Software Testing and Validation
Project Report

Paulo Bolinhas 110976, Nuno Palma 86903, Rui Martins 110890

Class Exam Model Tests

Method addQuestion()

Specification

- Returns True if the question was added
- Returns False if the ExamModel is CLOSE
- Returns False if the group does not exist
- Returns False if the topic of the groupd does not match any topic of the question groups
- Returns False if the group already reached the maximum of questions

Introduction

The Category-Partition Pattern suits the method “addQuestion()” more since it does not have complex logic requiring a decision table or tree. With that said the test model used is the Black Box Testing once the implementation is not provided and the analysis depends only on the input/output results. The steps to accomplish the analysis are in the section below.

Category-Partition Pattern Analysis

1st step - Identify all functions of the MUT

- Primary Function
  - Add the question to the group with groupId and update the Group and the ExamModel
- Secondary Functions
  - Check the ExamModel state and return false if appropriate
  - Check the group information and return false if appropriate

2nd step: Identify input and output parameters of MUT

All possibilities:

Input:

- Question
- groupId
- ExamModel.state
- ExamModel.groupsList (nao colocado nas tabelas abaixo porque é usado para as choices de outro input)
- Group.topic
- Group.questionsList
- Group.maxQuestions (nao colocado nas tabelas abaixo porque é usado para as choices de outro input)

Output:

- Predicate
- Group.questionsList

3rd step: Identify categories for each input parameter

Parameter	Category
Question	Valid Invalid
GroupId	Valid Invalid
ExamModel.state	OPEN CLOSE
Group.topic	Valid Invalid
Group.questionsList	Full With Space

4th step: Partition each category into choices

Parameter	Category	Choices
Question Q	Valid question	Qy ∈ (groupId)
		Qx ∉ (groupId)
	Invalid question **[error]**	null
GroupId	Valid groupId	groupId ∈ [0,ExamModel.groupsList.size[
	Invalid groupId **[error]**	groupId ∈]-inf,0[∪ [ExamModel.groupsList.size, + inf[
ExamModel.state	OPEN	ExamModel.state = OPEN
	CLOSE **[error]**	ExamModel.state = CLOSE
GroupId.topic	Valid	topic ∈ question.topicsList
	Invalid **[error]**	topic ∉ question.topicsList
GroupId.questionsList	With Space	questionsList.size < group.maxQuestions
	Full **[error]**	questionsList.size = group.maxQuestions

5th step – Identify constraints on choices

In the table above the invalid categories were signalled with **[error]** because there is no need to test them more than one time. Regarding the other choices, no constraint was evaluated.

6th step: Generate test cases by enumerating all choices

Our choices generates a total of 10 tests. The choices signalled with **[error]** don't need to be tested more than once, therefore 6 tests where created, 2 of them for the groupId choice due to being an union of two intervals. Then, we created 4 (2x2x1x1) more tests, being 2 combinations for the two states of valid questions, another 2 to test edge values of the groupId interval and the other choices generated 1 combination each.

7th step: Develop expected values for each test case

	Input								Expected Output	
TC	Question	groupId	ExamModel state	ExamModel groupsList **g(id)**	Question topicsList	GroupId topic **t(id)**	GroupId questionsList	GroupId maxQuestions	Group questionsList	Predicate
1	null	0	OPEN	{g0}	{t1}	t1	{}	2	{}	False
2	Qx	-1	OPEN	{g0}	{t1}	null	null	null	null	False
3	Qx	2	OPEN	{g0, g1}	{t1}	null	null	null	null	False
4	Qx	0	CLOSE	{g0}	{t1}	t1	{}	2	{}	False
5	Qx	0	OPEN	{g0}	{t1}	t2	{}	2	{}	False
6	Qx	0	OPEN	{g0}	{t1}	t1	{Qa}	1	{Qa}	False
7	Qy	0	OPEN	{g0}	{t1}	t1	{Qy}	2	{Qy}	False
8	Qy	2	OPEN	{g0, g1, g2}	{t1}	t1	{Qy}	2	{Qy}	False
9	Qx	0	OPEN	{g0}	{t1}	t1	{}	2	{Qx}	True
10	Qx	3	OPEN	{g0, g1, g2, g3}	{t1}	t1	{Qa}	2	{Qa, Qx}	True

Method validate()

