasses` and other methods from the `Class`class; in general keep to constructs that have been covered in the module.
- You are NOT allowed to use `instanceof`, other than within the same class (e.g. `obj instanceof A` within class `A`).
- You are NOT allowed to use `null` or any form of null values.
- You may assume that all tests provide valid arguments to methods; hence there is no need to validate method arguments.

You are required to complete ALL levels.

Level 1

Write an immutable class `FillInBlank` with a constructor that takes in a question of type `String`, followed by the expected answer of type `int`.

Include the `answer` method that takes in an integer as a guess. Note that the default guess is `0`.

```
jshell> FillInBlank fib = new FillInBlank("Snow white and the ? dwarfs", 7)
fib ==> Snow white and the ? dwarfs; Your answer: 0

jshell> fib.answer(7)
$.. ==> Snow white and the ? dwarfs; Your answer: 7

jshell> fib.answer(7).answer(3)
$.. ==> Snow white and the ? dwarfs; Your answer: 3

jshell> fib
fib ==> Snow white and the ? dwarfs; Your answer: 0
```

Level 2

Write an immutable MCQ class with a constructor that takes in a question of type `String`, the options as a `List<String>`, and the expected answer choice of type `int`. You may assume that there are two or more options.

Include the `answer` method that takes in an integer as a guess.

```
jshell> MCQ mcq = new MCQ("Colour of orange?", List.of("red", "green", "blue", "orange"), 4)
mcq ==> Colour of orange? [1:red][2:green][3:blue][4:orange]; Your answer: [ ? ]

jshell> mcq.answer(4)
$.. ==> Colour of orange? [1:red][2:green][3:blue][4:orange]; Your answer: [ 4:orange ]

jshell> mcq.answer(4).answer(3)
$.. ==> Colour of orange? [1:red][2:green][3:blue][4:orange]; Your answer: [ 3:blue ]

jshell> mcq.answer(4).answer(3).answer(4)
$.. ==> Colour of orange? [1:red][2:green][3:blue][4:orange]; Your answer: [ 4:orange ]

jshell> mcq
mcq ==> Colour of orange? [1:red][2:green][3:blue][4:orange]; Your answer: [ ? ]
```

Level 3

Write an immutable TFQ class for true-false questions with a constructor that takes in a question of type `String`, and the expected answer that is one of the `Strings` `"True"` or `"False"`.

Since a true-false question is just a multiple-choice question of two options, we can answer TFQs as though they are MCQs.

In addition to the `answer` method in the preceding level, include another `answer` method that takes in `"True"` or `"False"` as a guess. You may assume that either `"True"` or `"False"` will be passed to the `answer` method. There is no need to check for case-sensitivity.

```
jshell> TFQ tfq = new TFQ("An orange is blue.", "False")
tfq ==> An orange is blue. [1:True][2:False]; Your answer: [ ? ]

jshell> tfq.answer("True")
$.. ==> An orange is blue. [1:True][2:False]; Your answer: [ 1:True ]

jshell> tfq.answer("False")
$.. ==> An orange is blue. [1:True][2:False]; Your answer: [ 2:False ]

jshell> MCQ mcq = tfq
mcq ==> An orange is blue. [1:True][2:False]; Your answer: [ ? ]

jshell> mcq.answer(1)
$.. ==> An orange is blue. [1:True][2:False]; Your answer: [ 1:True ]

jshell> tfq.answer(2)
$.. ==> An orange is blue. [1:True][2:False]; Your answer: [ 2:False ]

jshell> mcq.answer("True")
|  Error:
|  incompatible types: java.lang.String cannot be converted to int
|  mcq.answer("True")
|             ^----^
```

Level 4

After a student answers a question (possibly through multiple attempts), the question is then locked for marking later. Define a `mark` method that returns `1` if the guess is the same as the answer, and `0` otherwise.

