SEC

Assignment

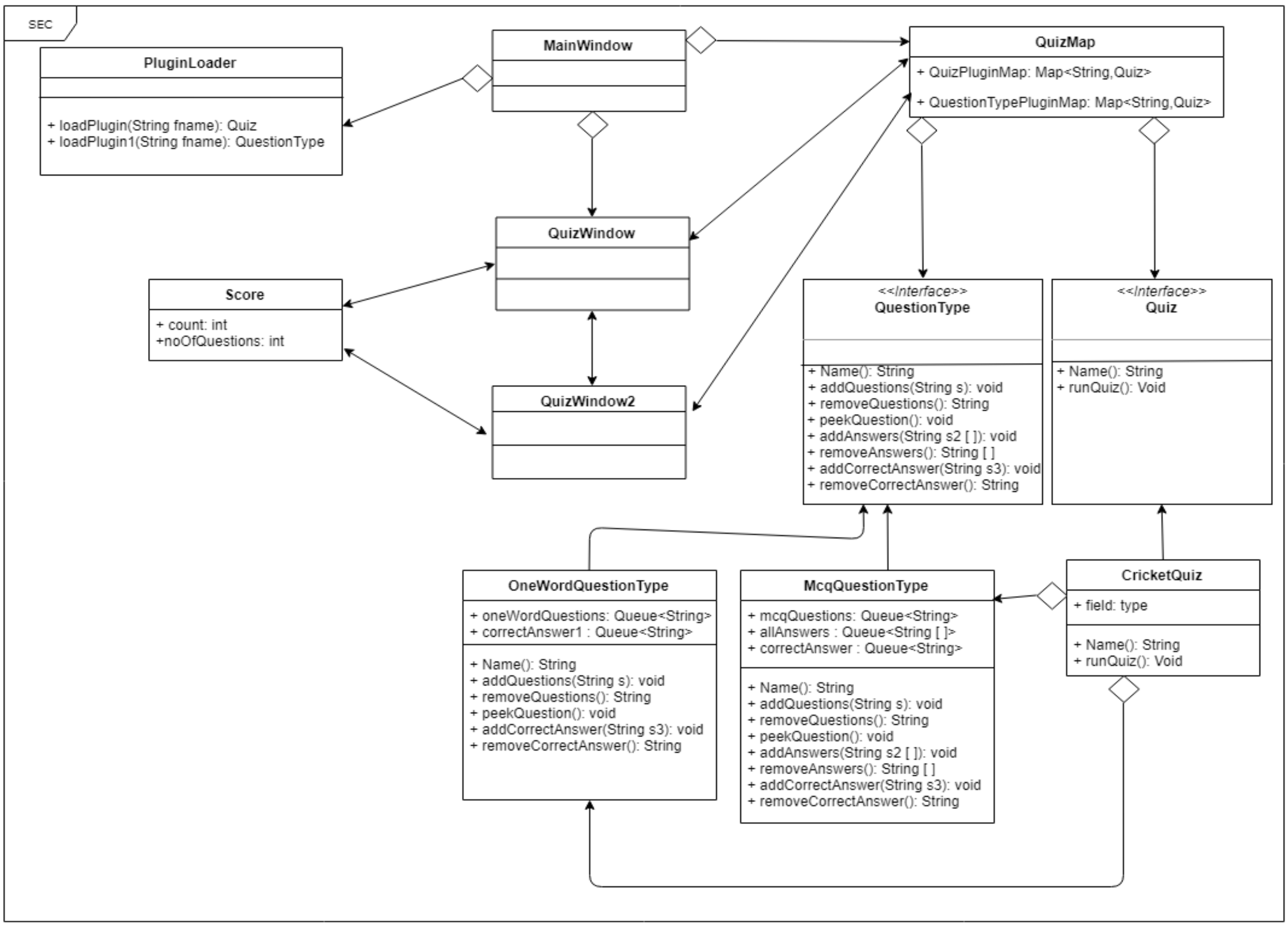
Plugin Writer's Manual

S.Hareendran

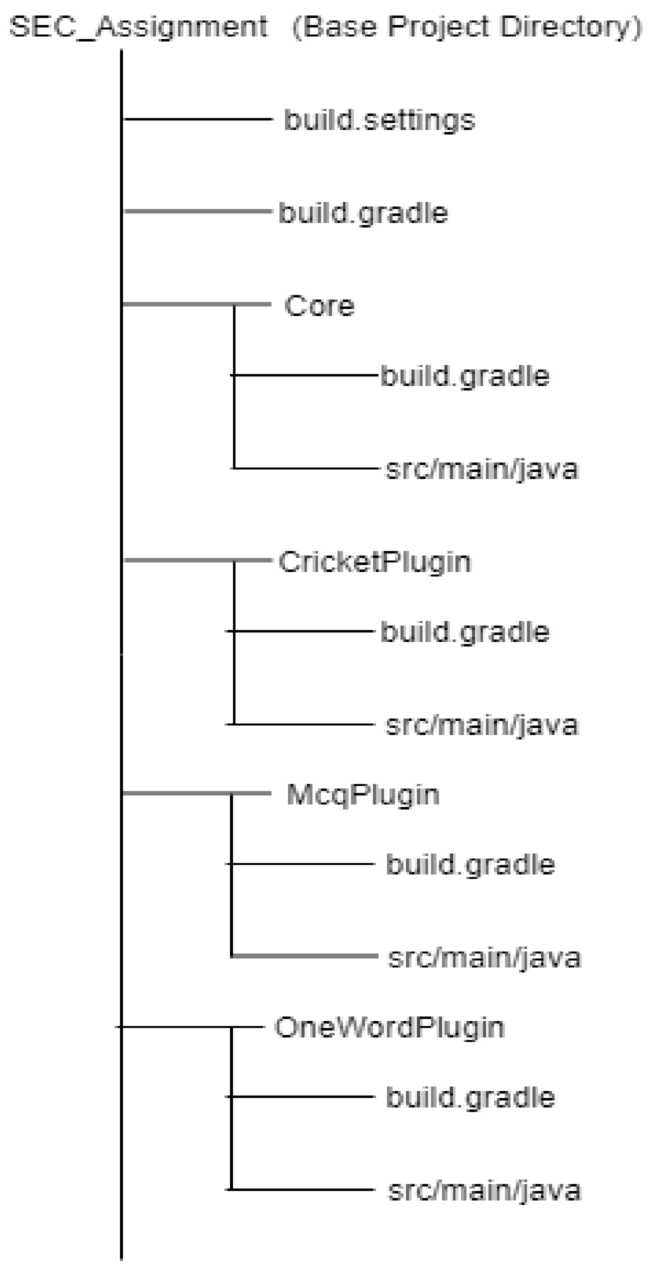
19508409

Contents

Class Diagram


SEC
PluginLoader
+ loadPlugin(String fname): Quiz
+ loadPlugin1(String fname): QuestionType
MainWindow
QuizMap
+ QuizPluginMap: Map<String,Quiz>
+ QuestionTypePluginMap: Map<String,Quiz>
QuizWindow
Score
+ count: int
+noOfQuestions: int
QuizWindow2
<<Interface>>
QuestionType
+ Name(): String
+ addQuestions(String s): void
+ removeQuestions(): String
+ peekQuestion(): void
+ addAnswers(String s2 [ ]): void
+ removeAnswers(): String [ ]
+ addCorrectAnswer(String s3): void
+ removeCorrectAnswer(): String
<<Interface>>
Quiz
+ Name(): String
+ runQuiz(): Void
OneWordQuestionType
+ oneWordQuestions: Queue<String>
+ correctAnswer1 : Queue<String>
+ Name(): String
+ addQuestions(String s): void
+ removeQuestions(): String
+ peekQuestion(): void
+ addCorrectAnswer(String s3): void
+ removeCorrectAnswer(): String
McqQuestionType
+ mcqQuestions: Queue<String>
+ allAnswers : Queue<String [ ]>
+ correctAnswer : Queue<String>
+ Name(): String
+ addQuestions(String s): void
+ removeQuestions(): String
+ peekQuestion(): void
+ addAnswers(String s2 [ ]): void
+ removeAnswers(): String [ ]
+ addCorrectAnswer(String s3): void
+ removeCorrectAnswer(): String
CricketQuiz
+ field: type
+ Name(): String
+ runQuiz(): Void

Gradle Directory Structure for the Project



Quiz Plugin

Placement of Quiz Plugin

- A quiz plugin should be placed as a subproject folder as shown in the above directory structure.

- Name of the subproject folder should be as *"*Plugin".*

  Ex : CricketPlugin

- The java file of the plugin should be placed under *"*Plugin/src/main/java".*

- Name of the java file should be *"*Quiz.java"*

  Ex : **CricketQuiz.java**

Implementation of Quiz Plugin

- Firstly the quiz plugin should implement the *"Quiz"* interface.

- Create an instance of "PluginLoader" class and store the object in a reference variable

  PluginLoader plugLoad = new PluginLoader();

- All the questions and answers related to the quiz should be placed into different fields as local variables. The questions should be in String fields, all optional answers in String arrays and correct answers in String fields.

