2. This question involves a scoreboard for a game. The game is played between two teams who alternate turns so that at any given time, one team is active and the other team is inactive. During a turn, a team makes one or more plays. Each play can score one or more points and the team's turn continues, or the play can fail, in which case no points are scored and the team's turn ends. The `Scoreboard` class, which you will write, is used to keep track of the score in a game.

   The `Scoreboard` class contains a constructor and two methods.

   - The constructor has two parameters. The first parameter is a `String` containing the name of team 1, and the second parameter is a `String` containing the name of team 2. The game always begins with team 1 as the active team.
   - The `recordPlay` method has a single nonnegative integer parameter that is equal to the number of points scored on a play or `0` if the play failed. If the play results in one or more points scored, the active team's score is updated and that team remains active. If the value of the parameter is `0`, the active team's turn ends and the inactive team becomes the active team. The `recordPlay` method does not return a value.
   - The `getScore` method has no parameters. The method returns a `String` containing information about the current state of the game. The returned string begins with the score of team 1, followed by a hyphen (`"-"`), followed by the score of team 2, followed by a hyphen, followed by the name of the team that is currently active.

The following table contains a sample code execution sequence and the corresponding results. The code execution sequence appears in a class other than `Scoreboard`.

Statement	Value Returned (blank if none)	Explanation
`String info;`		
`Scoreboard game = new Scoreboard("Red", "Blue");`		`game` is a new `Scoreboard` for a game played between team 1, whose name is `"Red"`, and team 2, whose name is `"Blue"`. The active team is set to team 1.
`info = game.getScore();`	`"0-0-Red"`	
`game.recordPlay(1);`		Team 1 earns 1 point because the game always begins with team 1 as the active team.
`info = game.getScore();`	`"1-0-Red"`	
`game.recordPlay(0);`		Team 1's play failed, so team 2 is now active.
`info = game.getScore();`	`"1-0-Blue"`	
`info = game.getScore();`	`"1-0-Blue"`	The score and state of the game are unchanged since the last call to `getScore`.
`game.recordPlay(3);`		Team 2 earns 3 points.
`info = game.getScore();`	`"1-3-Blue"`	
`game.recordPlay(1);`		Team 2 earns 1 point.
`game.recordPlay(0);`		Team 2's play failed, so team 1 is now active.
`info = game.getScore();`	`"1-4-Red"`	
`game.recordPlay(0);`		Team 1's play failed, so team 2 is now active.
`game.recordPlay(4);`		Team 2 earns 4 points.
`game.recordPlay(0);`		Team 2's play failed, so team 1 is now active.
`info = game.getScore();`	`"1-8-Red"`	
`Scoreboard match = new Scoreboard("Lions", "Tigers");`		`match` is a new and independent `Scoreboard` object.
`info = match.getScore();`	`"0-0-Lions"`	
`info = game.getScore();`	`"1-8-Red"`	

Write the complete `Scoreboard` class. Your implementation must meet all specifications and conform to the examples shown in the preceding table.

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

GO ON TO THE NEXT PAGE.

3. This question involves the manipulation and analysis of a list of words. The following `WordChecker` class contains an `ArrayList<String>` to be analyzed and methods that are used to perform the analysis. You will write two methods of the `WordChecker` class.

```
public class WordChecker
{
   /** Initialized in the constructor and contains no null elements */
   private ArrayList<String> wordList;

   /**
    *  Returns true if each element of wordList (except the first) contains the previous
    *  element as a substring and returns false otherwise, as described in part (a)
    *  Precondition: wordList contains at least two elements.
    *  Postcondition: wordList is unchanged.
    */
   public boolean isWordChain()
   {  /* to be implemented in part (a) */  }

   /**
    *  Returns an ArrayList<String> based on strings from wordList that start
    *  with target, as described in part (b). Each element of the returned ArrayList has had
    *  the initial occurrence of target removed.
    *  Postconditions: wordList is unchanged.
    *      Items appear in the returned list in the same order as they appear in wordList.
    */
   public ArrayList<String> createList(String target)
   {  /* to be implemented in part (b) */  }

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

GO ON TO THE NEXT PAGE.

Question 2: Class — 9 points

Canonical solution

9 points

