Java Advanced Course

Challenge 33. Terminal operations

Terminal operations

In this challenge, you'll be using the terminal operations shown on this slide.

You'll be using these operations to answer questions about a series of students.

It contains a Student class with demographic data.

This class has a factory method, to generate a new instance of a Student using random data.

This factory method will also generate some random activity, for each course passed as an argument to the Student constructor.

Return type	Terminal operations
long	count()
DoubleStatistics	summaryStatistics()
boolean	allMatch(Predicate<? super T> predicate)
boolean	anyMatch(Predicate<? super T> predicate)
boolean	noneMatch(Predicate<? super T> predicate)

Terminal operations

Create a source for a stream of Students.

Use the static method on Student as the Supplier.

Use a large enough number to get a variety of Student data.

Use a combination of the intermediate and terminal operations we've covered so far, to answer the following questions.

- How many male and female students are in the group.
- How many students fall into the three age ranges, less than age 30, between 30 and 60, over 60 years old. Use summaryStatistics on the student's age, to get a better idea of how old the student population is.

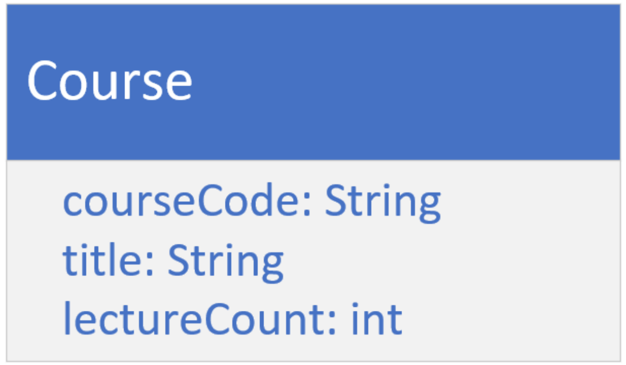
Terminal operations

- What countries are the students from? Print a distinct list of the country codes.
- Are there students that are still active and have been enrolled for more than 7 years? Use one of the match terminal operations to answer this question.
- Next, select 5 of the students above and print their information out.

The Course

The course type should have a course code, a course title, and a lecture count.

You can make this an immutable class.



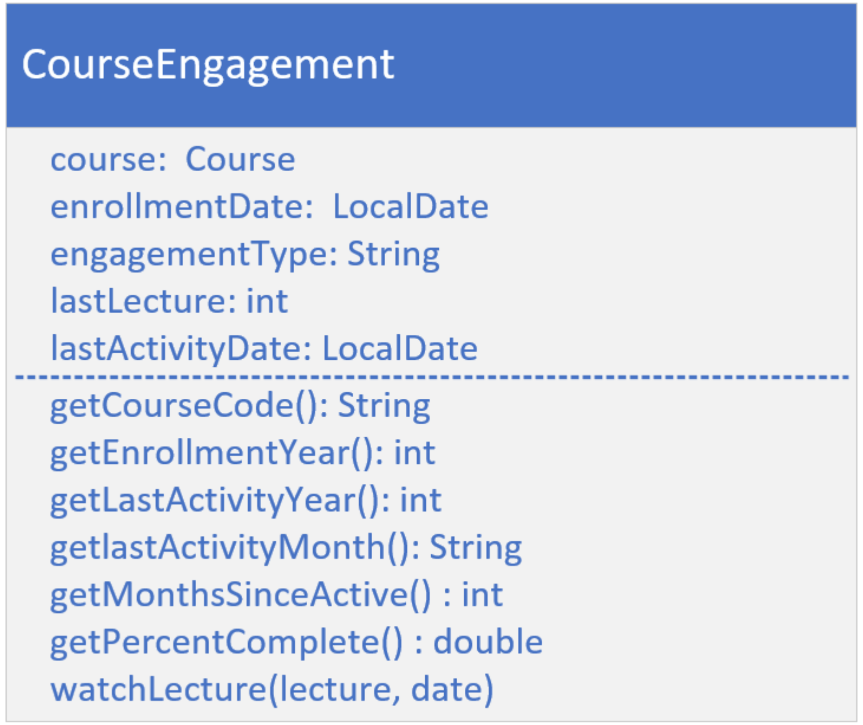
The Course Engagement type

Each student will have a course engagement instance, for every course they're enrolled in.

It should have the fields for the course, the enrollment date, the engagement type, the last lecture, and the last activity date.

It should have the usual getters, plus getters for calculated fields as shown here.

The getMonthsSinceActive method should return the months elapsed, since the last course activity.



