Past-paper walk-through

Nested Iteration ▪ 2.11

eXercise

Questions in this set

- 2026 Q3
- 2021 Q1
- 2018 Q2
- 2018 Q4

Reading the mark scheme

- **B1** a mark that stands on its own – the point is either made or not.
- **M1** a *method* mark: the right approach, even if the number is wrong.
- **A1** an *answer* mark, and it usually needs its M mark first.
- **C1** a working mark on the way to the answer.
- // separates answers the examiner accepts equally.
- **ecf** = error carried forward: a later part is marked on your own earlier answer.
- **owtte** = "or words to that effect" – the wording need not match.

Question 3

2026 ▪ Q3

The CourseRecord class is used to store information about a student enrolled in a course. A partial definition of the CourseRecord class is shown.

```
k+kdpublic+w k+kdclass n+ncCourseRecord
p
+w    c+cm/**
c+cm     *  Returns a unique ID for the student associated with this
c+cm     *  CourseRecord
c+cm     */
+w    k+kdpublic+w nString+w n+nfgetStudentIDp(p)
+w    p+w c+cm/* implementation not shown */+w p
+w    c+cm/**
c+cm     *  Returns a nonnegative integer representing the number of times the
c+cm     *  student associated with this CourseRecord has been absent in the
c+cm     *  course
c+cm     */
+w    k+kdpublic+w k+ktint+w n+nfgetAbsencesp(p)
+w    p+w c+cm/* implementation not shown */+w p

+w    c+cm/* There may be instance variables, constructors, and methods
```

Question 3

The Attendance class maintains two ArrayLists of CourseRecord objects, named historyList and mathList. A partial definition of the Attendance class is shown.

```
k+kdpublic+w k+kdclass n+ncAttendance
p
+w    c+cm/* The students enrolled in a history course */
+w    k+kdprivate+w nArrayListonCourseRecordo+w nhistoryListp;

+w    c+cm/* The students enrolled in a math course */
+w    k+kdprivate+w nArrayListonCourseRecordo+w nmathListp;
+w    c+cm/**
c+cm     *  Returns the number of students who are enrolled in both
c+cm     *  the history course and the math course but have more
c+cm     *  absences in the history course than the math course
c+cm     *  Preconditions:
c+cm     *          No student ID appears multiple times in historyList.
c+cm     *          No student ID appears multiple times in mathList.
c+cm     *          historyList and mathList do not contain any null elements.
c+cm     *  Postcondition:
c+cm     *          historyList and mathList are unchanged.
c+cm     */
```

Question 3

Write the `Attendance` method `moreHistoryThanMathAbsences`. The method should return the number of students who are enrolled in both the history course and the math course but have more absences in the history course than the math course.
For example, suppose `historyList` and `mathList` have the following contents.
historyList:

Student ID	**“rc29”**	**“br98”**	**“dr03”**	**“ot32”**	**“sq98”**	**“ry00”**
Number of Absences	1	1	2	2	3	1

Question 3

mathList:

Student ID	**“fr27”**	**“sq98”**	**“dr03”**	**“dk12”**	**“ot32”**	**“js33”**	**“ry00”**
Number of Absences	2	1	2	1	1	0	3

In this example, there are four student IDs (`"dr03"`, `"ot32"`, `"sq98"`, and `"ry00"`) that appear in both `historyList` and `mathList`. Of these, only `"ot32"` and `"sq98"` have more absences in the history course than the math course, so the `moreHistoryThanMathAbsences` method should return 2.

