Past-paper walk-through

if Statements ▪ 2.3

eXercise

Questions in this set

- 2026 Q2
- 2024 Q1
- 2021 Q1

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 2

2026 ▪ Q2

The `Bottle` class, which you will write, represents a bottle that contains liquid. `Bottle` objects are created by calls to a constructor with a `double` parameter that represents the bottle's capacity, in milliliters (mL), which is the maximum amount of liquid in the bottle. Assume that this value will be greater than or equal to 0. When `Bottle` objects are constructed, each is filled to its capacity.

Question 2

The `Bottle` class contains an `updateAmount` method, which updates the amount of liquid in the bottle. This method has a `double` parameter that represents the amount of liquid, in mL, to be removed from the bottle, with a precondition that the parameter is always greater than zero and less than or equal to the amount of liquid remaining in the bottle. If updating the amount of liquid in the bottle would cause it to be less than 25% of the capacity, the bottle is filled and reset to the capacity. The `updateAmount` method returns a `double` that represents the amount remaining in the bottle, in mL, after the amount has been updated.
The following table contains a sample code execution sequence and the corresponding results. The code execution sequence appears in a class other than `Bottle`.

Statement	Return Value (blank if no value)	Explanation
`double amt;`		
`Bottle water = new Bottle(1000.0);`		A `Bottle` object named `water` is constructed with a capacity of 1,000 mL of liquid.
`amt = water.updateAmount(400.0);`	600.0	400 mL are used, so 600 mL remain in the bottle.
`amt = water.updateAmount(100.0);`	500.0	100 mL are used, so 500 mL remain in the bottle.
`amt = water.updateAmount(300.0);`	1000.0	300 mL are used, so 200 mL remain in the bottle. Since this amount is less than 250 mL (25% of the capacity of 1,000 mL), the bottle is immediately filled and reset to the initial amount of 1,000 mL.
`Bottle shampoo = new`		A `Bottle` object named `shampoo` is constructed with a capacity of 40 mL of liquid.

Question 2

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

Question 2

Statement	Return Value (blank if no value)	Explanation
`double amt;`		
`Bottle water = new Bottle(1000.0);`		A `Bottle` object named `water` is constructed with a capacity of 1,000 mL of liquid.
`amt = water.updateAmount(400.0);`	`600.0`	400 mL are used, so 600 mL remain in the bottle.
`amt = water.updateAmount(100.0);`	`500.0`	100 mL are used, so 500 mL remain in the bottle.
`amt = water.updateAmount(300.0);`	`1000.0`	300 mL are used, so 200 mL remain in the bottle. Since this amount is less than 250 mL (25% of the capacity of 1,000 mL), the bottle is immediately filled and reset to the initial amount of 1,000 mL.
`Bottle shampoo =`		A `Bottle` object named `shampoo` is

Question 2

`Bottle shampoo = new Bottle(40.0);`		A Bottle object named shampoo is constructed with a capacity of 40 mL of liquid.
`amt = shampoo.updateAmount(30.0);`	`10.0`	30 mL are used, so 10 mL remain in the bottle.
`amt = shampoo.updateAmount(1.0);`	`40.0`	1 mL is used, so 9 mL remain in the bottle. Since this amount is less than 10 mL (25% of the capacity of 40 mL), the bottle is immediately filled and reset to the initial amount of 40 mL.

Question 1

2024 ▪ Q1

This question simulates birds or possibly a bear eating at a bird feeder. The following Feeder class updates information about how much food is in the bird feeder and simulates how much food is eaten. You will write two methods of the Feeder class.

```
k+kdpublic+w k+kdclass n+ncFeeder
p
+w    c+cm/**
c+cm     * The amount of food, in grams, currently in the bird feeder; initialized in the constructor
c+cm     * always greater than or equal to zero
c+cm     */
+w    k+kdprivate+w k+ktint+w ncurrentFoodp;

+w    c+cm/**
c+cm     * Simulates one day with numBirds birds or possibly a bear at the bird feeder,
c+cm     * as described in part (a)
c+cm     * Precondition: numBirds  0
c+cm     */
+w    k+kdpublic+w k+ktvoid+w n+nfsimulateOneDayp(k+ktint+w nnumBirdsp)
+w    p+w  c+cm/* to be implemented in part (a) */+w  p
+w    c+cm/**
```

Question 1

(a) Write the `simulateOneDay` method, which simulates `numBirds` birds or possibly a bear at the bird feeder for one day. The method determines the amount of food taken from the feeder on this day and updates the `currentFood` instance variable. The simulation accounts for normal conditions, which occur 95% of the time, and abnormal conditions, which occur the other 5% of the time.

Under normal conditions, the simulation assumes that on any given day, only birds visit the feeder and that each bird at the feeder consumes the same amount of food. This standard amount consumed is between 10 and 50 grams of food, inclusive, in 1-gram increments. That is, on any given day, each bird might eat 10, 11, . . . , 49, or 50 grams of food. The amount of food eaten by each bird on a given day is randomly generated and each integer from 10 to 50, inclusive, has an equal chance of being chosen.

