COP 3330: Object-Oriented Programming
Summer 2011

EXAM #2 Review

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cop3330/sum2011

Department of Electrical Engineering and Computer Science
Computer Science Division
University of Central Florida

Material Covered

- Classes in Java – 3 sections of notes. More emphasis placed on last section of these notes that dealt with abstract classes and interfaces which were only touched on by exam 1.
- However, don't neglect the topics of inheritance and polymorphism from the earlier sections of the Classes in Java notes.
- Exception Handling - 1 section of notes.
- GUIs and event-driven programming will appear on the final exam NOT this exam.
- All previous material – Don't forget the earlier material – it all builds from the start.

Test Format

- Some True/False questions
- Some fill-in-the-blanks questions.
- Some tracing through code and producing the output.
- Some writing of Java console application programs.
- Very similar in format to exam 1.

Sample Test Questions

1. Construct the UML diagram for the class `A` as described below:

   - `m` is an integer instance that should not be accessible outside of class `A` or to any class that extends `A`.
   - `n` is an integer instance that should be accessible only to classes that extend class `A` or are located in the same package as class `A`.
   - `setM`, `getM`, `setN`, and `getN` are instance methods that should be accessible to any class.
   - `calculate()` is an abstract method that returns an integer value and requires two integer parameters.

Sample Test Questions

2. Construct the UML diagram for the class `B`, which extends class `A` from problem 1. Class `B` is described below:

   – `r` is an integer instance that should not be accessible outside of class `B` or to any class that extends `B`.
   – `s` is an integer instance that should be accessible only to the class `B` or classes are located in the same package as class `B`.
   – `setR`, `getR`, `setS`, and `getS` are instance methods that should be accessible to any class.
   – `calculate()` should return the product of its parameters.

Sample Test Questions (cont.)

3. Answer the following multiple choice questions.

   a) A method in a subclass that has the same signature as a method in the superclass is an example of:

      a) overloading
      b) overriding
      c) composition
      d) an error in Java

   b) A subclass does not have access to these superclass members:

      a) public
      b) private
      c) protected
      d) all of these

Sample Test Questions (cont.)

4. Fill in the blank with the correct term.

a) When a method is declared ______ it cannot be overridden in a subclass.

b) In a subclass constructor, a call to the superclass constructor must ______ ____________________.

c) Every member in an interface must be declared ____________________

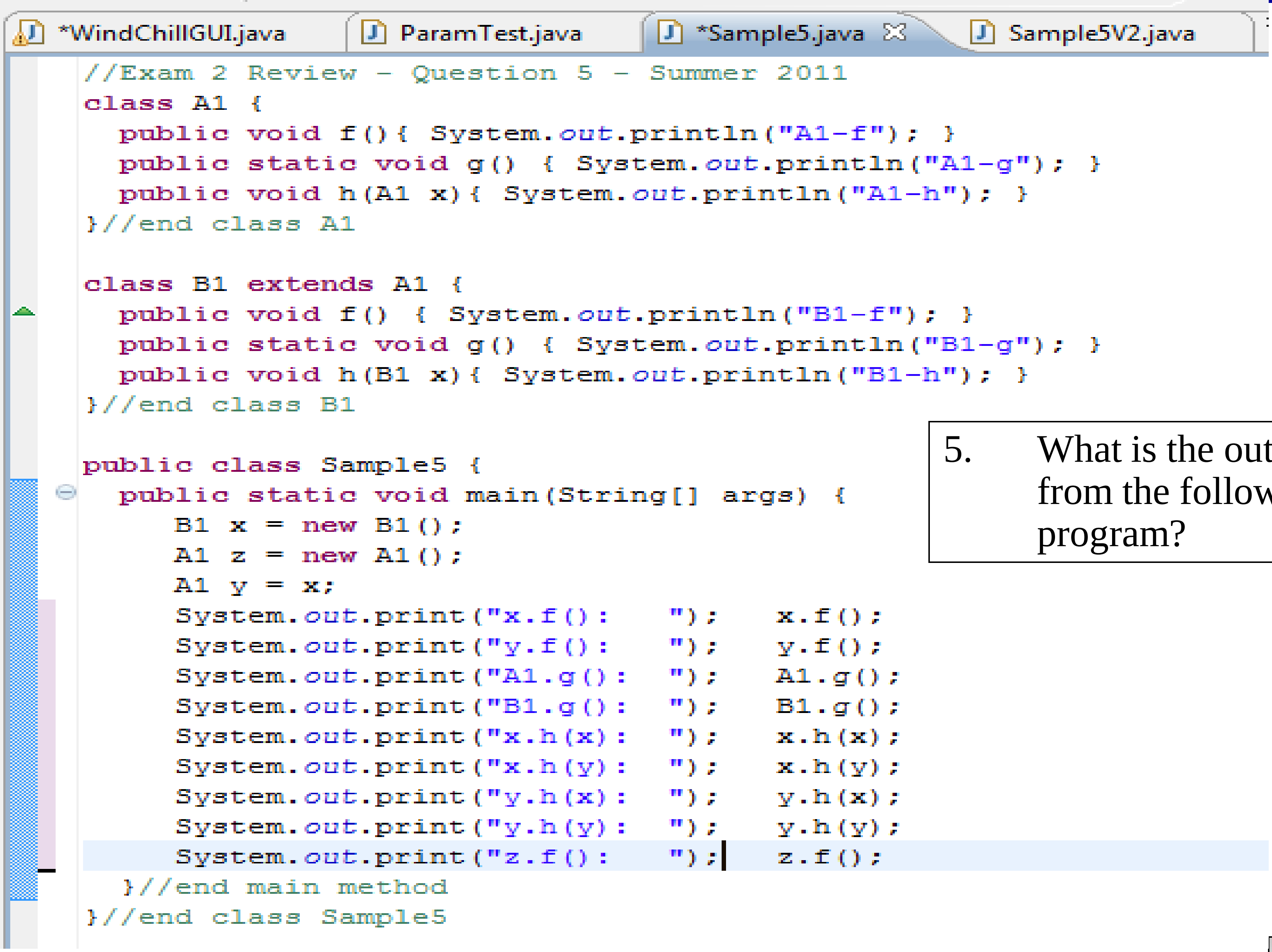
d) Java's mechanism for handling exceptions in an executing program is done using a ______ statement.