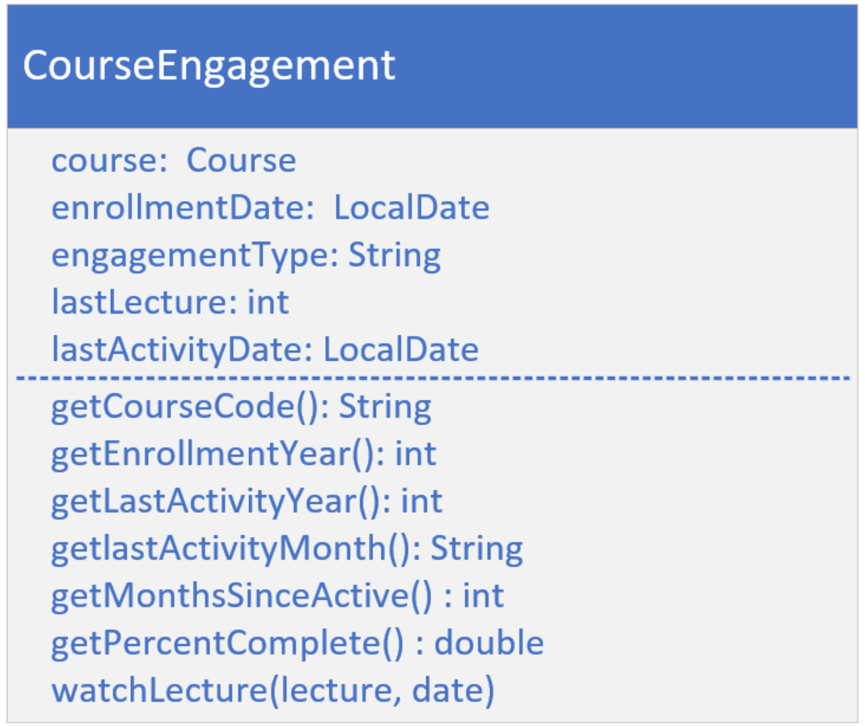
```
CourseEngagement
--------------------------------------------
course:  Course
enrollmentDate:  LocalDate
engagementType: String
lastLecture: int
lastActivityDate: LocalDate
--------------------------------------------
getCourseCode(): String
getEnrollmentYear(): int
getLastActivityYear(): int
getlastActivityMonth(): String
getMonthsSinceActive() : int
getPercentComplete() : double
watchLecture(lecture, date)
```

The Course Engagement type

The getPercentComplete method should use the last lecture, and the lecture count on course, to return a percentage complete.

The watchLecture method would get called when a student engaged in the course and should update fields on the engagement record.

It takes a lecture number, and an activity date.



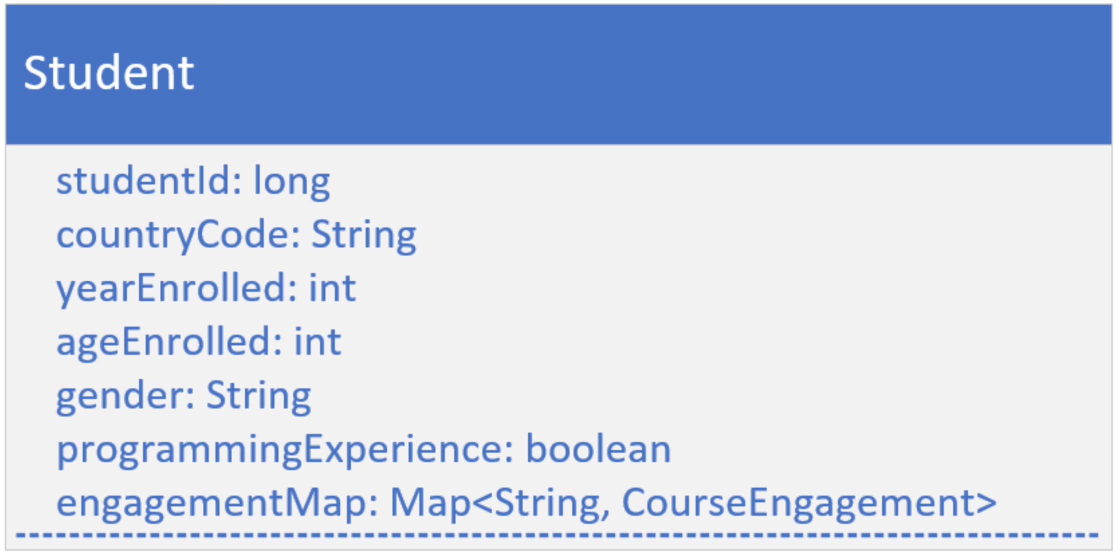
```
CourseEngagement

course:  Course
enrollmentDate:  LocalDate
engagementType: String
lastLecture: int
lastActivityDate: LocalDate
------------------------------------------
getCourseCode(): String
getEnrollmentYear(): int
getLastActivityYear(): int
getlastActivityMonth(): String
getMonthsSinceActive() : int
getPercentComplete() : double
watchLecture(lecture, date)
```

The Student class's attributes

The Student class should have a student id, and demographic data, like country code, year enrolled, age at time of enrollment, gender, and a programming experience flag.

Your student should also have a map of CourseEngagements, keyed by course code.

