2. This question involves the `SignedText` class, which contains methods that are used to include a signature as part of a string of text. You will write the complete `SignedText` class, which contains a constructor and two methods.

The `SignedText` constructor takes two `String` parameters. The first parameter is a first name and the second parameter is a last name. The length of the second parameter is always greater than or equal to 1.

The `getSignature` method takes no parameters and returns a formatted signature string constructed from the first and last names according to the following rules.

- If the first name is an empty string, the returned signature string contains just the last name.
- If the first name is not an empty string, the returned signature string is the first letter of the first name, a dash `("-")`, and the last name, in that order.

The `addSignature` method returns a possibly revised copy of its `String` parameter. The parameter will contain at most one occurrence of the object's signature, at either the beginning or the end of the parameter. The returned string is created from the parameter according to the following rules.

- If the object's signature does not occur in the `String` parameter of the method, the returned `String` is the value of the parameter with the signature added to the end.
- If the object's signature occurs at the end of the `String` parameter, the returned `String` is the unchanged value of the parameter.
- If the object's signature occurs at the beginning of the `String` parameter, the returned `String` is the value of the original parameter with the signature removed from the beginning and appended to the end of the parameter.

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

Statement	Method Call Return Value (blank if none)	Explanation
`SignedText st1 = new SignedText("", "Wong");`		The `SignedText` object `st1` has an empty first name and last name `"Wong"`.
`String temp = st1.getSignature();`	`"Wong"`	
`SignedText st2 = new SignedText("henri", "dubois");`		The `SignedText` object `st2` has first name `"henri"` and last name `"dubois"`.
`temp = st2.getSignature();`	`"h-dubois"`	
`SignedText st3 = new SignedText("GRACE", "LOPEZ");`		The `SignedText` object `st3` has first name `"GRACE"` and last name `"LOPEZ"`.
`temp = st3.getSignature();`	`"G-LOPEZ"`	
`SignedText st4 = new SignedText("", "FOX");`		The `SignedText` object `st4` has an empty first name and last name `"FOX"`.
`String text = "Dear";`		
`temp = st4.addSignature(text);`	`"DearFOX"`	The signature does not occur in the `addSignature` parameter, so the returned string is the value of the parameter with the signature appended.
`text = "Best wishesFOX";`		
`temp = st4.addSignature(text);`	`"Best wishesFOX"`	The signature occurs at the end of the `addSignature` parameter, so the returned string is the unchanged value of the parameter.

Statement	Method Call Return Value (blank if none)	Explanation
`text = "FOXThanks";`		
`temp = st4.addSignature(text);`	`"ThanksFOX"`	The signature occurs at the beginning of the `addSignature` parameter, so the returned string is the value of the original parameter with the signature removed from the beginning and appended to the end of the parameter.
`text = "G-LOPEZHello";`		
`temp = st3.addSignature(text);`	`"HelloG-LOPEZ"`	The signature occurs at the beginning of the `addSignature` parameter, so the returned string is the value of the original parameter with the signature removed from the beginning and appended to the end of the parameter.

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

3. This question involves pairing competitors in a tournament into one-on-one matches for one round of the tournament. For example, in a chess tournament, the competitors are the individual chess players. A game of chess involving two players is a match. The winner of each match goes on to a match in the next round of the tournament. Since half of the players are eliminated in each round of the tournament, there is eventually a final round consisting of one match and two competitors. The winner of that match is considered the winner of the tournament.

Competitors, matches, and rounds of the tournament are represented by the `Competitor`, `Match`, and `Round` classes. You will write the constructor and one method of the `Round` class.

```java
/** A single competitor in the tournament */
public class Competitor
{
  /** The competitor's name and rank */
  private String name;
  private int rank;

  /**
   * Assigns n to name and initialRank to rank
   * Precondition: initialRank >= 1
   */
  public Competitor(String n, int initialRank)
  { /* implementation not shown */ }

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

Question 2: Class — 9 points

Canonical solution

9 points