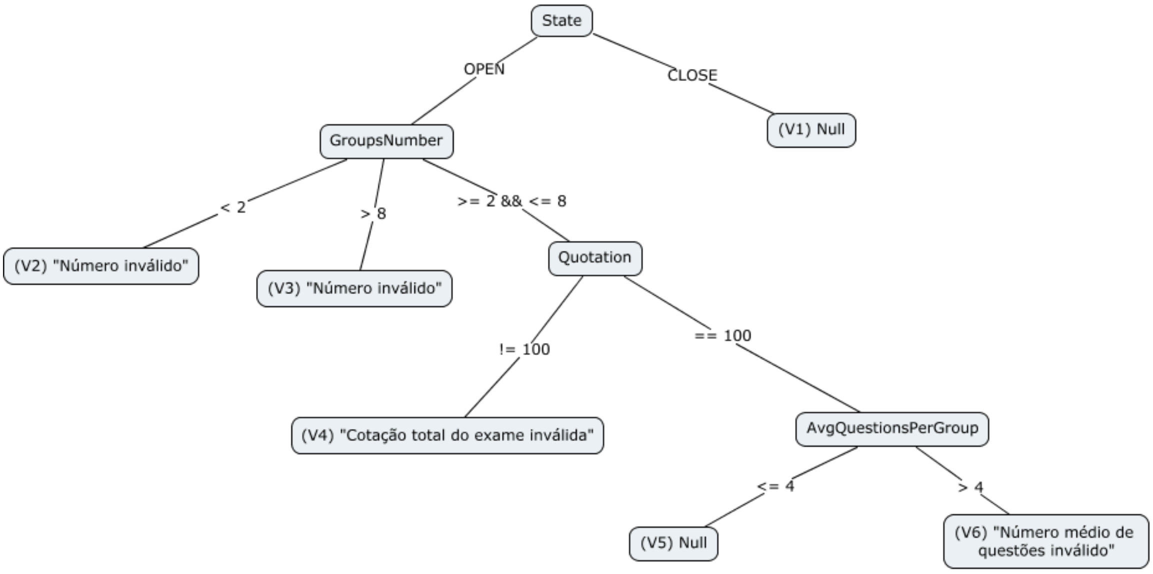
Since the method has several combinations of state and parameters complex logic the appropriate pattern to use is the Combinational Function Pattern.
The Expected Outputs defined in the legend below in order to accommodate space in the tables

Legend:

- “Null” - Null
- “Número Inválido” - NI
- “Cotação total do exame inválida” - CTEI
- “Número médio de questões inválido” - NMQI

Variables:

- State - ExamModel.State
- GroupsNumber - ExamModel.groupsList.size
- Quotation - ExamModel.quotation
- AvgQuestionPerGroup - ExamModel.avgQuestionPerGroup



Domain Matrixes

V1

State == Close

Boundary			Test Cases
Variable	**Condition**	**Type**	**1**
State	CLOSE	ON	C
		OFF	
	Typ	In	
Groups Number	Typ	In	2
Quotation	Typ	In	100
Avg Question PerGroup	Typ	In	4
Exp. Output			Null

V2

State == OPEN && GroupsNumber < 2

Boundary			Test Cases			
Variable	**Condition**	**Type**	**3**	**-**	**-**	**4**
State	OPEN	ON	O			
		OFF		C		
	Typ	In			O	O
Groups Number	< 2	ON			2	
		OFF				1
	Typ	In	0	-1		
Quotation	Typ	In	20	40	60	80
Avg Question PerGroup	Typ	In	4	5	6	7
Exp. Output			NI	Null	CTEI	NI

V3

State == OPEN && GroupsNumber > 8

Boundary			**Test Cases**			
Variable	**Condition**	**Type**	**5**	**-**	**-**	**6**
State	OPEN	ON	O			
		OFF		C		
	Typ	In			O	O
Groups Number	> 8	ON			8	
		OFF				9
	Typ	In	15	20		
Quotation	Typ	In	10	30	50	70
Avg Question PerGroup	Typ	In	4	5	6	7
Exp. Output			NI	Null	CTEI	NI

V4

State == OPEN && GroupsNumber >= 2 || GroupsNumber <= 8 && Quotation != 100

Boundary			Test Cases								
Variable	**Condition**	**Type**	**7**	**-**	**8**	**-**	**9**	**-**	**-**	**10**	**11**
State	OPEN	ON	O								
		OFF		C							
	Typ	In			O	O	O	O	O	O	O
Groups Number	>= 2	ON			2						
		OFF				1					
	<= 8	ON					8				
		OFF						9			
	Typ	In	3	4					5	6	7
Quotation	!= 100	ON							100		
		OFF								101	
		OFF									99
	Typ	In	-10	-1	0	150	300	501			
Avg Question Per Group	Typ	In	1	2	3	4	4	5	6	7	8
Exp. Output			CTEI	Null	CTEI	NI	CTEI	NI	NMQI	CTEI	CTEI

