Past-paper walk-through

Implementing ArrayList Algorithms ▪ 4.10

eXercise

Questions in this set

- 2026 Q3
- 2025 Q3
- 2024 Q3
- 2023 Q3
- 2022 Q3
- 2021 Q3
- 2019 Q3
- 2018 Q2
- 2017 Q1
- 2016 Q2

Questions in this set

- 2015 Q3
- 2015 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 3

2025 ▪ Q3

This question involves pairing competitors in a tournament for one round of matches. 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 game 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.

Question 3

```
c+cm/** A single competitor in the tournament */
k+kdpublic+w k+kdclass n+ncCompetitor
p
+w    c+cm/** The competitors name and rank */
+w    k+kdprivate+w nString+w nnamep;
+w    k+kdprivate+w k+ktint+w nrankp;

+w    c+cm/**
c+cm     * Assigns n to name and initialRank to rank
c+cm     * Precondition: initialRank = 1
c+cm     */
+w    k+kdpublic+w n+nfCompetitorp(nString+w nnp,+w k+ktint+w ninitialRankp)
+w    p+w c+cm/* implementation not shown */+w p

+w    c+cm/* There may be instance variables, constructors,
c+cm       and methods that are not shown. */
p
```