- For example, if we consider a MCQ answer question it should be placed into String fields as shown below,

```
String question1 = "What color is a traditional cricket ball?";
String [] answer1 = {"Black" ,"Orange", "Red"};
String correctAnswer1 = “Red”;
```

- For example, if we consider a OneWord answer question it should be placed into String fields as shown below,

```
String question7 = "Which country won the World Cup final of 1999?";
String correctAnswer7 = "Australia";
```

- At the end of preparing each Question-Type questions and answers, special String should be created to denote the end of that specific Question-Type.

  - For example, at the end of preparing MCQ answer questions, to denote the end of MCQ answer questions *“MCQEND”* should be placed into a String field

    ```
    String question6 = "MCQEND";
    ```

  - For example, at the end of preparing One Word answer questions, to denote the end of One Word answer questions *“OneWordEND”* should be placed into a String field

    ```
    String question10 = "OneWordEND";
    ```

- Then should override the methods

- Override the method *"name"* by giving a name for the Quiz

```
public String name() { return "Cricket"; }
```

- Override the method *"runQuiz"*.

- Inside *"runQuiz",* need to create instances of desired Question-Type Plugins using the *"loadPlugin1"* method of *"PluginLoader"* class and store the instances into *"QuestionType"* reference variables. Relative path to the question type plugin should be given as the parameter for "loadPlgin1"

```
QuestionType mcqPlugin =
plugLoad.loadPlugin1("McqPlugin/build/classes/java/main/McqQuestionType.class");

QuestionType oneWordPlugin =
plugLoad.loadPlugin1("OneWordPlugin/build/classes/java/main/OneWordQuestionType.class");
```

- Created instances of Question-Type Plugins should be added into the static type map named *"QuestionTypePluginMap"* which is in *"mapQuiz"* class, so that the created instances of question type plugins can be accessed from any class later.

```
QuizMap.QuestionTypePluginMap.put(plugin3.name(),plugin3);
QuizMap.QuestionTypePluginMap.put(plugin4.name(),plugin4);
```

- Variable type of *"QuestionType"* should be used to hold object references of Question-Type class instances taken from the map to access methods of relevant Question-Types

```
QuestionType a1 =  QuizMap.QuestionTypePluginMap.get("MCQ");
QuestionType a2 =  QuizMap.QuestionTypePluginMap.get("OneWord");
```

- Invoke the methods of the relevant Question-Type class and add the prepared questions and answers

    - For example, 3 methods should be invoked, if a Question is type of MCQ

      a1.addQuestions(question1);

      a1.addAnswers(answer1);

      a1.addCorrectAnswer(correctAnswer1);

    - For example, 2 methods should be invoked, if a Question is type of One Word

      a2.addQuestions(question7);

      a2.addCorrectAnswer(correctAnswer7);

- The special String which was created to denote the end of specific Question-Type should also be sent as questions by invoking the *"addQuestions*" method of relevant Question-Type

Question-Type Plugin

Placement of Question-Type Plugin

- A Question-Type plugin should be placed as a subproject folder as shown in the above directory structure.

- Name of the subproject folder should be as *"*Plugin".*

  Ex : MCQPlugin

- The java file of the plugin should be placed under *"*Plugin/src/main/java".*

- Name of the java file should be *"*QuestionType.java"*

  Ex : **McqQuestionType.java**

Implementation of Question-Type Plugin

- Firstly the Question-Type plugin should implement the *"QuestionType"* interface.

- According to the Question-Type, queue data structures should be declared to store questions and answers of a quiz

```
public Queue<String> mcqQuestions = new LinkedList<>();
public Queue<String []> allAnswers = new LinkedList<>();
public Queue<String> correctAnswer = new LinkedList<>();
```

- Then should override the methods

- Override the method *“name”* by giving a name for the Question-Type

```
@Override
public String name() {return "MCQ";}
```

- *“QuestionType”* interface contains several other methods to store and access questions and answers. Should be overridden according to the Question-Type that is being created.

- For example, if we consider a MCQ Question-Type the methods should be overridden as follows

    - Method to receive questions from quiz and store to the queue

```
@Override
public void addQuestions(String s){
mcqQuestions .add(s);
}
```

    - Method to access questions

```
@Override
public String removeQuestions(){
String question= mcqQuestions.remove();
return question;
}
```

- Method to view question on top of queue

```
@Override
public String peekQuestion(){
 String peek =  mcqQuestions.peek();
return peek;
}
```

- Method to add optional answers of a question

```
@Override
public void addAnswers(String s2[]){
allAnswers.add(s2);
    }
```

- Method to access optional answers of a question

```
@Override
public String [] removeAnswers(){
String  s2 []=allAnswers.remove();
 return  s2 ;
    }
```

- Method to add optional answer of a question

```
@Override
public void addCorrectAnswer(String s3){
correctAnswer.add(s3);
    }
```

- Method to access correct answer of a question

```
@Override
public String removeCorrectAnswer(){
String correct=correctAnswer.remove();
return correct;
 }
```