Question 3

Complete method `moreHistoryThanMathAbsences`.

```
c+cm/**
c+cm *  Returns the number of students who are enrolled in both
c+cm *  the history course and the math course but have more
c+cm *  absences in the history course than the math course
c+cm *  Preconditions:
c+cm *          No student ID appears multiple times in historyList.
c+cm *          No student ID appears multiple times in mathList.
c+cm *          historyList and mathList do not contain any null elements.
c+cm *  Postcondition:
c+cm *          historyList and mathList are unchanged.
c+cm */
k+kdpublic+w k+ktint+w n+nfmoreHistoryThanMathAbsencesp(p)
```

Question 1

2021 ▪ Q1

This question involves the `WordMatch` class, which stores a secret string and provides methods that compare other strings to the secret string. You will write two methods in the `WordMatch` class.

```
k+kdpublic+w k+kdclass n+ncWordMatch
p
+w    c+cm/** The secret string. */
+w    k+kdprivate+w nString+w nsecretp;

+w    c+cm/** Constructs a WordMatch object with the given secret string of lowercase letters. */
+w    k+kdpublic+w n+nfWordMatchp(nString+w nwordp)
+w    p
+w        c+cm/* implementation not shown */
+w    p
+w    c+cm/** Returns a score for guess, as described in part (a).
c+cm     *  Precondition: 0  guess.length() = secret.length()
c+cm     */
+w    k+kdpublic+w k+ktint+w n+nfscoreGuessp(nString+w nguessp)
+w    p+w   c+cm/* to be implemented in part (a) */+w   p

+w    c+cm/** Returns the better of two guesses, as determined by scoreGuess and the rules for a
```

Question 1

(a) Write the `WordMatch` method `scoreGuess`. To determine the score to be returned, `scoreGuess` finds the number of times that `guess` occurs as a substring of `secret` and then multiplies that number by the square of the length of `guess`. Occurrences of `guess` may overlap within `secret`.
Assume that the length of `guess` is less than or equal to the length of `secret` and that `guess` is not an empty string.
The following examples show declarations of a `WordMatch` object. The tables show the outcomes of some possible calls to the `scoreGuess` method.

```
WordMatch game = new WordMatch("mississippi");
```

Value of guess	Number of Substring Occurrences	Score Calculation: (Number of Substring Occurrences) x (Square of the Length of guess)	Return Value of game.scoreGuess(guess)
"i"	4	4 * 1 * 1 = 4	4
"iss"	2	2 * 3 * 3 = 18	18
"issipp"	1	1 * 6 * 6 = 36	36
"mississippi"	1	1 * 11 * 11 = 121	121

Question 1

```
WordMatch game = new WordMatch("aaaabb");
```

Value of guess	Number of Substring Occurrences	Score Calculation: (Number of Substring Occurrences) x (Square of the Length of guess)	Return Value of game.scoreGuess(guess)
“a”	4	4 * 1 * 1 = 4	4
“aa”	3	3 * 2 * 2 = 12	12
“aaa”	2	2 * 3 * 3 = 18	18
“aabb”	1	1 * 4 * 4 = 16	16
“c”	0	0 * 1 * 1 = 0	0

Question 1

Complete the scoreGuess method.

```
c+cm/** Returns a score for guess, as described in part (a).
c+cm *  Precondition: 0  guess.length() = secret.length()
c+cm */
k+kdpublic+w k+ktint+w n+nfscoreGuessp(nString+w nguessp)
```

[Class information box for this question, repeated for reference:]

Question 1

```
k+kdpublic+w k+kdclass n+ncWordMatch

k+kdprivate+w nString+w nsecret

k+kdpublic+w n+nfWordMatchp(nString+w nwordp)
k+kdpublic+w k+ktint+w n+nfscoreGuessp(nString+w nguessp)
k+kdpublic+w nString+w n+nffindBetterGuessp(nString+w nguess1p,+w nString+w nguess
```

Question 1

2021 ▪ Q1

This question involves the `WordMatch` class, which stores a secret string and provides methods that compare other strings to the secret string. You will write two methods in the `WordMatch` class.

```
k+kdpublic+w k+kdclass n+ncWordMatch
p
+w    c+cm/** The secret string. */
+w    k+kdprivate+w nString+w nsecretp;

+w    c+cm/** Constructs a WordMatch object with the given secret string of lowercase letters. */
+w    k+kdpublic+w n+nfWordMatchp(nString+w nwordp)
+w    p
+w        c+cm/* implementation not shown */
+w    p
+w    c+cm/** Returns a score for guess, as described in part (a).
c+cm     *  Precondition: 0  guess.length() = secret.length()
c+cm     */
+w    k+kdpublic+w k+ktint+w n+nfscoreGuessp(nString+w nguessp)
+w    p+w    c+cm/* to be implemented in part (a) */+w    p
```