e) The number of classes that a class can extend in Java is limited to ______.

f) The number of interfaces that a class can implement in Java is ____________.

g) A try statement can include this many catch blocks (clauses) ____________.

h) The block of statements following a try statement that are executed regardless of whether or not an exception occurred are included in this type of clause: ____________.

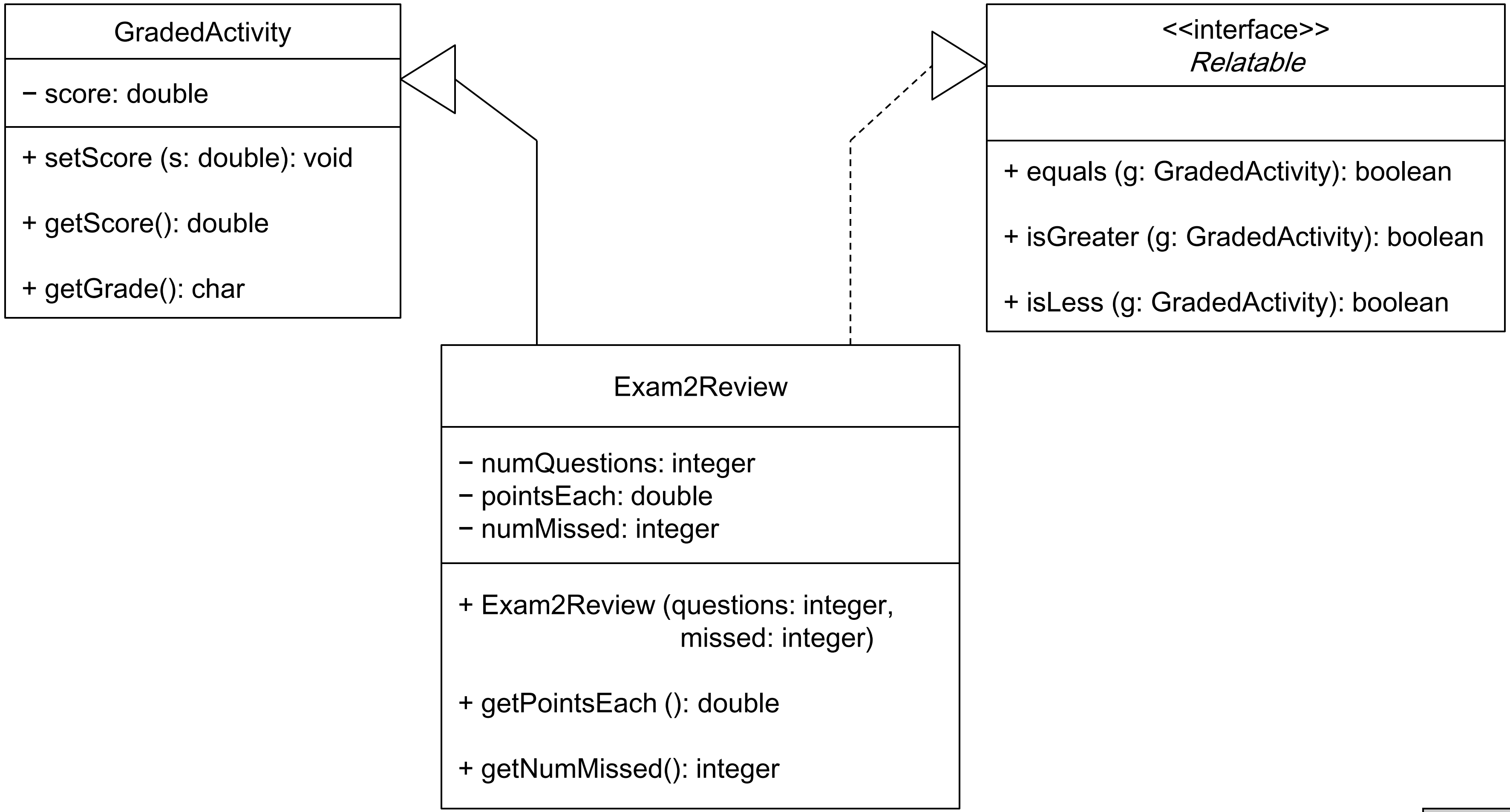


```
//Exam 2 Review - Question 5 - Summer 2011
class A1 {
  public void f(){ System.out.println("A1-f"); }
  public static void g() { System.out.println("A1-g"); }
  public void h(A1 x){ System.out.println("A1-h"); }
}//end class A1

class B1 extends A1 {
  public void f() { System.out.println("B1-f"); }
  public static void g() { System.out.println("B1-g"); }
  public void h(B1 x){ System.out.println("B1-h"); }
}//end class B1

public class Sample5 {
  public static void main(String[] args) {
      B1 x = new B1();
      A1 z = new A1();
      A1 y = x;
      System.out.print("x.f():    ");   x.f();
      System.out.print("y.f():    ");   y.f();
      System.out.print("A1.g():   ");   A1.g();
      System.out.print("B1.g():   ");   B1.g();
      System.out.print("x.h(x):   ");   x.h(x);
      System.out.print("x.h(y):   ");   x.h(y);
      System.out.print("y.h(x):   ");   y.h(x);
      System.out.print("y.h(y):   ");   y.h(y);
      System.out.print("z.f():    ");   z.f();
  }//end main method
}//end class Sample5
```

