Assignment 1

For this assignment, you will describe and implement release 1 of your term project. You will incorporate an abstract class, inheritance, and polymorphism. Please see the other attachments for a worked example solution.

Submit this completed Word document, observing and retaining the gray text. Your materials—in black 12-point Times New Roman—should not exceed 5 pages excluding references, figures, and appendices. Use the Appendix sections for additional material if you need to. These will be read only on an as-needed basis.

We want you to develop in Eclipse. Talk to your facilitator about exceptions. As you code, use JUnit tests—package-by-package, class-by-class, and method-by-method, except for trivial ones and ones requiring I/O. Use non-Junit classes for testing the latter. Keep the evaluation criteria in mind (listed at the end).

For this exercise, you do not need to read data from a file—you can build all of it into the code.

Include a ReadMe file describing where to run the application from, and including notes as necessary (not more). All JUnit tests will be assumed runnable.

1.1 SUMMARY DESCRIPTION

One- or two-paragraph overall description of your proposed term project. Term projects will incorporate most of the techniques discussed in the course. To incorporate them, we recognize that you may need to alter your project or even introduce an additional project. Consider giving your project an acronym.

This project is an educational tool for those looking to test and improve their music theory knowledge. The application will randomly generate questions from among several question templates ranging from simple to complex with random, appropriate values inserted at key points in the templates so that the user gets practice manipulating the questions with many different value permutations. After the user has selected his or her answer from among multiple options, the application will offer feedback on the correctness of his or her answer.

1.2 PROJECTED I/O EXAMPLE FROM PROJECTED COMPLETED PROJECT

Provide an example of projected *concrete* output for designated input. You will not be held to fulfilling exactly this—it is just explanatory at this point, to indicate where your project is going. We recognize that project direction and details will change as the term progress.

What is the 5th note of the melodic minor scale whose 7th note is the fifth overtone above C1? Enter the corresponding letter choice:

A: Eb

B: C

C: E

D: D

User: B

System:

Incorrect answer. The correct answer is A: Eb. Explanation: The fifth overtone above C1 is G. The melodic minor scale where G is the 7th is Ab melodic minor. The fifth note of the Ab melodic minor scale is Eb. The correct answer is Eb.

1.3 REQUIREMENTS IMPLEMENTED IN THIS RELEASE

High-level functional requirements statement that you accomplished for this assignment, together with input where applicable, and output.

1.3.1

The application will create simple questions based on two templates that each use the MajorScale class as the theoretical background for the question material.

1.3.2

The application will pick from among many instantiated Question subclasses at random, display the question to the user, receive the user's answer, evaluate the correctness of the answer, and display a simple message to the user about whether or not they were correct. The instantiation of objects is done with random, appropriate values.