V5

State == OPEN && GroupsNumber >= 2 || GroupsNumber <= 8 && Quotation == 100 && AvgQuestionPerGroup <= 4

Boundary			**Test Cases**										
Variable	**Condition**	**Type**	**12**	**-**	**13**	**-**	**14**	**-**	**15**	**-**	**-**	**16**	**-**
State	OPEN	ON	O										
		OFF		C									
	Typ	In			O	O	O	O	O	O	O	O	O
Groups Number	>= 2	ON			2								
		OFF				1							
	<= 8	ON					8						
		OFF						9					
	Typ	In	3	4					5	6	7	4	3
Quotation	== 100	ON							100				
		OFF								101			
		OFF									99		
	Typ	In	100	100	100	100	100	100				100	100
Avg Question Per Group	<=4	ON										4	
		OFF											5
	Typ	In	0	1	2	3	4	0	1	2	3		
Exp. Output			Null	Null	Null	NI	Null	NI	Null	NI	NI	Null	NMQI

V6
State == OPEN && GroupsNumber >= 2 || GroupsNumber <= 8 && Quotation == 100 && AvgQuestionPerGroup > 4

Boundary			Test Cases										
Variable	**Condition**	**Type**	**17**	**-**	**18**	**-**	**19**	**-**	**20**	**-**	**-**	**-**	**21**
State	OPEN	ON	O										
		OFF		C									
	Typ	In			O	O	O	O	O	O	O	O	O
Groups Number	>= 2	ON			2								
		OFF				1							
	<= 8	ON					8						
		OFF						9					
	Typ	In	3	4					5	6	7	4	3
Quotation	== 100	ON							100				
		OFF								101			
		OFF									99		
	Typ	In	100	100	100	100	100	100				100	100
Avg Question Per Group	> 4	ON										4	
		OFF											5
	Typ	In	6	7	8	9	10	11	12	13	14		
Exp. Output			NMQI	Null	NMQI	NI	NMQI	NI	NMQI	NI	NI	Null	NMQI

Class Exam Manager Tests

Class Scope Testing - Modal Class Test

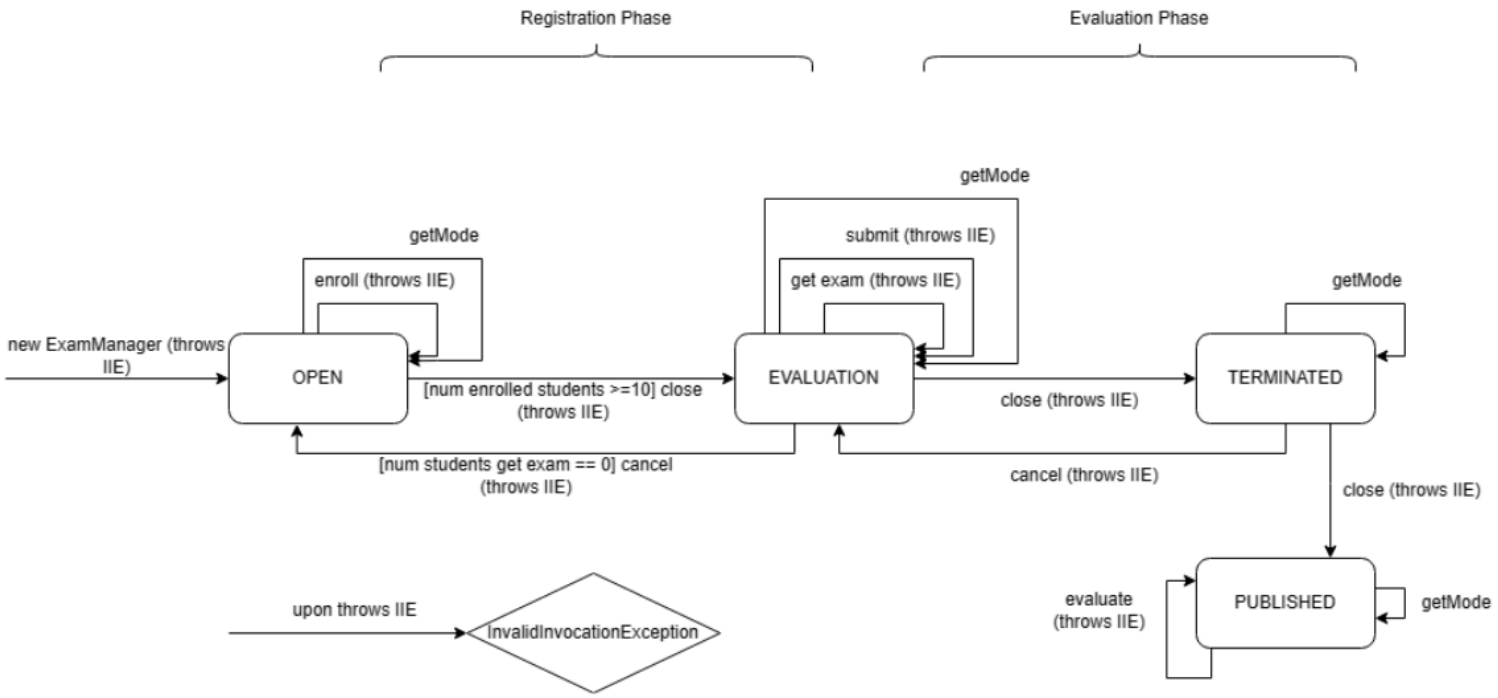
This pattern involves state-based testing, which aligns well with the behavior of ExamManager as it transitions through different modes (e.g., open, evaluation, published, terminated). State-based testing focuses on testing the behavior of a system or class based on its current state and the transitions it undergoes between states.