Define a `lock` method such that changes in an answer can only be accepted before locking, and marking can only proceed after locking. The same applies for all types of questions. Note that in the sample test cases below, variable `q` is declared of type `Question`.

```
jshell> Question q = new FillInBlank("Snow white and the ? dwarfs", 7)
q ==> Snow white and the ? dwarfs; Your answer: 0

jshell> q.answer(3)
$.. ==> Snow white and the ? dwarfs; Your answer: 3
```

```
jshell> q.answer(3).answer(4)
$.. ==> Snow white and the ? dwarfs; Your answer: 4

jshell> q.answer(3).answer(7).lock()
$.. ==> Snow white and the ? dwarfs; Your answer: 7

jshell> q.answer(3).answer(7).lock().answer(5)
|  Error:
|  cannot find symbol
|    symbol:   method answer(int)
|  q.answer(3).answer(7).lock().answer(5)
|  ^---------------------------------^

jshell> q.answer(3).answer(7).mark()
|  Error:
|  cannot find symbol
|    symbol:   method mark()
|  q.answer(3).answer(7).mark()
|  ^-------------------------^

jshell> q.answer(3).answer(7).lock().mark()
$.. ==> 1

jshell> q.answer(3).answer(7) instanceof FillInBlank
$.. ==> true

jshell> q.answer(3).answer(7).lock() instanceof FillInBlank
$.. ==> true

jshell> q.mark()
|  Error:
|  cannot find symbol
|    symbol:   method mark()
|  q.mark()
|  ^----^

jshell> q = new MCQ("Colour of orange?", List.of("red", "green", "blue", "orange"), 4)
q ==> Colour of orange? [1:red][2:green][3:blue][4:orange]; Your answer: [ ? ]

jshell> q.mark()
|  Error:
|  cannot find symbol
|    symbol:   method mark()
|  q.mark()
|  ^----^

jshell> q.answer(4).mark()
|  Error:
|  cannot find symbol
|    symbol:   method mark()
|  q.answer(4).mark()
|  ^--------------^

jshell> q.answer(4).lock().answer(3)
|  Error:
|  cannot find symbol
|    symbol:   method answer(int)
|  q.answer(4).lock().answer(3)
|  ^-------------------------^

jshell> q.answer(4).lock().mark()
$.. ==> 1

jshell> q.answer(4).answer(3)
$.. ==> Colour of orange? [1:red][2:green][3:blue][4:orange]; Your answer: [ 3:blue ]

jshell> q.answer(4).answer(3) instanceof MCQ
$.. ==> true

jshell> q.answer(4).answer(3).lock() instanceof MCQ
$.. ==> true

jshell> q = new TFQ("An orange is blue.", "False")
q ==> An orange is blue. [1:True][2:False]; Your answer: [ ? ]
```

```
jshell> q.mark()
|  Error:
|  cannot find symbol
|    symbol:   method mark()
|  q.mark()
|  ^----^

jshell> q.answer(1).mark()
|  Error:
|  cannot find symbol
|    symbol:   method mark()
|  q.answer(1).mark()
|  ^--------------^

jshell> q.answer(1).lock().answer(2)
|  Error:
|  cannot find symbol
|    symbol:   method answer(int)
|  q.answer(1).lock().answer(2)
|  ^----------------------^

jshell> q.answer(1).lock().mark()
$.. ==> 0

jshell> q.answer(1).answer(2)
$.. ==> An orange is blue. [1:True][2:False]; Your answer: [ 2:False ]

jshell> q.answer(1).answer(2) instanceof TFQ
$.. ==> true

jshell> q.answer(1).answer(2).lock() instanceof TFQ
$.. ==> true
```

Level 5

Fill-in-the-blank questions may require more subjective marking. We can do this by introducing a `Grader` when constructing the question.