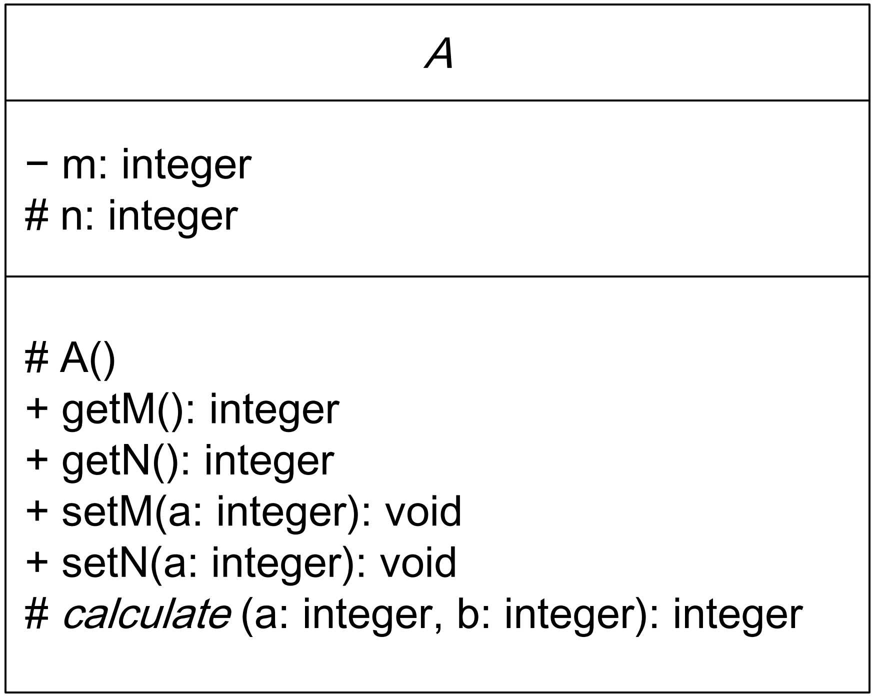
5. What is the output from the following program?

Sample Test Questions

6. Implement the classes shown in the UML diagram below.


GradedActivity
− score: double
+ setScore (s: double): void
+ getScore(): double
+ getGrade(): char
<<interface>>
Relatable
+ equals (g: GradedActivity): boolean
+ isGreater (g: GradedActivity): boolean
+ isLess (g: GradedActivity): boolean
Exam2Review
− numQuestions: integer
− pointsEach: double
− numMissed: integer
+ Exam2Review (questions: integer, missed: integer)
+ getPointsEach (): double
+ getNumMissed(): integer

Answer Sample Question #1

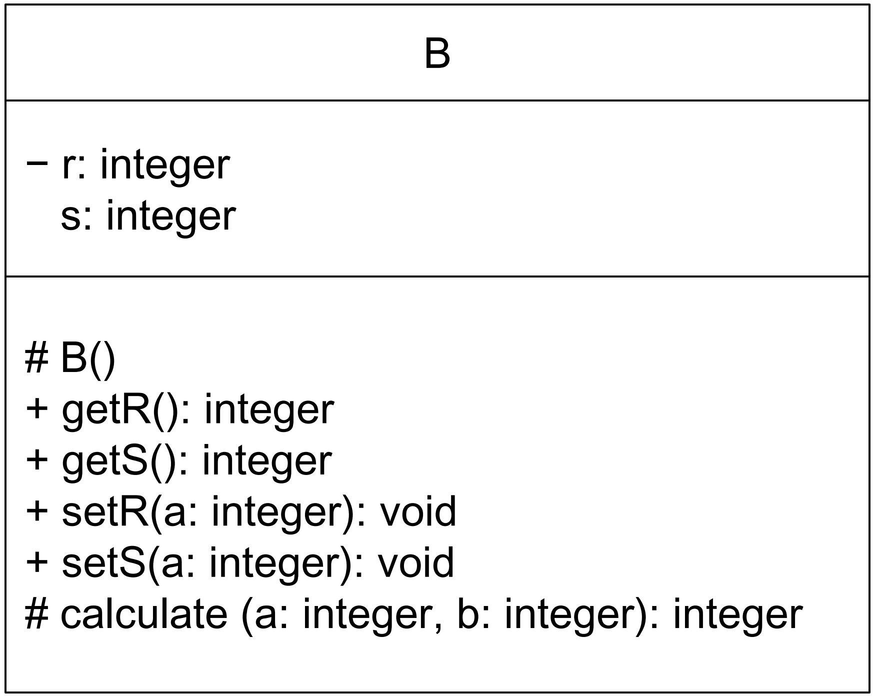


A

− m: integer
# n: integer

A()
+ getM(): integer
+ getN(): integer
+ setM(a: integer): void
+ setN(a: integer): void
# *calculate* (a: integer, b: integer): integer

Answer Sample Question #2



```
B
−  r: integer
   s: integer

# B()
+ getR(): integer
+ getS(): integer
+ setR(a: integer): void
+ setS(a: integer): void
# calculate (a: integer, b: integer): integer
```

Answers Sample Test Questions (cont.)

3. Answer the following multiple choice questions.

   a) A method in a subclass that has the same signature as a method in the superclass is an example of:

      a) overloading
      b) overriding
      c) composition
      d) an error in Java

   b) A subclass does not have access to these superclass members:

      a) public
      b) private
      c) protected
      d) all of these

Sample Test Questions (cont.)