Include getters for all these fields.



```
Student

studentId: long
countryCode: String
yearEnrolled: int
ageEnrolled: int
gender: String
programmingExperience: boolean
engagementMap: Map<String, CourseEngagement>
```

The Student class's behavior

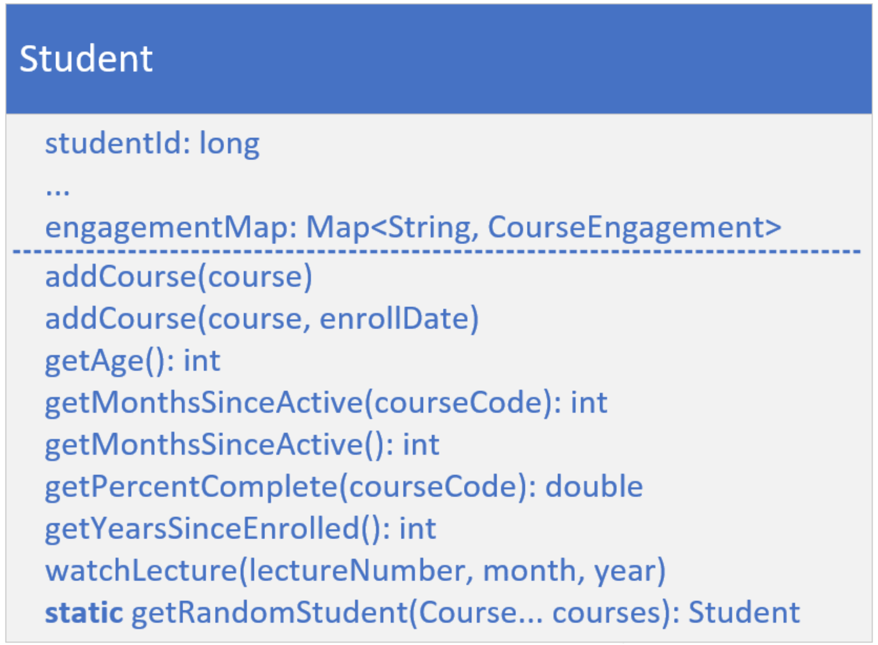
In addition to the usual getters, add getter methods for calculated fields, like getYearsSinceEnrolled and getAge.

Include the getters, getMonthsSinceActive and getPercentComplete, that take a course code and return data from the matching CourseEngagement record.

Add an overloaded version of getMonthsSinceActive, to get the least number of inactive months, from all courses.

You should have overloaded addCourse methods, one that takes a specified activity date, and one that will instead default to the current date.

Include the method watchLecture, that takes a course code, a lecture number and an activity year and month, and calls the method of the same name, on the course engagement record.



Student

studentId: long
...
engagementMap: Map<String, CourseEngagement>

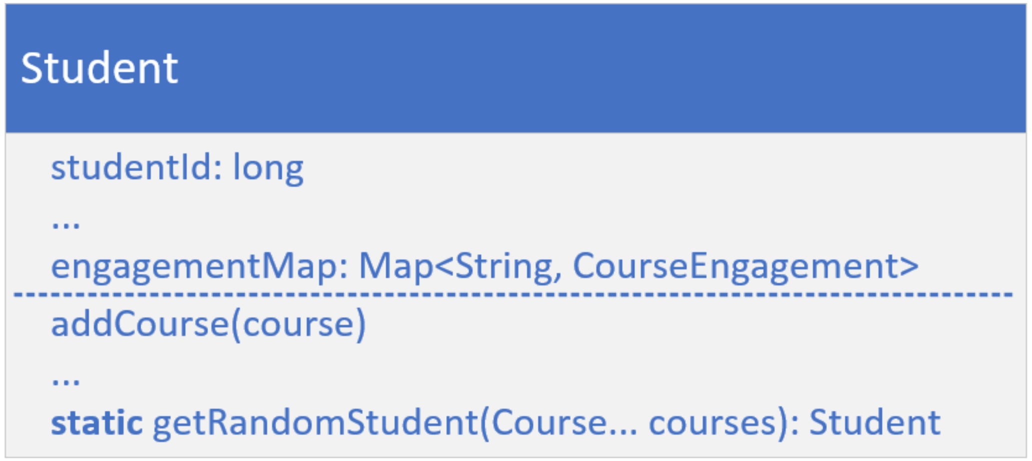
addCourse(course)
addCourse(course, enrollDate)
getAge(): int
getMonthsSinceActive(courseCode): int
getMonthsSinceActive(): int
getPercentComplete(courseCode): double
getYearsSinceEnrolled(): int
watchLecture(lectureNumber, month, year)
static getRandomStudent(Course... courses): Student

The Student random generation method

Finally, create a static factory method on this class, getRandomStudent, that will return a new instance of Student, with random data, populating a student's fields.

Make sure to pass courses to this method and pass them to the student constructor.

For each course, call watchLecture with a random lecture number, and activity year and month, so that each Student will have a different activity for each course.



```
Student
studentId: long
...
engagementMap: Map<String, CourseEngagement>
-------------------------------------------
addCourse(course)
...
static getRandomStudent(Course... courses): Student
```