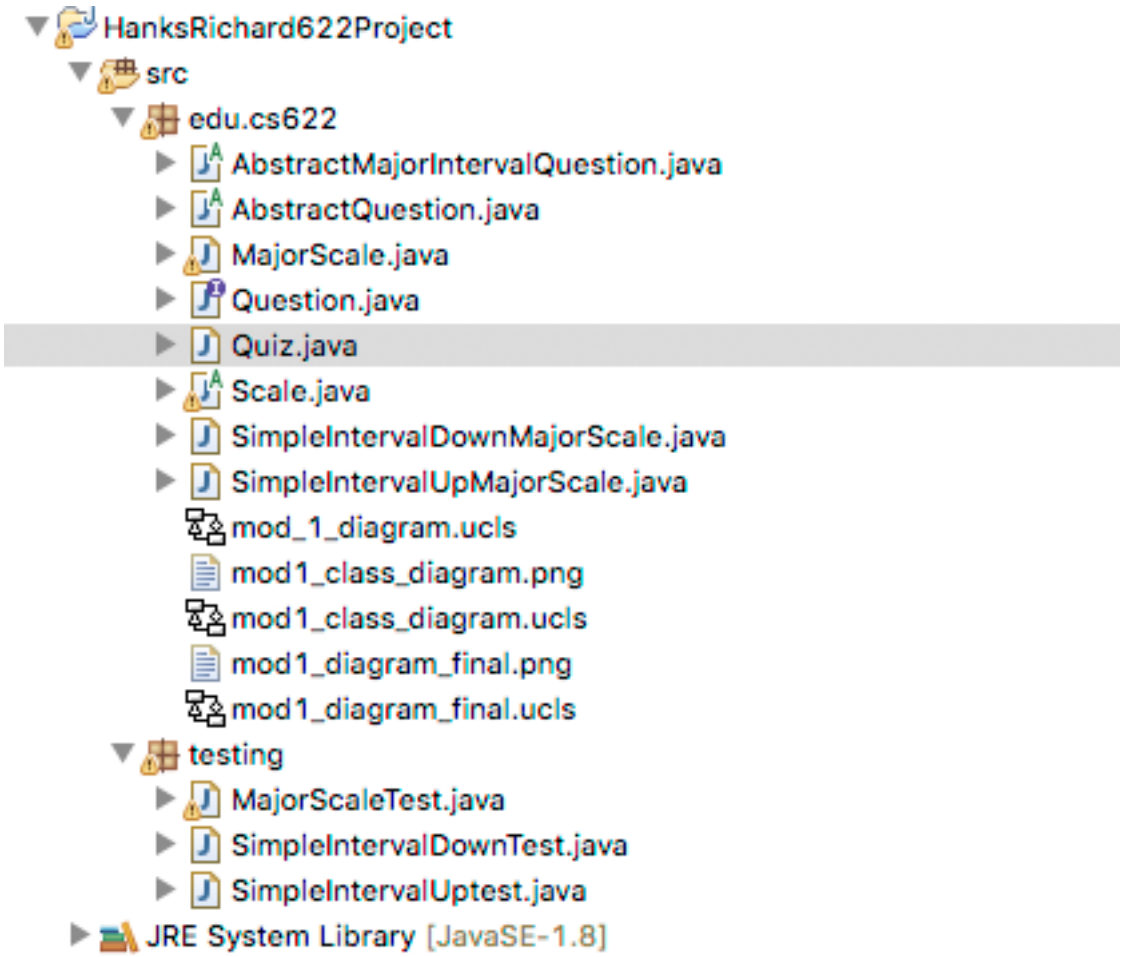
1.3.3 Illustrative Output

edu.cs622.Quiz.main() ————>

```
In the key of Eb Major, what is the note a perfect fifth above the
mediant?
Enter answer or type "Q" to exit
d
Correct!
What is the note a major 2nd below the fifth scale degree in Bb Major?
Enter answer or type "Q" to exit
q
Goodbye!
```

1.4 YOUR DIRECTORY

Show a screenshot of your directory. This should include a parallel directory of JUnit tests where possible—package-by-package, class-by-class, and method-by-method, except for trivial ones.