4. Fill in the blank with the correct term.

a) When a method is declared final it cannot be overridden in a subclass.

b) In a subclass constructor, a call to the superclass constructor must appear as the first statement.

c) Every member in an interface must be declared public, final, and static.

d) Java's mechanism for handling exceptions in an executing program is done using a try statement.

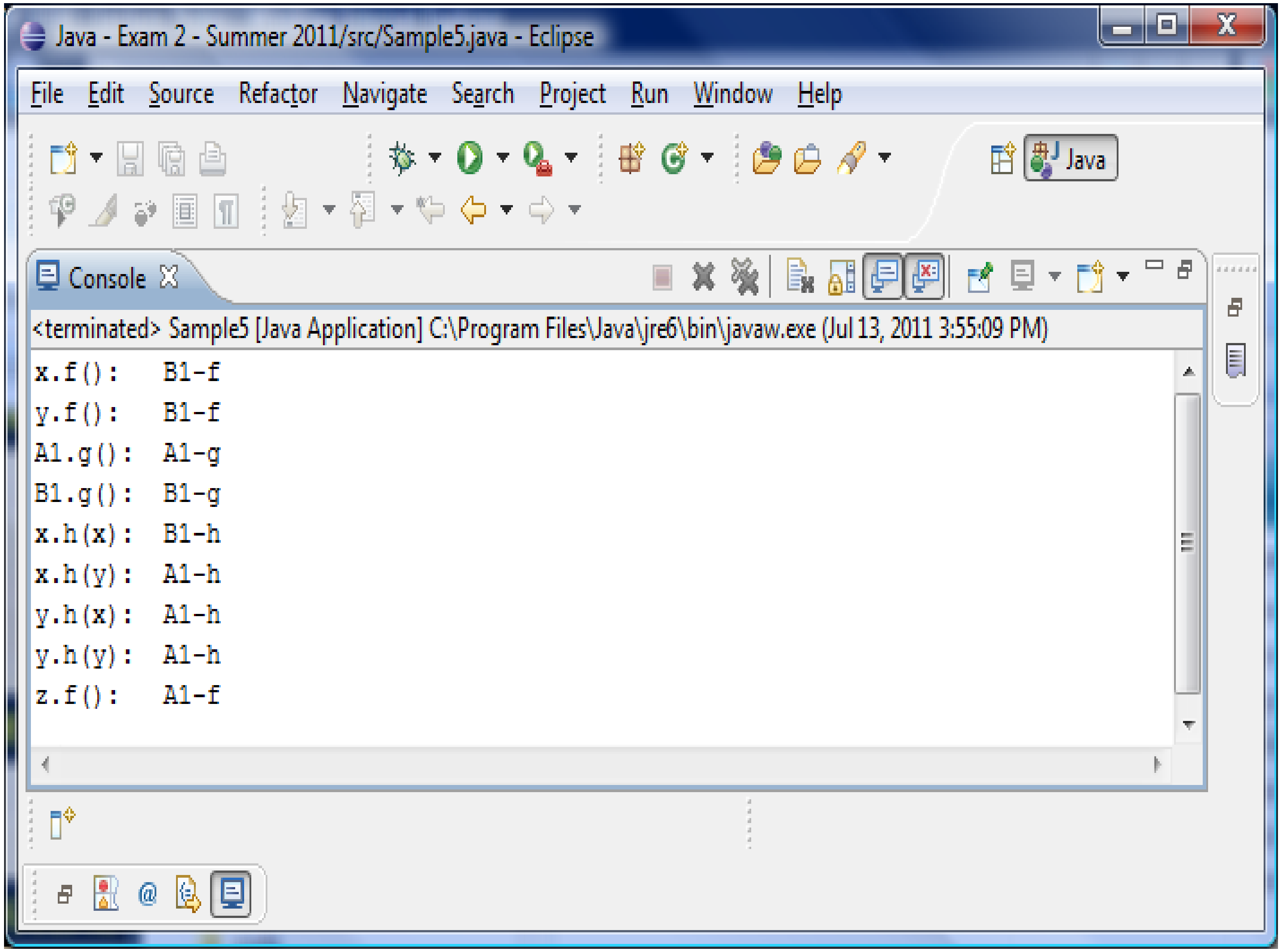
e) The number of classes that a class can extend in Java is limited to one.

f) The number of interfaces that a class can implement in Java is unlimited.

g) A try statement can include this many catch blocks (clauses) any number.

h) The block of statements following a try statement that are executed regardless of whether or not an exception occurred are included in this type of clause: finally.

