1. This question involves simulation of the play and scoring of a single-player video game. In the game, a player attempts to complete three levels. A level in the game is represented by the `Level` class.

```
public class Level
{
   /** Returns true if the player reached the goal on this level and returns false otherwise */
   public boolean goalReached()
   {  /* implementation not shown */  }

   /** Returns the number of points (a positive integer) recorded for this level */
   public int getPoints()
   {  /* implementation not shown */  }

   // There may be instance variables, constructors, and methods that are not shown.
}
```


© 2022 College Board.
Visit College Board on the web: collegeboard.org.

Play of the game is represented by the `Game` class. You will write two methods of the `Game` class.

```
public class Game
{
   private Level levelOne;
   private Level levelTwo;
   private Level levelThree;

   /** Postcondition: All instance variables have been initialized. */
   public Game()
   {  /* implementation not shown */  }

   /** Returns true if this game is a bonus game and returns false otherwise */
   public boolean isBonus()
   {  /* implementation not shown */  }

   /** Simulates the play of this Game (consisting of three levels) and updates all relevant
    *  game data
    */
   public void play()
   {  /* implementation not shown */  }

   /** Returns the score earned in the most recently played game, as described in part (a) */
   public int getScore()
   {  /* to be implemented in part (a) */  }

   /** Simulates the play of num games and returns the highest score earned, as
    *  described in part (b)
    *  Precondition: num > 0
    */
   public int playManyTimes(int num)
   {  /* to be implemented in part (b) */  }

   // There may be instance variables, constructors, and methods that are not shown.
}
```


© 2022 College Board.
Visit College Board on the web: collegeboard.org.

(a) Write the `getScore` method, which returns the score for the most recently played game. Each game consists of three levels. The score for the game is computed using the following helper methods.

- The `isBonus` method of the `Game` class returns `true` if this is a bonus game and returns `false` otherwise.
- The `goalReached` method of the `Level` class returns `true` if the goal has been reached on a particular level and returns `false` otherwise.
- The `getPoints` method of the `Level` class returns the number of points recorded on a particular level. Whether or not recorded points are earned (included in the game score) depends on the rules of the game, which follow.

The score for the game is computed according to the following rules.

- Level one points are earned only if the level one goal is reached. Level two points are earned only if both the level one and level two goals are reached. Level three points are earned only if the goals of all three levels are reached.
- The score for the game is the sum of the points earned for levels one, two, and three.
- If the game is a bonus game, the score for the game is tripled.

© 2022 College Board.
Visit College Board on the web: collegeboard.org.

The following table shows some examples of game score calculations.

	Level One Results	Level Two Results	Level Three Results	`isBonus` Return Value	Score Calculation
`goalReached` **Return Value:**	`true`	`true`	`true`	`true`	(200 + 100 + 500) × 3 = 2,400 The recorded points for levels one, two, and three are earned because the goals were reached in all three levels. The earned points are multiplied by 3 because `isBonus` returns `true`.
`getPoints` **Return Value:**	`200`	`100`	`500`		
`goalReached` **Return Value:**	`true`	`true`	`false`	`false`	200 + 100 = 300 The recorded points for level one and level two are earned because the goal was reached in levels one and two. The recorded points for level three are not earned because the goal was not reached in level three.
`getPoints` **Return Value:**	`200`	`100`	`500`		
`goalReached` **Return Value:**	`true`	`false`	`true`	`true`	200 × 3 = 600 The recorded points for only level one are earned because the goal was not reached in level two. The earned points are multiplied by 3 because `isBonus` returns `true`.
`getPoints` **Return Value:**	`200`	`100`	`500`		
`goalReached` **Return Value:**	`false`	`true`	`true`	`false`	0 Because the goal in level one was not reached, no points are earned for any level.
`getPoints` **Return Value:**	`200`	`100`	`500`		

Complete the `getScore` method.

```
/** Returns the score earned in the most recently played game, as described in part (a) */
public int getScore()
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number.
If there are multiple parts to this question, write the part letter with your response.

GO ON TO THE NEXT PAGE.

© 2022 College Board.
Visit College Board on the web: collegeboard.org.

(b) Write the `playManyTimes` method, which simulates the play of `num` games and returns the highest game score earned. For example, if the four plays of the game that are simulated as a result of the method call `playManyTimes(4)` earn scores of `75`, `50`, `90`, and `20`, then the method should return `90`.

Play of the game is simulated by calling the helper method `play`. Note that if `play` is called only one time followed by multiple consecutive calls to `getScore`, each call to `getScore` will return the score earned in the single simulated play of the game.

Complete the `playManyTimes` method. Assume that `getScore` works as intended, regardless of what you wrote in part (a). You must call `play` and `getScore` appropriately in order to receive full credit.