- The ExamManager class has fixed constraints on the sequence of messages/actions, such as enrolling students, closing exams, and publishing results.
- It transitions between different states (OPEN, EVALUATION, PUBLISHED, TERMINATED) based on specific conditions and message sequences, similar to the example of an Account object not accepting certain messages based on its balance or state.

Therefore, the ExamManager class aligns well with the Modal Class Test pattern as it involves testing the behavior of a class with fixed constraints on message sequence and interactions between message sequences and state, which are crucial aspects covered by the Modal Class Test pattern.

Modal Class Test Strategy usage

1 - Generating the state model for CUT (ExamManager):



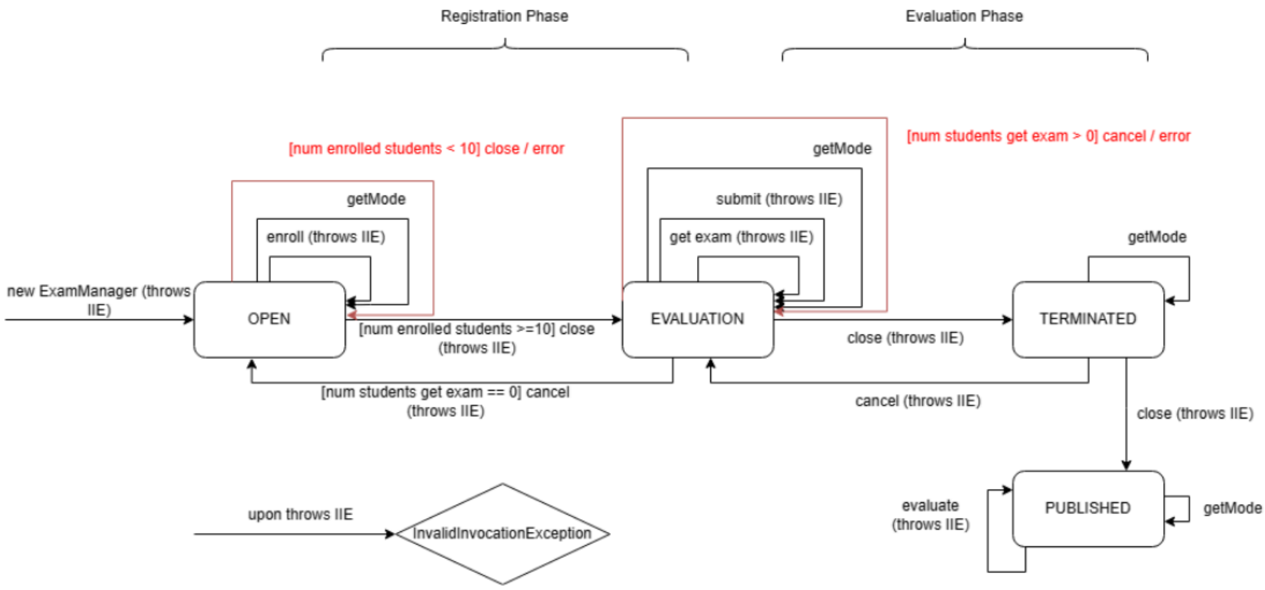
2 - Full expansion of conditional transition variants:

State	Message	Condition	Next State
OPEN	close ✗	POST: num enrolled students >=10	EVALUATION
EVALUATION	cancel ✗	PRE: num students get exam == 0	OPEN