```
public class Scoreboard
{
   private String team1Name, team2Name;
   private int whoseTurn;
   private int score1, score2;

   public Scoreboard(String team1, String team2)
   {
      team1Name = team1;
      team2Name = team2;
      whoseTurn = 1;
      score1 = 0;
      score2 = 0;
   }

   public void recordPlay(int points)
   {
      if (points == 0)
      {
         if (whoseTurn == 1)
         {
            whoseTurn = 2;
         }
         else
         {
            whoseTurn = 1;
         }
      }
      else
      {
         if (whoseTurn == 1)
         {
            score1 += points;
         }
         else
         {
            score2 += points;
         }
      }
   }

   public String getScore()
   {
      String result = score1 + "-" + score2 + "-";
      if (whoseTurn == 1)
      {
         result += team1Name;
      }
      else
      {
         result += team2Name;
      }
      return result;
   }
}
```

`Scoreboard`

	Scoring Criteria	Decision Rules	
1	Declares class header: `class Scoreboard`	Responses **will not** earn the point if they • declare the class as something other than `public`	**1 point**
2	Declares at least one `private String` instance variable and one `private int` instance variable	Responses **will not** earn the point if they • declare any instance variable `static` • declare a variable outside the class	**1 point**
3	Declares constructor header: `Scoreboard(String ___, String ___)` and constructor initializes both team name instance variables using parameters	Responses **can** still earn the point even if they • declare instance variable(s) outside the class, or in the class within a method or constructor Responses **will not** earn the point if they • fail to declare or initialize instance variables for both team names • declare the constructor as something other than `public`	**1 point**

4	Declares method headers:
	`public void recordPlay(int ___)`
	`public String getScore()`
		Responses **will not** earn the point if they
		• use incorrect method names
		• omit or declare incorrectly either method header
		• omit `public` in either method header or declare either method as something other than `public`
			1 point

5	Recording method checks for parameter value of zero	Responses **can** still earn the point even if they • use a method name inconsistent with the examples, as long as it is recognizably equivalent	**1 point**
6	Recording method increases at least one declared instance variable representing one team's score	Responses **can** still earn the point even if they • declare any instance variable incorrectly, outside the class, or in the class within a method or constructor • use something other than the parameter to update the instance variable • use a method name inconsistent with the examples, as long as it is recognizably equivalent	**1 point**

7	Recording method switches active team	Responses **can** still earn the point even if they • perform the switch in a method other than the recording method • store the switched active team in a local variable, as long as the switch occurs in both active team cases • use a method name inconsistent with the examples, as long as it is recognizably equivalent • perform the switch when the parameter is not zero	**1 point**
8	Recording method adds correct number of points to the active team's score (*algorithm*)	Responses **can** still earn the point even if they • fail to switch active team correctly • declare an instance variable that holds a team's score outside the class, or in the class within a method or constructor • use a method name inconsistent with the examples, as long as it is recognizably equivalent Responses **will not** earn the point if they • switch teams when the parameter is positive • fail to declare an instance variable to track the active team, initialize it incorrectly, or never change its value • add correct number of points for only one team • increase score by something other than the parameter • fail to declare instance variables to hold both teams' scores	**1 point**
9	Accessor method builds and returns specified string (*algorithm*)	Responses **can** still earn the point even if they • fail to declare instance variables and use variables from constructor or methods within the class • use a method name inconsistent with the examples, as long as it is recognizably equivalent Responses **will not** earn the point if they • omit the literal hyphens in the constructed string	**1 point**
			Total for question 2 **9 points**

Alternate canonical:

```
public class Scoreboard
{
   private String team1Name, team2Name;
   private boolean isTeam1Active;
   private int score1, score2;

   public Scoreboard(String team1, String team2)
   {
      team1Name = team1;
      team2Name = team2;
      isTeam1Active = true;
      score1 = 0;
      score2 = 0;
   }

   public void recordPlay(int score)
   {
      if (score == 0)
      {
         isTeam1Active = !isTeam1Active;
      }
      else if (isTeam1Active)
      {
         score1 += score;
      }
      else
      {
         score2 += score;
      }
   }

   public String getScore()
   {
      String result = score1 + "-" + score2 + "-";
      if (isTeam1Active)
      {
         result += team1Name;
      }
      else
      {
         result += team2Name;
      }
      return result;
   }
}
```

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.*