```
public class SignedText
{
   private String firstName;
   private String lastName;

   public SignedText(String first, String last)
   {
      firstName = first;
      lastName = last;
   }

   public String getSignature()
   {
      String sig = "";
      if (!firstName.equals(""))
      {
         sig += firstName.substring(0, 1) + "-";
      }
      sig += lastName;
      return sig;
   }

   public String addSignature(String textStr)
   {
      String sig = getSignature();
      int index = textStr.indexOf(sig);

      if (index == -1)
      {
         return textStr + sig;
      }
      else if (index == 0)
      {
          return textStr.substring(sig.length()) + sig;
      }
      else
      {
         return textStr;
      }
   }
}
```

`SignedText`

	Scoring Criteria	Decision Rules	
1	Declares class header: `class SignedText`	Responses **will not** earn the point if they • declare the class as `private` • include extraneous code outside the class • include `()` with or without parameters in the class header	**1 point**
2	Declares all appropriate `private` instance variable(s) and constructor initializes instance variable(s) using appropriate parameters	Responses **can** still earn the point even if they • create and store the signature using only one `String` instance variable Responses **will not** earn the point if they • declare any instance variables `static` • omit `private` in instance variable declaration • declare variables outside the class • declare an instance variable inside the constructor • fail to use either parameter • assign values to instance variables from an unknown source • assign initial value to local variable instead of instance variable, even if the local variables are declared as `private` • assign parameter(s) to variable(s) with incompatible data type • return a value from the constructor • define the constructor inside a method or define a method inside the constructor	**1 point**

3	Declares constructor header:
	`SignedText(String ___,`
	`String ___)`
		Responses **will not** earn the point if they
		• declare the constructor as `private` or `static`
			1 point

4	Declares method headers:
	`public String`
	`getSignature()`
	`public String`
	`addSignature(String ___)`
		Responses **will not** earn the point if they
		• omit `public` in either method header or declare either method as something other than `public`
		• use incorrect method name
		• use incorrect return or parameter type
			1 point

5	Compares first name to the empty string	Responses **can** still earn the point even if they • perform the comparison implicitly (e.g., by comparing the string's length to 0)	**1 point**

6	Determines appropriate signature string in both cases (*algorithm*)	Responses **can** still earn the point even if they • fail to return a value in some cases (*return from* `getSignature` *not assessed*) Responses **will not** earn the point if they • construct the correct string but return something else • define another method inside the signature method or vice versa	**1 point**
7	Calls `String` methods using correct syntax throughout the class	Responses **can** still earn the point even if they • call `substring` only one time • call `indexOf`, `contains`, `charAt`, or other appropriate methods Responses **will not** earn the point if they • only call `length` and/or `equals` without using other `String` methods • fail to call at least one `String` method which accesses elements of a string	**1 point**
8	Identifies the three required cases for `addSignature` using appropriate conditions	Responses **can** still earn the point even if they • return an incorrect string in one or more cases	**1 point**
9	`addSignature` returns appropriate `String` in all three cases (*algorithm*)	Responses **can** still earn the point even if they • identify one or more of the three cases incorrectly, as long as they are unambiguously no-match, match-start, and match-end • implement `getSignature` incorrectly Responses **will not** earn the point if they • call `getSignature` incorrectly, or incorrectly implement its functionality in `addSignature` • fail to identify three distinct cases • build correct strings but assign them to cases incorrectly • print the string instead of or in addition to returning it • define another method inside `addSignature` or vice versa	**1 point**

	Question-specific penalties
	None

Alternate canonical:

```
public class SignedText
{
   private String signature;

   public SignedText(String first, String last)
   {
      signature = last;
      if (first.length() > 0)
      {
         signature = first.substring(0, 1) + "-" + last;
      }
   }

   public String getSignature()
   {
      return signature;
   }

   public String addSignature(String textStr)
   {
      String result = textStr;
      int index = textStr.indexOf(signature);
      if (index < 0)
      {
         result += signature;
      }
      else if (index == 0)
      {
         result = result.substring(signature.length()) + signature;
      }
      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.*

2025 Digital Decision Rules

- Some non-ASCII characters are not printing correctly in ONE, particularly extended punctuation. Certain kinds of double-quotes may display as â⍰⍰; a character that displays as ï¼⍰ may be a parenthesis, comma, or semicolon (or other punctuation). If the badly displayed character makes sense as one of those, evaluate the response accordingly.
- If there are missing closed-double-quotes or closed-parentheses, assume they are at the end of the line where they opened, immediately before the semicolon or curly bracket (if any).
- Assume an open/left curly bracket `{` immediately after any method header or class header that does not already have one.
- Assume an appropriate amount of closing brackets before any method header to close all open brackets from the previous method or constructor.
- If there are missing curly brackets, clear indentation can "convey intent". Evaluate the response accordingly.
- Inside a method with left-justified code, indentation cannot "convey intent", so missing curly brackets cannot be assumed. Without bracketing or indentation, only the first line of a `while` / `if` / `for` is controlled by the statement; with open curly bracket and no indentation cues, the entire remainder of the method is "inside" the statement.

No Penalty

- `:` instead of `;` and vice versa
- `,` instead of `;` and vice versa