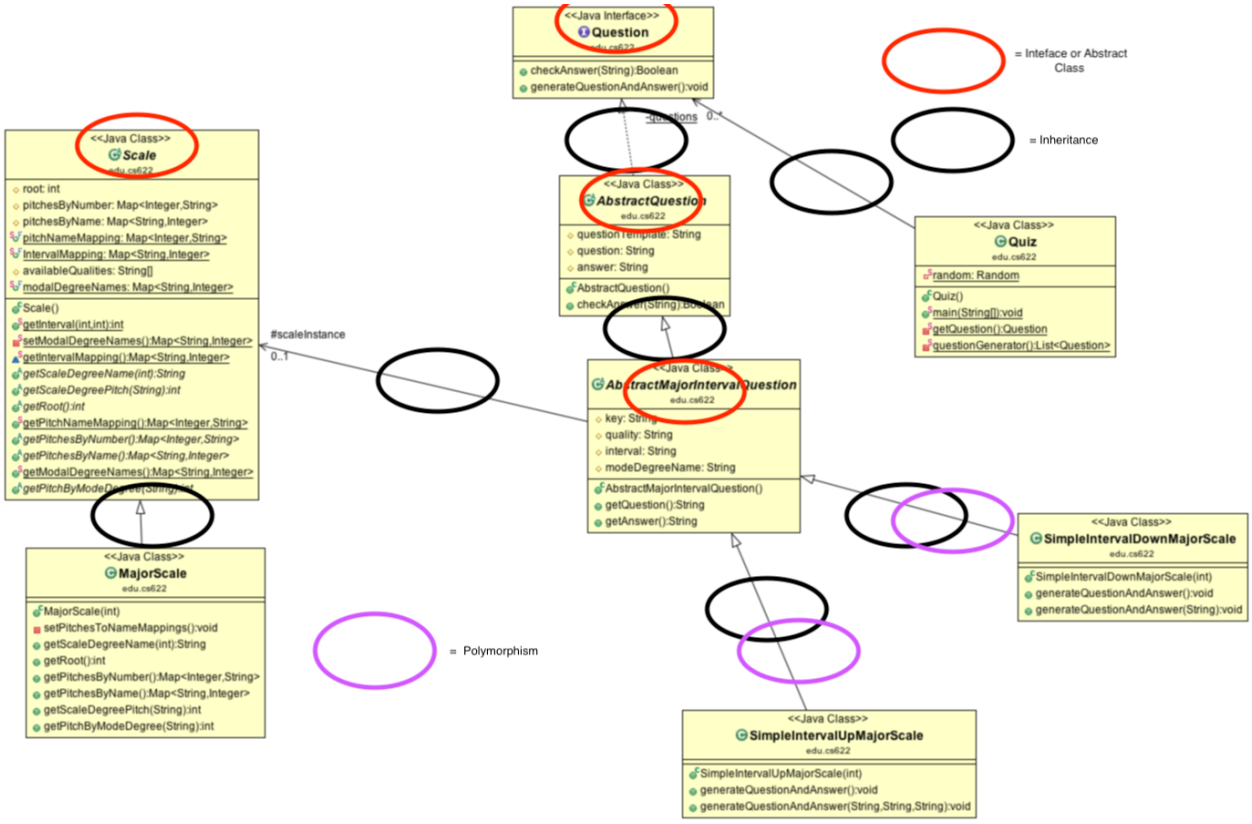


1.5 TECHNIQUES IMPLEMENTED

Your implementation should exploit *inheritance*, *polymorphism*, and *interfaces* at least once, and in a natural manner. Explain where and how you applied these, using the headings below.

1.5.1 Class model and Sequence Diagram

Cricle where you applied *inheritance*, *polymorphism*, and (*abstract classes* or *interfaces)* in your class model. Make classes and members *static* or not as per their intended usage. To do this use tools, PowerPoint, or a combine models as in this example (which you are free to cut and paste from). Insert indications in red (as in this example) to show where the three features below apply.



1.5.2 Code showing an abstract class.

Show the relevant code (only) and explain why an abstract class is appropriate here. It should be clear where the code is located (class and method).

I am using this Scale class as an abstract base class from which many different kinds of scales will derive. There are many things that all scales have in common and I tried to isolate those commonalities here. I also included static methods in here that will be useful in all scale operations without the need for an instantiated object.

```
package edu.cs622;


import java.util.HashMap;

import java.util.List;

import java.util.Map;


public abstract class Scale {

/**
```

```
* @author - Richard Hanks
* Interface for representing a music scale including instance variables and methods for accessing
* scale information.
* @param root int The root of the scale, also known as the tonic
* @param pitchesByNumber Map<int, String> A collection of the pitches in the scale. The key is the int value of the
* position in the scale.  The value is the String value of the pitch name
* @param pitchesByName Map<String, int>  A collection of the pitches in the scale.  The key is the String value of the
* pitch name. The value is the int value of the position in the scale
* @param static pitchNameMapping Map<Integer, String> is the mapping of all the the pitch numbers in the chromatic
* scale to the names of the corresponding pitches
* @param availableQualities String is the qualities of the intervals allowed for a given type of scale (ie. Perfect, Major);
*/
		protected int root;
		protected Map<Integer, String> pitchesByNumber;
		protected Map<String, Integer> pitchesByName;
		public static final Map<Integer, String> pitchNameMapping = getPitchNameMapping();
		public static final Map<String, Integer> IntervalMapping = getIntervalMapping();
		protected String[] availableQualities;

		/**
		 * Static method returns the pitch, as an int, a given number
		 * of halfSteps above the given fundamental
		 * @param fundamental The base of the interval from which to calculate halfSteps
		 * @param halfSteps The distance to the top notes, in half steps
		 * @return the int value associated with the upper note in the interval
		 */
		public static int getInterval(int fundamental, int halfSteps){
			// Ensure that going down an interval loops properly
			if (halfSteps < 0){
				int funPlusHalf = fundamental + halfSteps;
				while(funPlusHalf < -12){
					funPlusHalf += 12;
				}
```

```
                        return ((funPlusHalf) + 12) % 12;
                }
                return (fundamental + halfSteps) % 12;
}

/**
 * Creates the static constant mapping between the names of intervals and their integer values
 * @return a Map<String, Integer> where the key is the name of the interval and the value is the Integer
 * representation of the interval in half steps (aka semi tones)
 * TODO this mapping is incomplete.  Need to finish to be fully functional!
 */
static Map<String, Integer> getIntervalMapping() {
                Map<String, Integer> sampleMapping = new HashMap<>();
                sampleMapping.put("perfect unison", 0);
                sampleMapping.put("minor second above",  1);
                sampleMapping.put("minor 2nd below", -1);
                sampleMapping.put("major second above",  2);
                sampleMapping.put("major 2nd below", -2);
                return sampleMapping;

}
public abstract String getScaleDegreeName(int scaleDegree);
public abstract int getScaleDegreePitch(String name);
public abstract int getRoot();

/**
 * Method called to instantiate the static mapping of pitch numbers  to their corresponding pitch names
 * @return The mapping of pitch numbers to their corresponding pitch names.  C is represented by 0 and
 * each chromatic step higher, 1/2 step, corresponds to the next letter name, including accidentals
 */
public static Map<Integer, String> getPitchNameMapping() {
                Map<Integer, String> placeHolderMapping = new HashMap<Integer, String>();
                placeHolderMapping.put(0, "C");
```

```
                placeHolderMapping.put(1,  "C#");
                placeHolderMapping.put(2,  "D");
                placeHolderMapping.put(3, "Eb");
                placeHolderMapping.put(4,  "E");
                placeHolderMapping.put(5,  "F");
                placeHolderMapping.put(6,  "F#");
                placeHolderMapping.put(7,  "G");
                placeHolderMapping.put(8,  "Ab");
                placeHolderMapping.put(9,  "A");
                placeHolderMapping.put(10,  "Bb");
                placeHolderMapping.put(11,  "B");

                return placeHolderMapping;
        }


        public abstract Map<Integer, String> getPitchesByNumber();
        public abstract Map<String, Integer> getPitchesByName();
}
```