You will need to define two graders:

- `OffByOneGrader` that constructs a grader with the expected answer, and awards two marks if the guess is exactly the same as the answer, and one mark if the guess is off-by-one. Otherwise, no marks are awarded.
- `FreeTenMarksGrader` that awards ten marks regardless of the guess.

```
jshell> Question q = new FillInBlank("Snow white and the ? dwarfs", new OffByOneGrader(7))
q ==> Snow white and the ? dwarfs; Your answer: 0

jshell> q.answer(7).lock().mark()
$.. ==> 2

jshell> q.answer(6).lock().mark()
$.. ==> 1

jshell> q.answer(8).lock().mark()
$.. ==> 1

jshell> q.answer(5).lock().mark()
$.. ==> 0

jshell> q.answer(10).lock().mark()
$.. ==> 0

jshell> Question q = new FillInBlank("? blind mice", new FreeTenMarksGrader())
q ==> ? blind mice; Your answer: 0

jshell> q.answer(3).lock().mark()
$.. ==> 10

jshell> q.mark()
|  Error:
|  cannot find symbol
|    symbol:   method mark()
|  q.mark()
```

```
|  ^----^

jshell> q.answer(0).lock().mark()
$.. ==> 10
```

Level 6

Write an immutable `MRQ` class with a constructor that takes in a question of type `String`, the options as a `List<String>`, and a `List<Integer>` of expected answers.

Include the `answer` method that takes in an integer as a guess. This method has a toggling effect, i.e. if the guess was already selected, answering it will unselect the guess. You may assume that there is at least one expected answer.

Define a `mark` method that returns `1` only if the guesses are the same as the answers. Otherwise, `0` is returned.

Note that `MRQ` is meant to be an extension to the existing implementation. Removing `MRQ` should not cause the preceding levels to fail.

```
jshell> List<String> options = List.of("apple", "banana", "car", "orange")
options ==> [apple, banana, car, orange]

jshell> List<Integer> answers = List.of(2, 1, 4)
answers ==> [2, 1, 4]

jshell> MRQ mrq = new MRQ("Pick the fruits.", options, answers)
mrq ==> Pick the fruits. [1:apple][2:banana][3:car][4:orange]; Your answer: [ ]

jshell> mrq.answer(4)
$.. ==> Pick the fruits. [1:apple][2:banana][3:car][4:orange]; Your answer: [ 4 ]

jshell> mrq.answer(2).answer(4).answer(3)
$.. ==> Pick the fruits. [1:apple][2:banana][3:car][4:orange]; Your answer: [ 2 3 4 ]

jshell> mrq.answer(4).answer(3).answer(4)
$.. ==> Pick the fruits. [1:apple][2:banana][3:car][4:orange]; Your answer: [ 3 ]

jshell> mrq
mrq ==> Pick the fruits. [1:apple][2:banana][3:car][4:orange]; Your answer: [ ]

jshell> List<String> options = List.of("apple", "banana", "car", "orange")
options ==> [apple, banana, car, orange]

jshell> List<Integer> answers = List.of(2, 1, 4)
answers ==> [2, 1, 4]

jshell> Question q = new MRQ("Pick the fruits.", options, answers)
q ==> Pick the fruits. [1:apple][2:banana][3:car][4:orange]; Your answer: [ ]

jshell> q.mark()
|  Error:
|  cannot find symbol
|    symbol:   method mark()
|  q.mark()
|  ^----^

jshell> q.answer(1).mark()
|  Error:
|  cannot find symbol
|    symbol:   method mark()
|  q.answer(1).mark()
|  ^--------------^

jshell> q.answer(1).lock().answer(2)
|  Error:
|  cannot find symbol
|    symbol:   method answer(int)
|  q.answer(1).lock().answer(2)
|  ^----------------------^

jshell> q.answer(1).lock().mark()
$.. ==> 0

jshell> q.answer(4).answer(1)
$.. ==> Pick the fruits. [1:apple][2:banana][3:car][4:orange]; Your answer: [ 1 4 ]
```

```
jshell> q.answer(4).answer(1).lock().mark()
$.. ==> 0

jshell> q.answer(4).answer(1).answer(4)
$.. ==> Pick the fruits. [1:apple][2:banana][3:car][4:orange]; Your answer: [ 1 ]

jshell> q.answer(4).answer(1).answer(2).lock().mark()
$.. ==> 1

jshell> q.answer(1) instanceof MRQ
$.. ==> true

jshell> q.answer(1).lock() instanceof MRQ
$.. ==> true
```