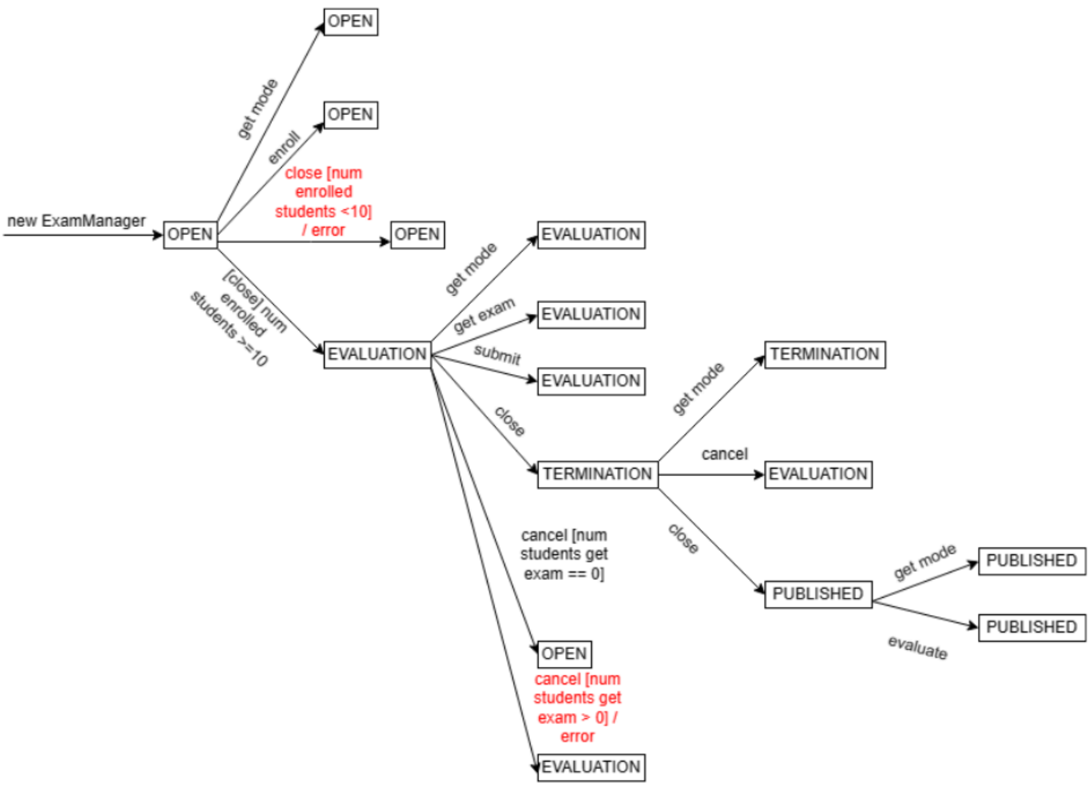
close() in OPEN generates an additional transition when ‘num enrolled students’ < 10.

cancel() in EVALUATION generates an additional transition when ‘num students get exam’ > 0.

<u>Updated state diagram with the additional transitions:</u>



3 - Generate transition tree:



4 - Generate Conformance Test Suite (Tabulate events and actions along each path to form message sequences):

Run	Test Run/Event Path	Test Run/Event Path	Test Run/Event Path	Test Run/Event Path	Test Run/Event Path	Expected Terminal State	Exception
	Level 1	*Level 2*	*Level 3*	*Level 4*	*Level 5*		
1	new					OPEN	-
2	new	get mode				OPEN	-
3	new	enroll				OPEN	-
4	new	close [num enrolled students <10] / error				OPEN	IIE
5	new	close [num enrolled students >=10]				EVALUATION	-
6	new	close [num enrolled students >=10]	cancel [num students get exam > 0] / error			EVALUATION	IIE
7	new	close [num enrolled students >=10]	get mode			EVALUATION	-
8	new	close [num enrolled students >=10]	get exam			EVALUATION	-
9	new	close [num enrolled students >=10]	submit			EVALUATION	-
10	new	close [num enrolled students >=10]	num students get exam == 0			OPEN	-
11	new	close [num enrolled students >=10]	close			TERMINATION	-
12	new	close [num enrolled students >=10]	close	get mode		TERMINATION	-
13	new	close [num enrolled students >=10]	close	cancel		EVALUATION	-
14	new	close [num enrolled students >=10]	close	close		PUBLISHED	-
15	new	close [num enrolled students >=10]	close	close	get mode	PUBLISHED	-
16	new	close [num enrolled students >=10]	close	close	evaluate	PUBLISHED	-

5 - Develop test data for each path using Invariant Boundaries pattern for events, messages, and actions:

OPEN	Condition	ON	OFF
num enrolled students	<10	10 ✓	9 (repeated)
	>=10	10 (repeated)	9 ✓

EVALUATION	Condition	ON	OFF
num students get exam	>0	0 (repeated)	1 ✓
	==0	0 ✓	1 (repeated),-1 (impossible)