Question 1

(b) Write the `WordMatch` method `findBetterGuess`, which returns the better guess of its two String parameters, `guess1` and `guess2`. If the `scoreGuess` method returns different values for `guess1` and `guess2`, then the guess with the higher score is returned. If the `scoreGuess` method returns the same value for `guess1` and `guess2`, then the alphabetically greater guess is returned.
The following example shows a declaration of a `WordMatch` object and the outcomes of some possible calls to the `scoreGuess` and `findBetterGuess` methods.

```
WordMatch game = new WordMatch("concatenation");
```

Method Call	Return Value	Explanation
`game.scoreGuess("ten")`	9	1 * 3 * 3
`game.scoreGuess("nation")`	36	1 * 6 * 6
`game.findBetterGuess("ten", "nation")`	"nation"	Since `scoreGuess` returns 36 for "nation" and 9 for "ten", the guess with the greater score, "nation", is returned.
`game.scoreGuess("con")`	9	1 * 3 * 3
`game.scoreGuess("cat")`	9	1 * 3 * 3
`game.findBetterGuess("con", "cat")`	"con"	Since `scoreGuess` returns 9 for both "con" and "cat", the alphabetically greater guess, "con", is returned.

Question 1

Complete method `findBetterGuess`.
Assume that `scoreGuess` works as specified, regardless of what you wrote in part (a). You must use `scoreGuess` appropriately to receive full credit.

```
c+cm/** Returns the better of two guesses, as determined by scoreGuess and the rul
c+cm *  tiebreaker that are described in part (b).
c+cm *  Precondition: guess1 and guess2 contain all lowercase letters.
c+cm *                guess1 is not the same as guess2.
c+cm */
k+kdpublic+w nString+w n+nffindBetterGuessp(nString+w nguess1p,+w nString+w nguess
```

Question 1

[Class information box for this question, repeated for reference:]

```
k+kdpublic+w k+kdclass n+ncWordMatch

k+kdprivate+w nString+w nsecret

k+kdpublic+w n+nfWordMatchp(nString+w nwordp)
k+kdpublic+w k+ktint+w n+nfscoreGuessp(nString+w nguessp)
k+kdpublic+w nString+w n+nffindBetterGuessp(nString+w nguess1p,+w nString+w nguess
```

Solution 1

(a) Mark scheme

- Model solution (scoreGuess, 5 pts):
- “‘java
- public int scoreGuess(String guess)
- {
- int count = 0;
- for (int i = 0; i <= secret.length() - guess.length(); i++)
- {
- if (secret.substring(i, i + guess.length()).equals(guess))

Solution 1

- {
- count++;
- }
- }
- return count * guess.length() * guess.length();
- }
- ‘‘‘
- Counts every (possibly overlapping) occurrence of ‘guess‘ as a substring of ‘secret‘, then returns ‘count * guess.length()^2‘.
- Points (1 each, 5 total):

Solution 1

- 1. Compares ‘guess‘ to a substring of ‘secret‘ (calling ‘secret.indexOf(guess)‘ alone still earns this).
- 2. Uses a substring of ‘secret‘ with the correct length (== guess.length()) for the comparison.
- 3. Loops through all necessary substrings of ‘secret‘ with no bounds errors and without skipping overlapping occurrences.
- 4. Counts the number of identified occurrences of ‘guess‘ within ‘secret‘ (in the context of a condition involving both ‘secret‘ and ‘guess‘).
- 5. Calculates and returns the correct final score: must use a loop, compare ‘guess‘ to multiple substrings of ‘secret‘, not count the same matching substring more than once, and use the correct (unchanged) ‘guess‘ length when computing the score.