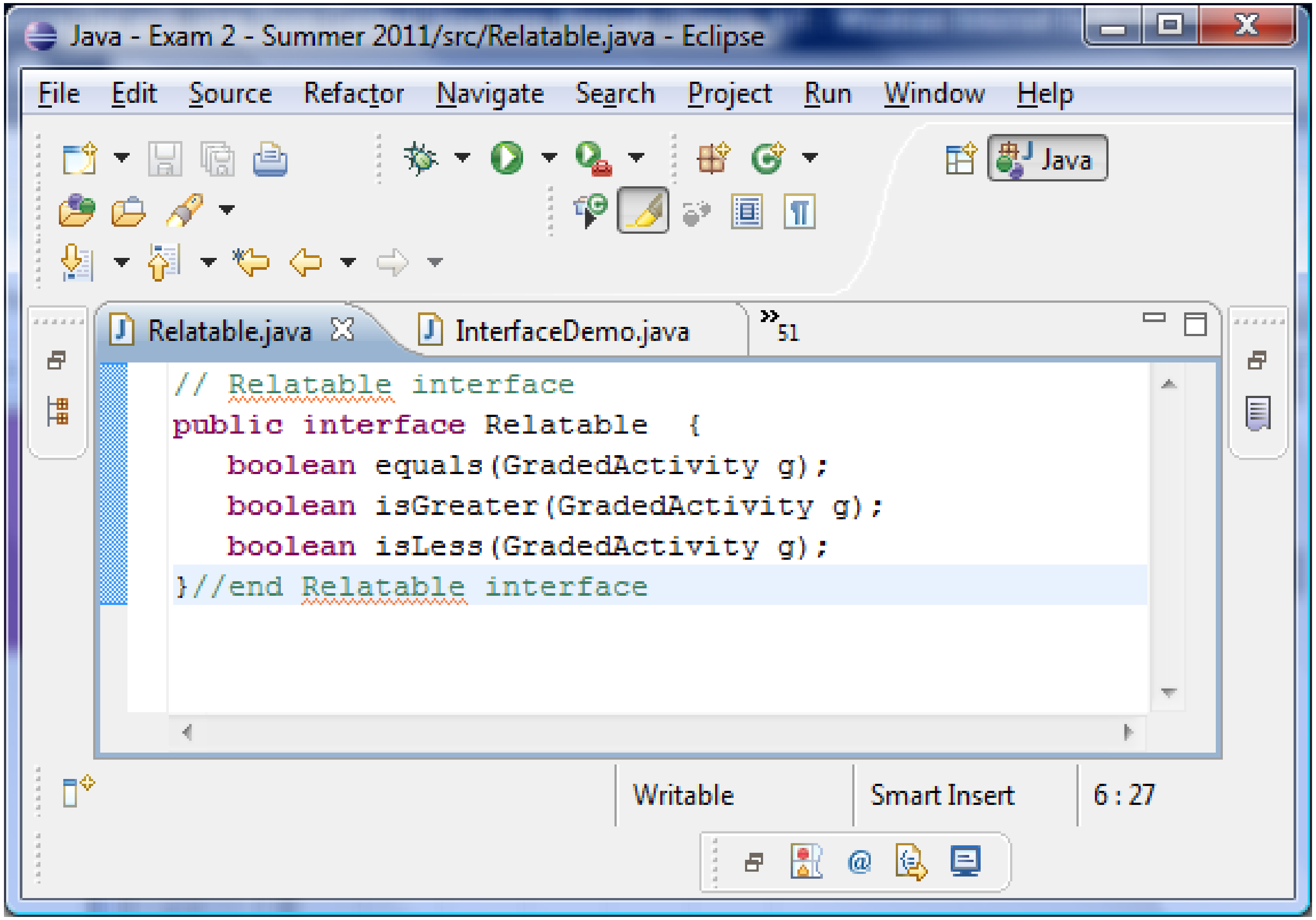
Answer Sample Question #5


Java - Exam 2 - Summer 2011/src/Sample5.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Console

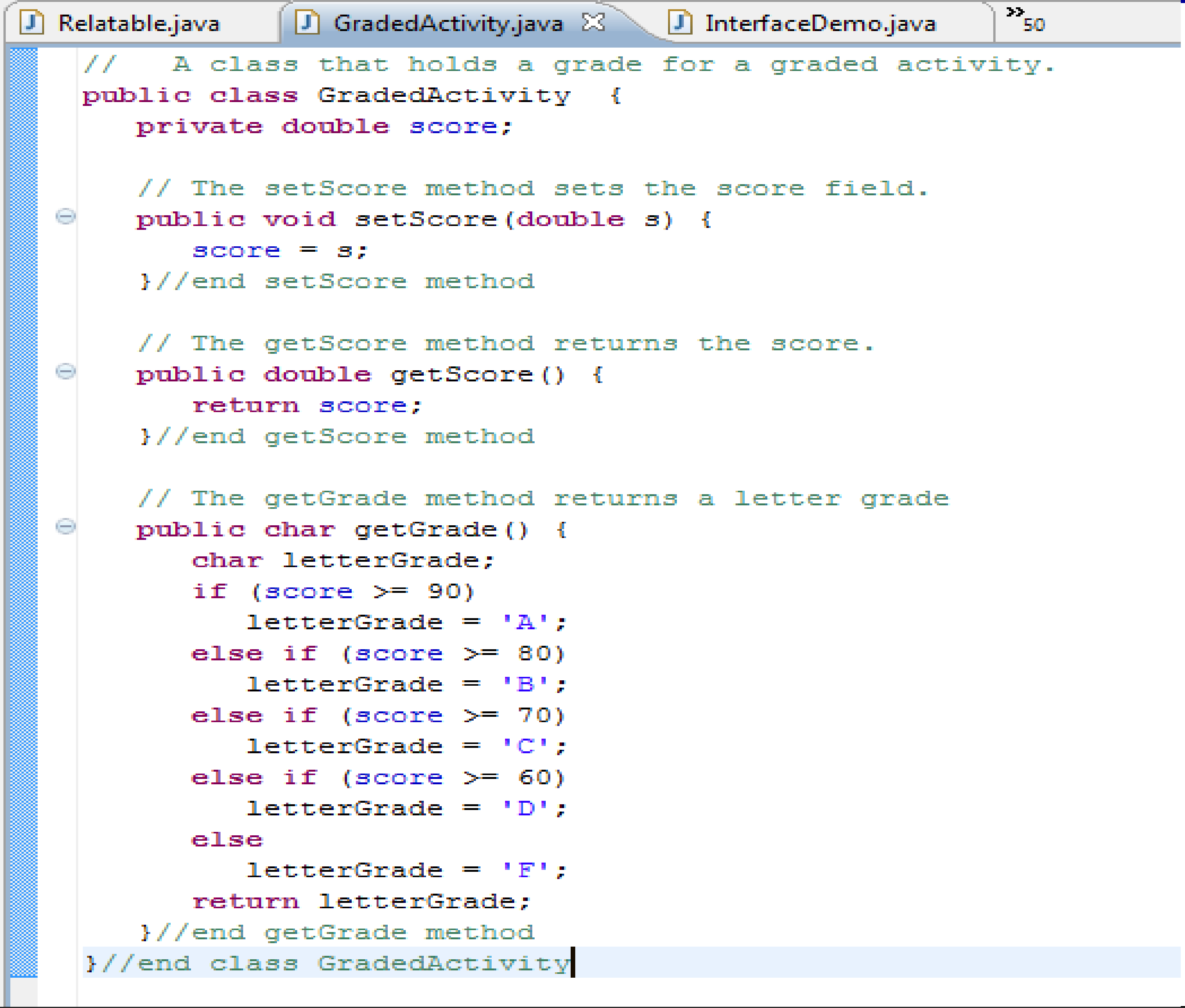
<terminated> Sample5 [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Jul 13, 2011 3:55:09 PM)

```
x.f():   B1-f
y.f():   B1-f
A1.g():  A1-g
B1.g():  B1-g
x.h(x):  B1-h
x.h(y):  A1-h
y.h(x):  A1-h
y.h(y):  A1-h
z.f():   A1-f
```