7 - Develop a sneak path test suite:

Events	**States**	**States**	**States**	**States**
	OPEN	*EVALUATION*	*TERMINATION*	*PUBLISHED*
enroll	✓	PSP	PSP	PSP
close	✓	✓	✓	PSP
cancel	PSP	✓	✓	PSP
get exam	PSP	✓	PSP	PSP
submit	PSP	✓	PSP	PSP
evaluate	PSP	PSP	PSP	✓

7b - Develop a sneak path test suite:

One test case per PSP:

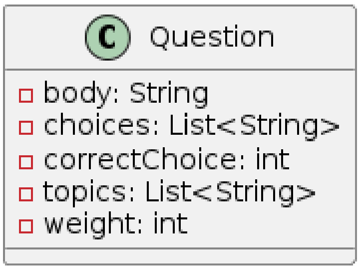
Run	Test Run/Event Path	Test Run/Event Path	Test Run/Event Path	Test Run/Event Path	Test Run/Event Path	Expected Terminal State	Exception
	Level 1	Level 2	Level 3	Level 4	Level 5		
17	new	cancel				OPEN	IIE
18	new	get exam				OPEN	IIE
19	new	submit				OPEN	IIE
20	new	evaluate				OPEN	IIE
21	new	close	enroll			EVALUATION	IIE
22	new	close	evaluate			EVALUATION	IIE
23	new	close	close	enroll		TERMINATION	IIE
24	new	close	close	get exam		TERMINATION	IIE
25	new	close	close	submit		TERMINATION	IIE
26	new	close	close	evaluate		TERMINATION	IIE
27	new	close	close	close	enroll	PUBLISHED	IIE
28	new	close	close	close	close	PUBLISHED	IIE
29	new	close	close	close	cancel	PUBLISHED	IIE
30	new	close	close	close	get exam	PUBLISHED	IIE
31	new	close	close	close	submit	PUBLISHED	IIE

Class Question Tests

Class Question

```java
public class Question {
   public Question(String body, List<String> choices, int correctChoice,
String topic, int weight) throws
           InvalidOperationException {}

   public void remove(String topic) throws InvalidOperationException {}
   public void add(String topic) throws InvalidOperationException {}
   public List<String> getTopics() {}
   public float grade(int selectedChoice) {}
   public void setWeight(int weight) throws InvalidOperationException {}
   public int getWeight(){}
   public List<String> getChoices() {}
}
```



Specification

- A question is composed of a body.
    - For reference, it is assumed that a given body must be a non-empty String (length greater or equal than 1) and can have a maximum length of up to Integer.MAX_INT which is the maximum length for a String under JVM - assuming there is enough memory.
- Weight is an integer greater than 0 and less than or equal to 15.
- Topics are represented by a string with at least 6 characters.
    - For reference, it is assumed that a given topic can have a length of up to Integer.MAX_INT which is the maximum length for a String under JVM - assuming there is enough memory.
    - A question cannot hold repeated topics.
- A question can be associated with a maximum of 5 topics. Must be associated with at least 1 topic.
- A question may have between 2 and 8 choices.

Context

- Class **Question** is composed of 3 basic data-types attributes (body, choices and weight) and 2 compound basic data-type attributes (choices and topics).
- Class **Question** imposes barely any constraints on message sequences.
  - Cannot hold repeated topics which means:
    - Cannot executed two ***add*** methods on the same primitive in a row (i.e., without ***remove*** being executed on said primitive)
    - Cannot execute one ***add*** method on a primitive that was already received on the class constructor.
- All valid combinations of value attributes can be defined - **class invariant**.

Taking the previous point into account, we can define that the class **Question** falls under a valid definition of a **non-modal modality**.

Strategy: Test procedure

The chosen test pattern: **Non-Modal Class Test**.

1a - Define the variable domain

Variable	Property under test	Type	Minimum	Maximum
body	length	String	1	N
choices	size	List<String>	2	8
correctChoice	value	int	0	choices.size() - 1
topics	size	List<String>	1	5
topic	length	String	6	N
weight	value	int	1	15

Assuming that ***correctChoice*** represents the index of the correct choice within the list of possible choices.
Assuming that ***body*** and each ***topic*** (element of ***topics***) can have a length of up to N (Integer.MAX_INT, which is the maximum length for a String under JVM - assuming there is enough memory).