Solution 1

(b) Mark scheme

- Model solution (findBetterGuess, 4 pts):
- “‘java
- public String findBetterGuess(String guess1, String guess2)
- {
- if (scoreGuess(guess1) > scoreGuess(guess2))
- {
- return guess1;
- }

Solution 1

- if (scoreGuess(guess2) $>$ scoreGuess(guess1))
- {
- return guess2;
- }
- if (guess1.compareTo(guess2) $>$ 0)
- {
- return guess1;
- }
- return guess2;
- }

Solution 1

- “‘
- Returns the guess with the higher scoreGuess() value; if scores tie, returns the alphabetically greater guess (by String.compareTo).
- Points (1 each, 4 total):
- 6. Calls ‘scoreGuess‘ (with parameters, on ‘this‘) to get scores for ‘guess1‘ and ‘guess2‘.
- 7. Compares the two scores using the comparison result in a conditional statement (not just ‘==‘/‘!=‘).
- 8. Determines which of ‘guess1‘/‘guess2‘ is alphabetically greater (reversing the comparison direction is still acceptable; must not reimplement compareTo incorrectly or treat its result as a boolean).

Solution 1

- 9. Returns the correctly identified ‘guess1‘ or ‘guess2‘ in every case (must not reverse a comparison, omit either comparison, or fail to return a guess in some case).

Question 2

2018 ▪ Q2

This question involves reasoning about pairs of words that are represented by the following `WordPair` class.

```
k+kdpublic+w k+kdclass n+ncWordPair
p
+w    c+cm/** Constructs a WordPair object. */
+w    k+kdpublic+w n+nfWordPairp(nString+w nfirstp,+w nString+w nsecondp)
+w    p+w  c+cm/* implementation not shown */+w  p
+w    c+cm/** Returns the first string of this WordPair object. */
+w    k+kdpublic+w nString+w n+nfgetFirstp(p)
+w    p+w  c+cm/* implementation not shown */+w  p

+w    c+cm/** Returns the second string of this WordPair object. */
+w    k+kdpublic+w nString+w n+nfgetSecondp(p)
+w    p+w  c+cm/* implementation not shown */+w  p
p
```

Question 2

You will implement the constructor and another method for the following `WordPairList` class.

```
k+kdpublic+w k+kdclass n+ncWordPairList
p
+w    c+cm/** The list of word pairs, initialized by the constructor. */
+w    k+kdprivate+w nArrayListonWordPairo+w nallPairsp;

+w    c+cm/** Constructs a WordPairList object as described in part (a).
c+cm     *  Precondition: words.length = 2
c+cm     */
+w    k+kdpublic+w n+nfWordPairListp(nStringo[o]+w nwordsp)
+w    p+w  c+cm/* to be implemented in part (a) */+w  p

+w    c+cm/** Returns the number of matches as described in part (b).
c+cm     */
+w    k+kdpublic+w k+ktint+w n+nfnumMatchesp(p)
+w    p+w  c+cm/* to be implemented in part (b) */+w  p
p
```

Question 2

Class information for this question

```
k+kdpublic+w k+kdclass n+ncWordPair

k+kdpublic+w n+nfWordPairp(nString+w nfirstp,+w nString+w nsecondp)
k+kdpublic+w nString+w n+nfgetFirstp(p)
k+kdpublic+w nString+w n+nfgetSecondp(p)

k+kdpublic+w k+kdclass n+ncWordPairList

k+kdprivate+w nArrayListonWordPairo+w nallPairs

k+kdpublic+w n+nfWordPairListp(nStringo[o]+w nwordsp)
k+kdpublic+w k+ktint+w n+nfnumMatchesp(p)
```

Question 2

(a) Write the constructor for the ‘WordPairList‘ class. The constructor takes an array of strings ‘words‘ as a parameter and initializes the instance variable ‘allPairs‘ to an ArrayList of ‘WordPair‘ objects.

A ‘WordPair‘ object consists of a word from the array paired with a word that appears later in the array. The ‘allPairs‘ list contains ‘WordPair‘ objects ‘(words[i], words[j])‘ for every ‘i‘ and ‘j‘, where $0 \leq i < j <$ words.length. Each ‘WordPair‘ object is added exactly once to the list.

The following examples illustrate two different ‘WordPairList‘ objects.