Question 1

For example, a run of the simulation might predict that for a certain day under normal conditions, each bird coming to the feeder will eat 11 grams of food. If 10 birds come to the feeder on that day, then a total of 110 grams of food will be consumed.
If the simulated food consumed is greater than the amount of food in the feeder at the beginning of the day, the simulation empties the feeder and the amount of food in the feeder at the end of the day is zero.
Under abnormal conditions, a bear empties the feeder and the amount of food in the feeder at the end of the day is zero.
The following examples show possible results of three calls to `simulateOneDay`.

- Example 1: If the feeder initially contains 500 grams of food, the call `simulateOneDay(12)` could result in 12 birds eating 20 grams of food each, leaving 260 grams of food in the feeder.
- Example 2: If the feeder initially contains 1,000 grams of food, the call `simulateOneDay(22)` could result in a bear eating all the food, leaving 0 grams of food in the feeder.
- Example 3: If the feeder initially contains 100 grams of food, the call `simulateOneDay(5)` could result in 5 birds attempting to eat 30 grams of food each. Since the feeder initially contains less than 150 grams of food, the feeder is emptied, leaving 0 grams of food in the feeder.

Question 1

Complete the `simulateOneDay` method.

```
c+cm/**
c+cm * Simulates one day with numBirds birds or possibly a bear at the bird feede
c+cm * as described in part (a)
c+cm * Precondition: numBirds  0
c+cm */
k+kdpublic+w k+ktvoid+w n+nfsimulateOneDayp(k+ktint+w nnumBirdsp)
```

Question 1

Class information for this question

```
k+kdpublic+w k+kdclass n+ncFeeder
k+kdprivate+w k+ktint+w ncurrentFood
k+kdpublic+w k+ktvoid+w n+nfsimulateOneDayp(k+ktint+w nnumBirdsp)
k+kdpublic+w k+ktint+w n+nfsimulateManyDaysp(k+ktint+w nnumBirdsp,+w k+kti
```

Question 1

2024 ▪ Q1

This question simulates birds or possibly a bear eating at a bird feeder. The following `Feeder` class updates information about how much food is in the bird feeder and simulates how much food is eaten. You will write two methods of the `Feeder` class.

```
k+kdpublic+w k+kdclass n+ncFeeder
p
+w    c+cm/**
c+cm     * The amount of food, in grams, currently in the bird feeder; initialized in the constructor
c+cm     * always greater than or equal to zero
c+cm     */
+w    k+kdprivate+w k+ktint+w ncurrentFoodp;

+w    c+cm/**
c+cm     * Simulates one day with numBirds birds or possibly a bear at the bird feeder,
c+cm     * as described in part (a)
c+cm     * Precondition: numBirds  0
c+cm     */
+w    k+kdpublic+w k+ktvoid+w n+nfsimulateOneDayp(k+ktint+w nnumBirdsp)
+w    p+w  c+cm/* to be implemented in part (a) */+w  p
```

Question 1

(b) Write the `simulateManyDays` method. The method uses `simulateOneDay` to simulate the feeder on at most `numDays` consecutive days. The `numBirds` birds or a bear coming to the feeder on each day of the simulation returns the number of days that birds or a bear found food at the feeder.

Consider the following examples.

Value of `currentFood` and Method Call	Possible Outcomes and Resulting Return Value
`currentFood` = 2400 `simulateManyDays(610, 4)`	Day 1: `simulateOneDay` leaves 2100 grams of food in the feeder. Day 2: `simulateOneDay` leaves 1650 grams of food in the feeder. Day 3: `simulateOneDay` leaves 1500 grams of food in the feeder. Day 4: `simulateOneDay` leaves 1260 grams of food in the feeder. The simulation returns 4 because, on all four days of the simulation, birds or a bear found food at the feeder. The instance variable `currentFood` has the value 1260.
`currentFood` = 250 `simulateManyDays(150, 5)`	Day 1: `simulateOneDay` leaves 150 grams of food in the feeder. Day 2: `simulateOneDay` leaves 0 grams of food in the feeder. The simulation returns 2 because, on two of the five simulated days, birds or a bear found food at the feeder. The instance variable `currentFood` has the value 0.
`currentFood` = 0 `simulateManyDays(10, 3)`	The simulation returns 0 because no food was found at the feeder on any day.

Question 1

Complete the `simulateManyDays` method. Assume that `simulateOneDay` works as intended, regardless of what you wrote in part (a). You must use `simulateOneDay` appropriately in order to receive full credit.

```
c+cm/**
c+cm * Returns the number of days birds or a bear found food to eat at the feeder
c+cm * as described in part (b)
c+cm * Precondition: numBirds  0, numDays  0
c+cm */
k+kdpublic+w k+ktint+w n+nfsimulateManyDaysp(k+ktint+w nnumBirdsp,+w k+ktint+w nnu
```

Question 1

Class information for this question

```
k+kdpublic+w k+kdclass n+ncFeeder
k+kdprivate+w k+ktint+w ncurrentFood
k+kdpublic+w k+ktvoid+w n+nfsimulateOneDayp(k+ktint+w nnumBirdsp)
k+kdpublic+w k+ktint+w n+nfsimulateManyDaysp(k+ktint+w nnumBirdsp,+w k+kti
```

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).