Answer Sample Question #6

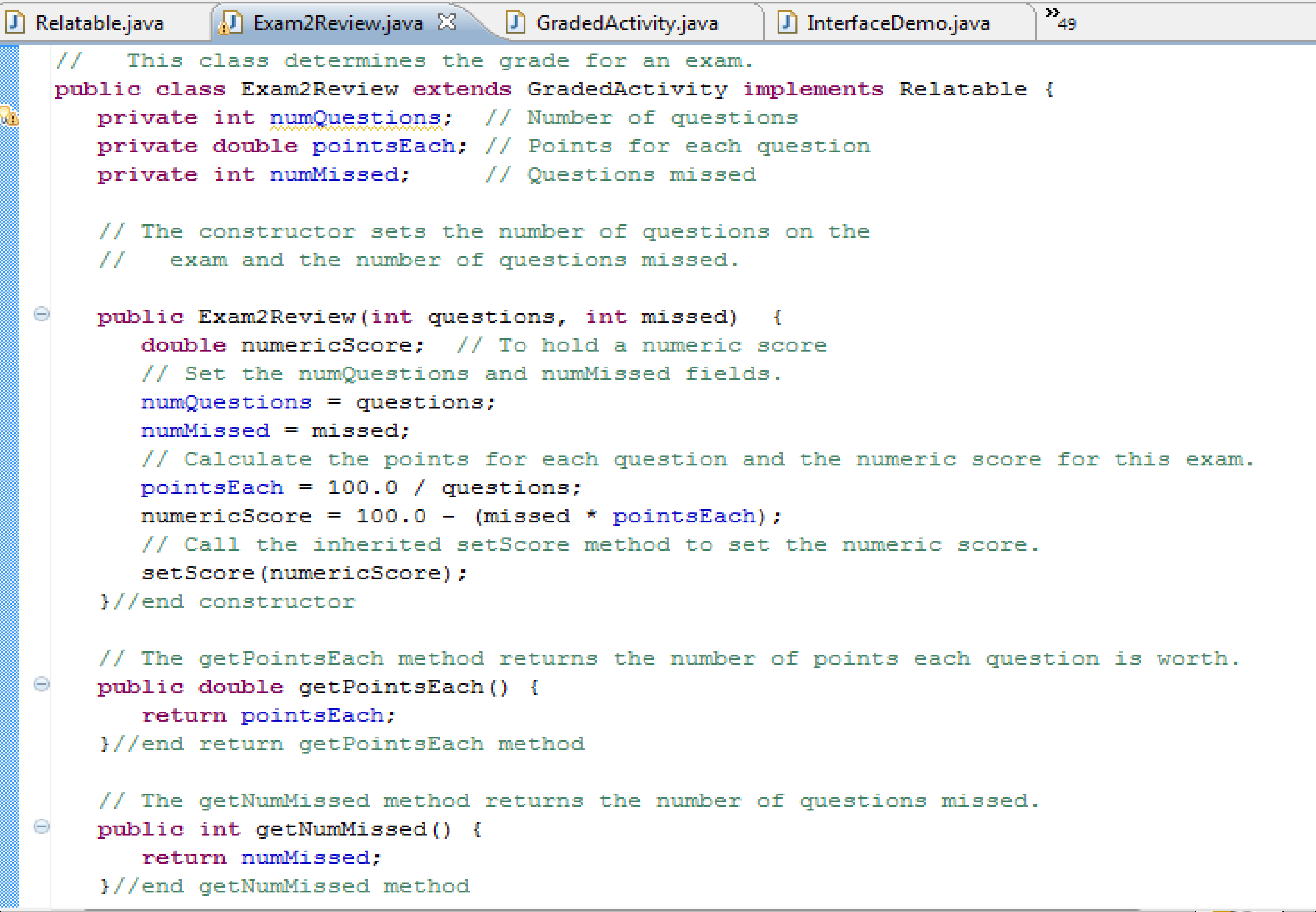

Java - Exam 2 - Summer 2011/src/Relatable.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Relatable.java
InterfaceDemo.java
51
Writable
Smart Insert
6 : 27

```java
// Relatable interface
public interface Relatable  {
   boolean equals(GradedActivity g);
   boolean isGreater(GradedActivity g);
   boolean isLess(GradedActivity g);
}//end Relatable interface
```