1b - Find Class Invariant

```
!body.isEmpty()&& body.length() <= Integer.MAX?VALUE && 2 <= choices.size()
<= 8 && 0 <= correctChoice <= (choices.size() - 1)
       && 1 <= topics.size() <= 5 && 1 <= weight <= 15
      && {topics.get(i),…,topics.get(k)}≠
      && {topics.get(i).length(),…,topics.get(k).length()} >= 6
```

2a - Define On and Off points for the Question invariant (*Invariant Boundaries*)

Condition	On Point	Off Point
body.length() >= 1 (non-empty)	1	0
body.length() <= Integer.MAX_VALUE	Integer.MAX_VALUE	Integer.MAX_VALUE + 1
choices.size() >= 2	2	1
choices.size() <= 8	8	9
correctChoice >= 0	0	-1
correctChoice <= choices.length() - 1	choices.size() - 1	choices.size() *
topics.size() >= 1	1	0
topics.size() <= 5	5	6
topic.length() >= 6	6	5
topic.length() <= Integer.MAX_VALUE	Integer.MAX_VALUE	Integer.MAX_VALUE + 1
weight >= 1	1	0
weight <= 15	15	16
∀ topics.get(i), topcs.get(j) where i != j , topics.get(i) != topics.get(j)	TRUE	FALSE

* Ideal to catch common programmatic mistakes when dealing with indexes. Again, assuming that we have a pre-defined domain rule stating that ***correctChoice*** represents the index of the correct choice within the list of possible choices.

2b - Construct question Matrix Domain to develop set of test cases using *Invariant Boundaries*

	Rejected
	Accept

[1] ∀ topics.get(i), topcs.get(j) where i != j , topics.get(i) != topics.get(j)

T	TRUE
F	FALSE

Boundaries and Input Test Values 1-5

Variable	**Condition**	**Type**	**1**	**2**	**3**	**4**	**5**
body	body.length() >= 1 (non-empty)	On	1				
		Off		0			
	body.length() <= Integer.MAX_VALUE	On			Integer.MAX_VALUE		
		Off				Integer.MAX_VALUE + 1	
	Typical	In					20
choices	choices.size() >= 2	On					2
		Off					
	choices.size() <= 8	On					
		Off					
	Typical	In	3	3	3	3	
correctChoice	correctChoice >= 0	On					
		Off					
	correctChoice <= choices.size() - 1	On					
		Off					
	Typical	In	1	1	1	1	1
topic	topic.length() >= 6	On					
		Off					
	topic.length() <= Integer.MAX_VALUE	On					
		Off					
	Typical	In	7	7	7	7	7
weight	weight >= 1	On					
		Off					
	weight <= 15	On					
		Off					
	Typical	In	2	2	2	2	2

		On					
	topics.size() >= 1	Off					
		On					
	topics.size() <= 5	Off					
topics	Typical	In	2	2	2	2	2
		On					
	[1]	Off					
topics.get(i)	Typical	In	T	T	T	T	T
Expected Result							

Boundaries and Input Test Values 6-10

Variable	Condition	Type	6	7	8	9	10
body	body.length() >= 1 (non-empty)	On					
		Off					
	body.length() <= Integer.MAX_VALUE	On					
		Off					
	Typical	In	20	20	20	20	20
choices	choices.size() >= 2	On					
		Off	1				
	choices.size() <= 8	On		8			
		Off			9		
	Typical	In				3	3
correctChoice	correctChoice >= 0	On				0	
		Off					-1
	correctChoice <= choices.size() - 1	On					
		Off					
	Typical	In	1	1	1		
topic	topic.length() >= 6	On					
		Off					
	topic.length() <= Integer.MAX_VALUE	On					
		Off					
	Typical	In	7	7	7	7	7
weight	weight >= 1	On					
		Off					
	weight <= 15	On					
		Off					
	Typical	In	2	2	2	2	2

topics	topics.size() >= 1	On					
		Off					
	topics.size() <= 5	On					
		Off					
	Typical	In	2	2	2	2	2
topics.get(i)	[1]	On					
		Off					
	Typical	In	T	T	T	T	T
Expected Result							

Boundaries and Input Test Values 11-15

Variable	Condition	Type	11	12	13	14	15
body	body.length() >= 1 (non-empty)	On					
		Off					
	body.length() <= Integer.MAX_VALUE	On					
		Off					
	Typical	In	20	20	20	20	20
choices	choices.size() >= 2	On					
		Off					
	choices.size() <= 8	On					
		Off					
	Typical	In	3	3	3	3	3
correctChoice	correctChoice >= 0	On					
		Off					
	correctChoice <= choices.size() - 1	On	choices.size() - 1				
		Off		choices.size()			
	Typical	In			1	1	1
topic	topic.length() >= 6	On			6		
		Off				5	
	topic.length() <= Integer.MAX_VALUE	On					Integer.MAX_VALUE
		Off					
	Typical	In	7	7			
weight	weight >= 1	On					
		Off					
	weight <= 15	On					
		Off					
	Typical	In	2	2	2	2	2

	topics.size() >= 1	On					
		Off					
	topics.size() <= 5	On					
		Off					
topics	Typical	In	2	2	2	2	2
	[1]	On					
		Off					
topics.get(i)	Typical	In	T	T	T	T	T
Expected Result							