Example 1

“‘java String[] wordNums = ”one”, ”two”, ”three”; WordPairList exampleOne = new WordPairList(wordNums); “‘

After the code segment has executed, the ‘allPairs‘ instance variable of ‘exampleOne‘ will contain the following ‘WordPair‘ objects in some order.

Question 2

“‘ (”one”, ”two”), (”one”, ”three”), (”two”, ”three”) “‘
Example 2
“‘java String[] phrase = ”the”, ”more”, ”the”, ”merrier”; WordPairList exampleTwo = new WordPairList(phrase); “‘
After the code segment has executed, the ‘allPairs‘ instance variable of ‘exampleTwo‘ will contain the following ‘WordPair‘ objects in some order.
“‘ (”the”, ”more”), (”the”, ”the”), (”the”, ”merrier”), (”more”, ”the”), (”more”, ”merrier”), (”the”, ”merrier”) “‘
Class information for this question
“‘java public class WordPair
public WordPair(String first, String second) public String getFirst() public String getSecond()
public class WordPairList

Question 2

private ArrayList<WordPair> allPairs
public WordPairList(String[] words) public int numMatches() “‘
Complete the ‘WordPairList‘ constructor below.
“‘java /** Constructs a WordPairList object as described in part (a). * Precondition: words.length >= 2 */ public WordPairList(String[] words) /* to be implemented in part (a) */ “‘

Question 2

Class information for this question

```
public class WordPair

public WordPair(String first, String second)
public String getFirst()
public String getSecond()

public class WordPairList

private ArrayList<WordPair> allPairs

public WordPairList(String[] words)
public int numMatches()
```

Question 2

Class information for this question

```
public class WordPair

public WordPair(String first, String second)
public String getFirst()
public String getSecond()

public class WordPairList

private ArrayList<WordPair> allPairs

public WordPairList(String[] words)
public int numMatches()
```

Question 2

2018 ▪ Q2

This question involves reasoning about pairs of words that are represented by the following `WordPair` class.

```
k+kdpublic+w k+kdclass n+ncWordPair
p
+w    c+cm/** Constructs a WordPair object. */
+w    k+kdpublic+w n+nfWordPairp(nString+w nfirstp,+w nString+w nsecondp)
+w    p+w  c+cm/* implementation not shown */+w  p

+w    c+cm/** Returns the first string of this WordPair object. */
+w    k+kdpublic+w nString+w n+nfgetFirstp(p)
+w    p+w  c+cm/* implementation not shown */+w  p

+w    c+cm/** Returns the second string of this WordPair object. */
+w    k+kdpublic+w nString+w n+nfgetSecondp(p)
+w    p+w  c+cm/* implementation not shown */+w  p
p
```

Question 2

You will implement the constructor and another method for the following `WordPairList` class.

```
k+kdpublic+w k+kdclass n+ncWordPairList
p
+w    c+cm/** The list of word pairs, initialized by the constructor. */
+w    k+kdprivate+w nArrayListonWordPairo+w nallPairsp;

+w    c+cm/** Constructs a WordPairList object as described in part (a).
c+cm     *  Precondition: words.length = 2
c+cm     */
+w    k+kdpublic+w n+nfWordPairListp(nStringo[o]+w nwordsp)
+w    p+w  c+cm/* to be implemented in part (a) */+w  p

+w    c+cm/** Returns the number of matches as described in part (b).
c+cm     */
+w    k+kdpublic+w k+ktint+w n+nfnumMatchesp(p)
+w    p+w  c+cm/* to be implemented in part (b) */+w  p
p
```

Question 2

Class information for this question

```
k+kdpublic+w k+kdclass n+ncWordPair

k+kdpublic+w n+nfWordPairp(nString+w nfirstp,+w nString+w nsecondp)
k+kdpublic+w nString+w n+nfgetFirstp(p)
k+kdpublic+w nString+w n+nfgetSecondp(p)

k+kdpublic+w k+kdclass n+ncWordPairList

k+kdprivate+w nArrayListonWordPairo+w nallPairs

k+kdpublic+w n+nfWordPairListp(nStringo[o]+w nwordsp)
k+kdpublic+w k+ktint+w n+nfnumMatchesp(p)
```