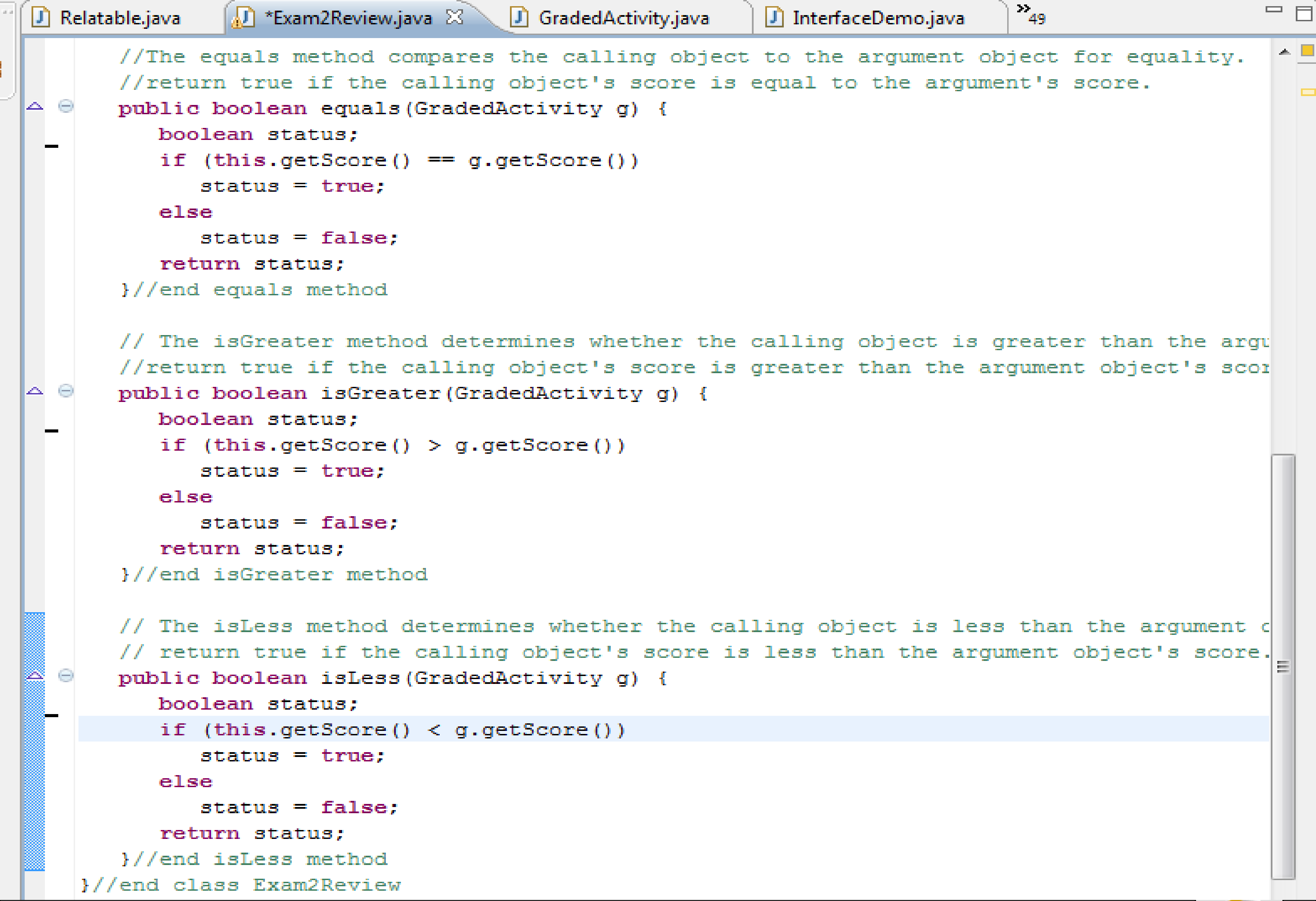


Relatable.java | GradedActivity.java | InterfaceDemo.java | »50

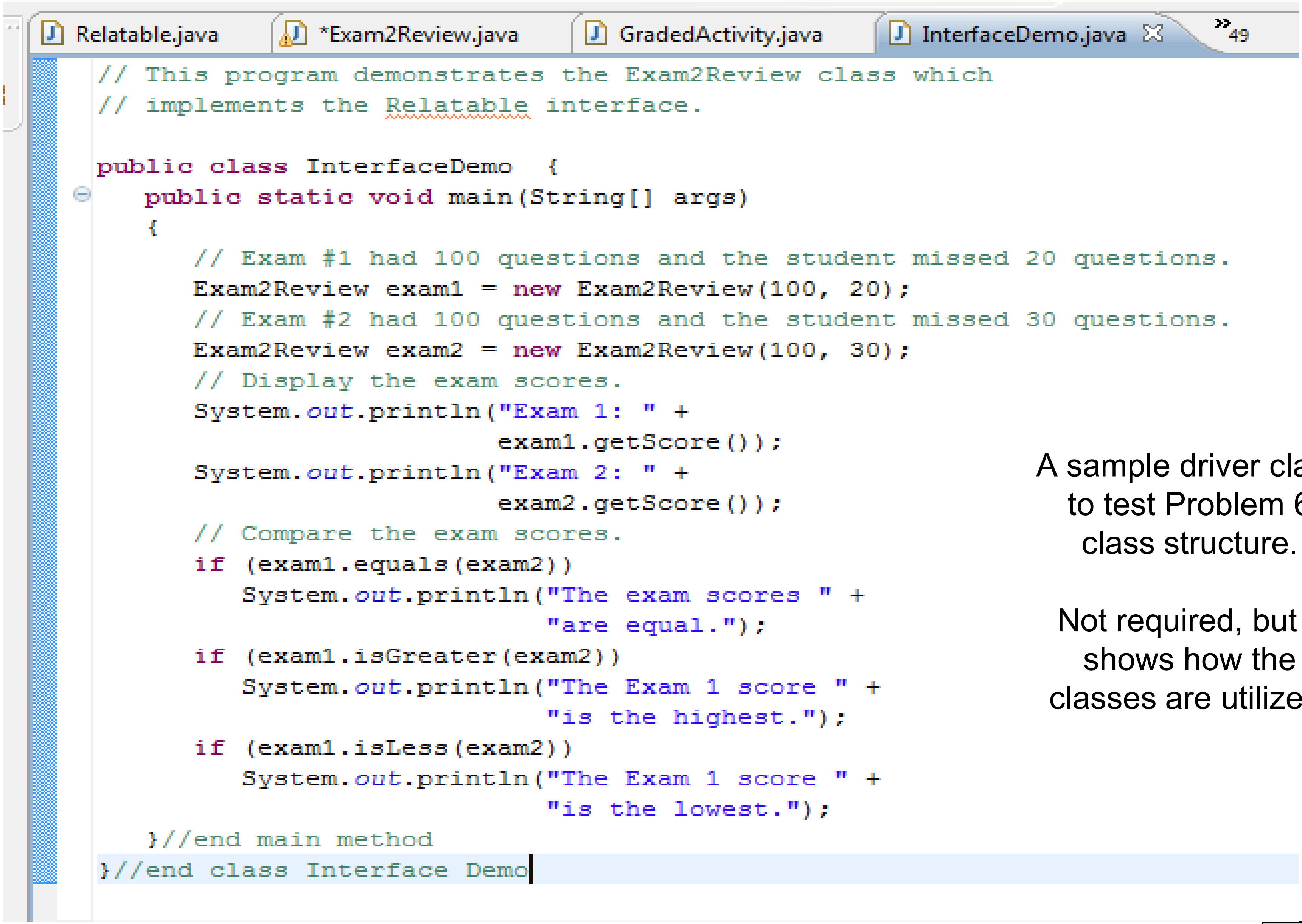
```java
//   A class that holds a grade for a graded activity.
public class GradedActivity  {
   private double score;

   // The setScore method sets the score field.
   public void setScore(double s) {
      score = s;
   }//end setScore method

   // The getScore method returns the score.
   public double getScore() {
      return score;
   }//end getScore method

   // The getGrade method returns a letter grade
   public char getGrade() {
      char letterGrade;
      if (score >= 90)
         letterGrade = 'A';
      else if (score >= 80)
         letterGrade = 'B';
      else if (score >= 70)
         letterGrade = 'C';
      else if (score >= 60)
         letterGrade = 'D';
      else
         letterGrade = 'F';
      return letterGrade;
   }//end getGrade method
}//end class GradedActivity
```



```java
//   This class determines the grade for an exam.
public class Exam2Review extends GradedActivity implements Relatable {
   private int numQuestions;  // Number of questions
   private double pointsEach; // Points for each question
   private int numMissed;     // Questions missed

   // The constructor sets the number of questions on the
   //   exam and the number of questions missed.

   public Exam2Review(int questions, int missed)  {
      double numericScore;  // To hold a numeric score
      // Set the numQuestions and numMissed fields.
      numQuestions = questions;
      numMissed = missed;
      // Calculate the points for each question and the numeric score for this exam.
      pointsEach = 100.0 / questions;
      numericScore = 100.0 - (missed * pointsEach);
      // Call the inherited setScore method to set the numeric score.
      setScore(numericScore);
   }//end constructor

   // The getPointsEach method returns the number of points each question is worth.
   public double getPointsEach() {
      return pointsEach;
   }//end return getPointsEach method

   // The getNumMissed method returns the number of questions missed.
   public int getNumMissed() {
      return numMissed;
   }//end getNumMissed method
```



```
    //The equals method compares the calling object to the argument object for equality.
    //return true if the calling object's score is equal to the argument's score.