```
/** Simulates the play of num games and returns the highest score earned, as
 *  described in part (b)
 *  Precondition: num > 0
 */
public int playManyTimes(int num)
```

Begin your response at the top of a new page in the Free Response booklet and fill in the appropriate circle indicating the question number. If there are multiple parts to this question, write the part letter with your response.

Class information for this question

```
public class Level

public boolean goalReached()
public int getPoints()

public class Game

private Level levelOne
private Level levelTwo
private Level levelThree

public Game()
public boolean isBonus()
public void play()
public int getScore()
public int playManyTimes(int num)
```

GO ON TO THE NEXT PAGE.

© 2022 College Board.
Visit College Board on the web: collegeboard.org.

2. The `Book` class is used to store information about a book. A partial `Book` class definition is shown.

```
public class Book
{
   /** The title of the book */
   private String title;

   /** The price of the book */
   private double price;

   /** Creates a new Book with given title and price */
   public Book(String bookTitle, double bookPrice)
   {  /* implementation not shown */  }

   /** Returns the title of the book */
   public String getTitle()
   {  return title;  }

   /** Returns a string containing the title and price of the Book */
   public String getBookInfo()
   {
      return title + "-" + price;
   }

   // There may be instance variables, constructors, and methods that are not shown.
}
```

You will write a class `Textbook`, which is a subclass of `Book`.

A `Textbook` has an edition number, which is a positive integer used to identify different versions of the book. The `getBookInfo` method, when called on a `Textbook`, returns a string that also includes the edition information, as shown in the example.

Information about the book title and price must be maintained in the `Book` class. Information about the edition must be maintained in the `Textbook` class.

The `Textbook` class contains an additional method, `canSubstituteFor`, which returns `true` if a `Textbook` is a valid substitute for another `Textbook` and returns `false` otherwise. The current `Textbook` is a valid substitute for the `Textbook` referenced by the parameter of the `canSubstituteFor` method if the two `Textbook` objects have the same title and if the edition of the current `Textbook` is greater than or equal to the edition of the parameter.

GO ON TO THE NEXT PAGE.

© 2022 College Board.
Visit College Board on the web: collegeboard.org.

Question 1: Methods and Control Structures — 9 points

Canonical solution

(a) — **4 points**

```
public int getScore()
{
   int score = 0;

   if (levelOne.goalReached())
   {
      score = levelOne.getPoints();

      if (levelTwo.goalReached())
      {
         score += levelTwo.getPoints();

         if (levelThree.goalReached())
         {
            score += levelThree.getPoints();
         }
      }
   }

   if (isBonus())
   {
      score *= 3;
   }

   return score;
}
```

(b) — **5 points**

```
public int playManyTimes(int num)
{
   int max = 0;

   for (int i = 0; i < num; i++)
   {
      play();
      int score = getScore();
      if (score > max)
      {
         max = score;
      }
   }