Boundaries and Input Test Values 16-20

Variable	Condition	Type	16	17	18	19	20
body	body.length() >= 1 (non-empty)	On					
		Off					
	body.length() <= Integer.MAX_VALUE	On					
		Off					
	Typical	In	20	20	20	20	20
choices	choices.size() >= 2	On					
		Off					
	choices.size() <= 8	On					
		Off					
	Typical	In	3	3	3	3	3
correctChoice	correctChoice >= 0	On					
		Off					
	correctChoice <= choices.size() - 1	On					
		Off					
	Typical	In	1	1	1	1	1
topic	topic.length() >= 6	On					
		Off					
	topic.length() <= Integer.MAX_VALUE	On					
		Off	Integer.MAX_VALUE + 1				
	Typical	In		7	7	7	7
weight	weight >= 1	On		1			
		Off			0		
	weight <= 15	On				15	
		Off					16
	Typical	In	2				

topics	topics.size() >= 1	On					
		Off					
	topics.size() <= 5	On					
		Off					
	Typical	In	2	2	2	2	2
topics.get(i)	[1]	On					
		Off					
	Typical	In	T	T	T	T	T
Expected Result							

Boundaries and Input Test Values 21-25

Variable	Condition	Type	21	22	23	24	25
	body.length() >= 1 (non-empty)	On					
		Off					
	body.length() <= Integer.MAX_VALUE	On					
		Off					
body	Typical	In	20	20	20	20	20
	choices.size() >= 2	On					
		Off					
	choices.size() <= 8	On					
		Off					
choices	Typical	In	3	3	3	3	3
	correctChoice >= 0	On					
		Off					
	correctChoice <= choices.size() - 1	On					
		Off					
correctChoice	Typical	In	1	1	1	1	1
	topic.length() >= 6	On					
		Off					
	topic.length() <= Integer.MAX_VALUE	On					
		Off					
topic	Typical	In	7	7	7	7	7
	weight >= 1	On					
		Off					
	weight <= 15	On					
		Off					
weight	Typical	In	2	2	2	2	2

	topics.size() >= 1	On	1				
		Off		0			
	topics.size() <= 5	On			5		
		Off				6	
topics	Typical	In					2
	[1]	On					
		Off					F
topics.get(i)	Typical	In	T	T	T	T	
Expected Result							

3 - Define a message sequence strategy: define-use / random

We can assume a defune-use based strategy for all the **on/off** points and a random-based strategy for the **in** points.
For example, for test **#1** we take the **on** value for the **variable body** and the **in** value for all the other variables.
For each of the test scenarios, we can choose a randomized value, for the **in** variable under test, that obeys to the domain constraints of the given variable.

Here's an example taken from the tests integrated with the TestNG framework, where we generate a random valid topic (an **in** point):

```
public static String generateRandomValidTopic() {
   int length = RANDOM.nextInt(6, 10);

   return RANDOM.ints(length, 0, CHARACTERS.length())
           .mapToObj(CHARACTERS::charAt)
           .collect(StringBuilder::new, StringBuilder::append,
StringBuilder::append)
           .toString();
}
```

4 - Set the OUT to a test case from the domain matrix

Instantiate the OUT with the given values from the domain matrix and, if needed, perform the appropriate modifications (sets) to get the object to the desired state. For example, testing that a given question cannot hold more than 5 topics.
This can be seen as the Arrange/Act portion under the Arrange-Act-Assert pattern.

5 - Send all accessor messages and verify that the returned values are consistent with the defining value

After instantiating and modifying the OUT, we have to ensure that the object is still correct and returning the appropriate values.
This can be seen as the Assert portion under the Arrange-Act-Assert pattern.

6 - Repeat steps 3 and 4 until all cases of the domain matrix have been exercised

Description of the test cases

The description of the test cases is given, implicitly, by the definition of the domain matrix: taking the given input values within the domain and the expected output for each test.
Special mention to the fact that **topics.get(i)** does not have a corresponding on point, due to the fact that this scenario is already covered by any test case that contains an IN point set to **true** - having an additional test case for this component would merely duplicate results.

Test cases - TestNG

The integration of the test cases can be found under the directory "**Software Testing and Validation - Project**". The tests that were created can be executed using **gradle**. A detailed description can be found on the project's readme.