    public boolean equals(GradedActivity g) {
       boolean status;
       if (this.getScore() == g.getScore())
          status = true;
       else
          status = false;
       return status;
    }//end equals method

    // The isGreater method determines whether the calling object is greater than the argu
    //return true if the calling object's score is greater than the argument object's scor
    public boolean isGreater(GradedActivity g) {
       boolean status;
       if (this.getScore() > g.getScore())
          status = true;
       else
          status = false;
       return status;
    }//end isGreater method

    // The isLess method determines whether the calling object is less than the argument c
    // return true if the calling object's score is less than the argument object's score.
    public boolean isLess(GradedActivity g) {
       boolean status;
       if (this.getScore() < g.getScore())
          status = true;
       else
          status = false;
       return status;
    }//end isLess method
}//end class Exam2Review
```


Relatable.java | *Exam2Review.java | GradedActivity.java | InterfaceDemo.java | »49

```
// This program demonstrates the Exam2Review class which
// implements the Relatable interface.

public class InterfaceDemo  {
   public static void main(String[] args)
   {
      // Exam #1 had 100 questions and the student missed 20 questions.
      Exam2Review exam1 = new Exam2Review(100, 20);
      // Exam #2 had 100 questions and the student missed 30 questions.
      Exam2Review exam2 = new Exam2Review(100, 30);
      // Display the exam scores.
      System.out.println("Exam 1:  " +
                         exam1.getScore());
      System.out.println("Exam 2:  " +
                         exam2.getScore());
      // Compare the exam scores.
      if (exam1.equals(exam2))
         System.out.println("The exam scores " +
                            "are equal.");
      if (exam1.isGreater(exam2))
         System.out.println("The Exam 1 score " +
                            "is the highest.");
      if (exam1.isLess(exam2))
         System.out.println("The Exam 1 score " +
                            "is the lowest.");
   }//end main method
}//end class Interface Demo
```

A sample driver class to test Problem 6 class structure.

Not required, but it shows how the classes are utilized.