   return max;
}
```

© 2022 College Board

(a) `getScore`

	Scoring Criteria	Decision Rules	
1	Calls `getPoints`, `goalReached`, and `isBonus`	Responses **will not** earn the point if they • fail to call `getPoints` or `goalReached` on a `Level` object • call `isBonus` on an object other than `this` (use of `this` is optional) • include parameters	**1 point**
2	Determines if points are earned based on `goalReached` return values	Responses **can** still earn the point even if they • calculate the score total incorrectly • call `goalReached` incorrectly • fail to distinguish all cases correctly Responses **will not** earn the point if they • fail to use a nested `if` statement or equivalent	**1 point**
3	Guards update of score for bonus game based on `isBonus` return value	Responses **can** still earn the point even if they • triple the calculated score incorrectly • update the score with something other than tripling • call `isBonus` incorrectly Responses **will not** earn the point if they • use the `isBonus` return value incorrectly	**1 point**
4	Initializes and accumulates appropriate score (*algorithm*)	Responses **can** still earn the point even if they • call methods incorrectly, as long as method calls are attempted • fail to return the score (*return is not assessed*) Responses **will not** earn the point if they • calculate the score total incorrectly • triple the calculated score incorrectly	**1 point**
		Total for part (a)	**4 points**

© 2022 College Board

(b) `playManyTimes`

	Scoring Criteria	Decision Rules	
5	Loops `num` times	Responses **can** still earn the point even if they • return early	**1 point**
6	Calls `play` and `getScore`	Responses **will not** earn the point if they • call either method on an object other than `this` (use of `this` is optional) • include parameters	**1 point**
7	Compares a score to an identified max or to another score	Responses **can** still earn the point even if they • make the comparison outside the loop • call `getScore` incorrectly • fail to call `play` between calls to `getScore`	**1 point**
8	Identifies the maximum score (*algorithm*)	Responses **will not** earn the point if they • fail to initialize the result variable • compare a score to an identified max or to another score outside the loop • fail to call `play` exactly once each time through the loop	**1 point**
9	Returns identified maximum score	Responses **can** still earn the point even if they • calculate the maximum score incorrectly Responses **will not** earn the point if they • assign a value to the identified maximum score without any loop or logic to find the maximum	**1 point**
		Total for part (b)	**5 points**

Question-specific penalties
None

Total for question 1 **9 points**

© 2022 College Board

Applying the Scoring Criteria

Apply the question scoring criteria first, which always takes precedence. Penalty points can only be deducted in a part of the question that has earned credit via the question rubric. No part of a question (a, b, c) may have a negative point total. A given penalty can be assessed only once for a question, even if it occurs multiple times or in multiple parts of that question. A maximum of 3 penalty points may be assessed per question.

1-Point Penalty

v) Array/collection access confusion (`[] get`)
w) Extraneous code that causes side-effect (e.g., printing to output, incorrect precondition check)
x) Local variables used but none declared
y) Destruction of persistent data (e.g., changing value referenced by parameter)
z) Void method or constructor that returns a value

No Penalty

- Extraneous code with no side-effect (e.g., valid precondition check, no-op)
- Spelling/case discrepancies where there is no ambiguity*
- Local variable not declared provided other variables are declared in some part
- `private` or `public` qualifier on a local variable
- Missing `public` qualifier on class or constructor header
- Keyword used as an identifier
- Common mathematical symbols used for operators (× • ÷ ≤ ≥ <> ≠)
- `[]` vs. `()` vs. `<>`
- `=` instead of `==` and vice versa
- `length/size` confusion for array, `String`, `List`, or `ArrayList`; with or without `( )`
- Extraneous `[]` when referencing entire array
- `[i,j]` instead of `[i][j]`
- Extraneous size in array declaration, e.g., `int[size] nums = new int[size];`
- Missing `;` where structure clearly conveys intent
- Missing `{ }` where indentation clearly conveys intent
- Missing `( )` on parameter-less method or constructor invocations
- Missing `( )` around `if` or `while` conditions

Spelling and case discrepancies for identifiers fall under the "No Penalty" category only if the correction can be **unambiguously inferred from context, for example, "ArayList" instead of "ArrayList". As a counterexample, note that if the code declares `"int G=99, g=0;"`, then uses `"while (G < 10)"` instead of `"while (g < 10)"`, the context does **not** allow for the reader to assume the use of the lower case variable.*

© 2022 College Board

(a) `getAverageRating`

	Scoring Criteria	Decision Rules	
1	Initializes and accumulates sum	Response **can** still earn the point even if they • fail to use a loop to accumulate • fail to call `getRating` or call `getRating` incorrectly	**1 point**
2	Accesses every element of `allReviews` (*no bounds errors*)	Responses **will not** earn the point if they • access the elements of `allReviews` incorrectly	**1 point**
3	Computes and returns `double` average rating based on `getRating` return values (*algorithm*)	Response **can** still earn the point even if they • fail to initialize the accumulator for the sum Responses **will not** earn the point if they • fail to accumulate the sum of all ratings • use integer division to compute average • include parameters on call to `getRating` • fail to call `getRating` on all elements of `allReviews`	**1 point**
		Total for part (a)	**3 points**

© 2022 College Board