2. This question involves the design of a class that will be used to produce practice problems. The following `StudyPractice` interface represents practice problems that can be used to study some subject.

```
public interface StudyPractice
{
    /** Returns the current practice problem. */
    String getProblem();

    /** Changes to the next practice problem. */
    void nextProblem();
}
```

The `MultPractice` class is a `StudyPractice` that produces multiplication practice problems. A `MultPractice` object is constructed with two integer values: *first integer* and *initial second integer*. The first integer is a value that remains constant and is used as the first integer in every practice problem. The initial second integer is used as the starting value for the second integer in the practice problems. This second value is incremented for each additional practice problem that is produced by the class.

For example, a `MultPractice` object created with the call `new MultPractice(7, 3)` would be used to create the practice problems `"7 TIMES 3"`, `"7 TIMES 4"`, `"7 TIMES 5"`, and so on.

In the `MultPractice` class, the `getProblem` method returns a string in the format of `"`*first integer* `TIMES` *second integer*`"`. The `nextProblem` method updates the state of the `MultPractice` object to represent the next practice problem.

The following examples illustrate the behavior of the `MultPractice` class. Each table shows a code segment and the output that would be produced as the code is executed.

Example 1

Code segment	Output produced
`StudyPractice p1 = new MultPractice(7, 3);` `System.out.println(p1.getProblem());`	`7 TIMES 3`
`p1.nextProblem();` `System.out.println(p1.getProblem());`	`7 TIMES 4`
`p1.nextProblem();` `System.out.println(p1.getProblem());`	`7 TIMES 5`
`p1.nextProblem();` `System.out.println(p1.getProblem());`	`7 TIMES 6`

Example 2

Code segment	Output produced

`StudyPractice p2 = new MultPractice(4, 12);`
`p2.nextProblem();`
`System.out.println(p2.getProblem());`
`System.out.println(p2.getProblem());`
	`4 TIMES 13`
	`4 TIMES 13`

`p2.nextProblem();`
`p2.nextProblem();`
`System.out.println(p2.getProblem());`
	`4 TIMES 15`

`p2.nextProblem();` `System.out.println(p2.getProblem());`	`4 TIMES 16`

Question 2 continues on page 10.

Interface information for this question

```
public interface StudyPractice

String getProblem()
void nextProblem()
```

Write the complete `MultPractice` class. Your implementation must be consistent with the specifications and the given examples.

3. This question involves analyzing and modifying a string. The following `Phrase` class maintains a phrase in an instance variable and has methods that access and make changes to the phrase. You will write two methods of the `Phrase` class.

```
public class Phrase
{
   private String currentPhrase;

   /** Constructs a new Phrase object. */
   public Phrase(String p)
   {  currentPhrase = p;  }


   /** Returns the index of the nth occurrence of str in the current phrase;
    *  returns -1 if the nth occurrence does not exist.
    *  Precondition: str.length() > 0 and n > 0
    *  Postcondition: the current phrase is not modified.
    */
   public int findNthOccurrence(String str, int n)
   {  /* implementation not shown */  }


   /** Modifies the current phrase by replacing the nth occurrence of str with repl.
    *  If the nth occurrence does not exist, the current phrase is unchanged.
    *  Precondition: str.length() > 0 and n > 0
    */
   public void replaceNthOccurrence(String str, int n, String repl)
   {  /* to be implemented in part (a) */  }


   /** Returns the index of the last occurrence of str in the current phrase;
    *  returns -1 if str is not found.
    *  Precondition: str.length() > 0
    *  Postcondition: the current phrase is not modified.
    */
   public int findLastOccurrence(String str)
   {  /* to be implemented in part (b) */  }


   /** Returns a string containing the current phrase. */
   public String toString()
   {  return currentPhrase;  }
}
```

Part (a) begins on page 12.

AP® COMPUTER SCIENCE A
2017 SCORING GUIDELINES

Question 2: MultPractice

Class: `MultPractice`	**9 points**

Intent: *Define implementation of class to produce multiplication practice problems*

+1 Declares header: `public class MultPractice implements StudyPractice`

+1 Declares all necessary `private` instance variables

+2 Constructor

- **+1** Declares header: `public MultPractice(int __, int __)`
- **+1** Initializes all instance variables using parameters

+3 `getProblem` method

- **+1** Declares header: `public String getProblem()`
- **+1** Builds string with current values of instance variables
- **+1** Returns constructed string

+2 `nextProblem` method

- **+1** Declares header: `public void nextProblem()`
- **+1** Updates instance variable(s) to reflect incremented second number

AP® COMPUTER SCIENCE A
2017 SCORING GUIDELINES

Question 2: Scoring Notes

Class `MultPractice`			**9 points**
Points	Rubric Criteria	Responses earn the point if they ...	Responses will not earn the point if they ...
+1	Declares header: `public class MultPractice implements StudyPractice`	• omit keyword `public`	• declare class `private`
+1	Declares all necessary `private` instance variables	• declare the unchanging instance variable as `final`	• declare variables as `static` • omit keyword `private`
+2	Constructor		
+1	Declares header: `public MultPractice (int ___, int ___)`	• omit keyword `public`	
+1	Initializes all instance variables using parameters		• fail to declare nonlocal variables • initialize local variables instead of instance variables • assign variables to parameters
+3	`getProblem` method		
+1	Declares header: `public String getProblem()`		• fail to declare method `public`
+1	Builds string with current values of instance variables	• write appropriate code in a method other than `getProblem` • make capitalization or spacing errors	• fail to declare nonlocal variables • fail to use instance variables • miscast `(String) intVar` • call `intVar.toString()`
+1	Returns constructed string		• return a literal string
+2	`nextProblem` method		
+1	Declares header: `public void nextProblem()`		• fail to declare method `public`
+1	Updates instance variable(s) to reflect incremented second number		• fail to declare non-local variables

AP® COMPUTER SCIENCE A
2017 SCORING GUIDELINES

Question 2: MultPractice

```
public class MultPractice implements StudyPractice
{
      private int first;
      private int second;

      public MultPractice(int num1, int num2)
      {
            first = num1;
            second = num2;
      }

      public String getProblem()
      {
            return first + " TIMES " + second;
      }

      public void nextProblem()
      {
            second++;
      }
}
```

These canonical solutions serve an expository role, depicting general approaches to solution. Each reflects only one instance from the infinite set of valid solutions. The solutions are presented in a coding style chosen to enhance readability and facilitate understanding.