1.5.3 Code showing polymorphism.

Show the relevant code (only) and explain why *polymorphism* is appropriate here. It should be clear where the code is located (class and method).

I used polymorphism to check which class the specific question comes from. Using this information, I update a score related to that question class so that I can report back to the user what his/her score is for each type of question.

```
        // provide the correct answer
        if(correctAnswer){
            System.out.println("Correct!");
            // Keep track of correct answers for each type of question
            if(question.getClass() == SimpleIntervalUpMajorScale.class){
                String key = "Simple Interval Up Major Scale";
                String updatedScore = updateScore(key);
                System.out.println(updatedScore);

            }else if(question.getClass() == SimpleIntervalDownMajorScale.class){
                String key = "Simple Interval Down Major Scale";
                String updatedScore = updateScore(key);
                System.out.println(updatedScore);
            }
```

1.5.4 Code showing upcasting or downcasting.

Show the relevant code (only) and explain why upcasting or downcasting is appropriate here. It should be clear where the code is located (class and method).

In the main method of the application I create a List of <<<Question>>> subclasses that I instantiate with random values. When asking for questions and getting answers, I need to call methods and get instance variables that don't exist in the Question class, so I need to explicitly downcast. From Quiz.main() ———>

```
        //get random question
        question = (AbstractQuestion) getQuestion();
        // Ask question
        System.out.println(((AbstractQuestion)question).question);
        System.out.println("Enter answer or type \"Q\" to exit");
        // get user answer
        String answer = reader.next();
        if(answer.toUpperCase().equals("Q")){
            askQuestions = false;
            System.out.println("Goodbye!");
            break;
        }
        // check answer against and return if the answer was correct
        boolean correctAnswer = question.checkAnswer(answer.toUpperCase());
        // provide the correct answer
        if(correctAnswer){
            System.out.println("Correct!");
        }else{
            System.out.println("Sorry, that was not correct.  Correct answer was: " + ((AbstractQuestion)question).answer);
        }
    }

    reader.close();
```

1.6 YOUR CODE

Unless your facilitator requests another method, copy your Eclipse project to your file system, zip it, and attach it. Please contact your facilitator in advance if you want to request an exception.

Code is included in zipped file.

1.7 Instructor's Evaluation

Criterion	D	C	B	A	Letter Grade	%
Technical Correctness	No justification of correctness	Technically mostly correct	Explanation justify technical correctness; appropriately used	Correct, complete, and thorough technical justification; very appropriately used		0.0
Clarity in presentation of project	Unclear	Somewhat clear	Clear	Entirely clear		0.0
Understan-ding of the relavent technologies	Minor understanding evidenced	Satisfactory understanding evidenced	Evidence of good understanding throughout	Evidence throughout of entirely thorough understanding		0.0
				Assignment Grade:		0.0
The resulting grade is the average of these, using A+=97, A=95, A-=90, B+=87, B=85, B-=80 etc.						
To obtain an A grade for the course, your weighted average should be >93. A-:>=90. B+:>=87. B:>83. B-:>=80 etc.						

Appendix 1 (will be read as-needed only)

Appendix 2 (will be read as-needed only)

Appendix 3 (will be read as-needed only)