Project 1: Online Course Experience DB

1 Understand

1.1 Know the Objectives

After completing this project, you should have a have a solid understanding of this coding tech:

- ArrayList (usage and internals)
- Iterator and Iterable interfaces
- Generics (coding in a collection)
- Creating and using a record
- Using a GUI created in Java Swing

1.2 Understand the Problem

We want to create an OnlineCourseDatabase class that reads and flexibly handles data regarding student experiences with online courses.

"Flexible" means that we assume that some options listed in the files may change in the future; more might be added, for example, to dropout reasons; we need to code accordingly so our code is insulated against such future changes. We'll scan the file and create non-duplicated sub-lists for each field (column) that contains such data, doing this before reading and creating records (rows) of student result data.

2 Design

Before you code, work in groups of 3–4 students to create design docs. One group member should submit the designs, listing the names of all group members. Update designs per feedback until they are approved. Then, *on your own*, use them during the rest of the development process and submit them as part of your project (see *Submission Guide*). Use Violet, which creates the diagram types we need, along with others.

2.1 Draw a Class Diagram

Draw and submit a class diagram. Include these classes/interfaces: Main, ArrayList, CourseData, CourseStats, OnlineCourseDatabaseInterface, OnlineCourseDatabase, CourseDataFilteringGui (include only the constructor and related fields). Remember that a record in Java is just a shortcut for creating a class; document them as full classes. For ArrayList, document nested (internal) classes and related interfaces (we're learning internals!). Document all has-a relationships with appropriate arrows.

2.2 Draw an Object Diagram

Draw and submit an object diagram for this project. Start with *Main*. Capture the moment at which all elements have been added to the collection, and the GUI has been instantiated. Remember that a record in Java is just a shortcut for creating a class; document them as the classes they become. There are many has-a relationships involved; keep drawing objects until you run out of them. Use "…" to show omissions for brevity but make sure the reader sees enough examples that they understand the model.

3 Code

3.1 Do Use These

- ArrayList (author code, Ch. 15, ArrayList.java)
- Generics (with linter checks on usage)
- Iterator and Iterable interfaces
- GUI class (instructor provided)

3.2 Don't Use These (violations will result in work being rejected for rework)

- AI tools inside or outside your IDE. Please do not have the AI Plugin installed in IntelliJ.
- @SuppressWarnings, except in the spots discussed in class or in project instructions.
- Stream operations (including stream(), asList(), and similar methods).

3.3 Write Preconditions

Write preconditions where they make sense, especially where Java won't throw a reasonable exception. Always think about these, even if the project doesn't specifically mention them, even in helper functions.

3.4 Implement Other Requirements

3.4.1 ArrayList (using author code as discussed above)

- Add a *toArray* method that takes an array parameter: `public E[] toArray(E[] array) {…}`
  - Use the Java ArrayList API on *toArray* and follow the requirements listed there, paying special attention to what Java does with argument arrays that are too small, just right, or too big.
- Add for/each loop support via the Iterable interface.
- Make no other changes to this class's code.

3.4.2 CourseData (record)

Data for each course experience response will be stored in a record called *CourseData*. The record should have the following data, with these exact data types and orders: userId (String), experienceLevel, courseType, platform, hoursPerWeek, courseDuration, completionStatus, completionPercentage, dropoutReason, satisfactionScore (all byte).

3.4.3 CourseStats (record)

This record is used to package up data to return from a specific method in the *OnlineCourseDatabase* class. The record should have the following data, with these exact data types and orders: recordCount (int), avgHrsPerWeek, avgCourseDuration, avgCompletionPercent, avgSatisfactionScore (all double).

3.4.4 OnlineCourseDatabase

This class must implement the provided interface *OnlineCourseDatabaseInterface*. Instructions for writing the public methods are documented in the interface, via JavaDoc notation.[1]

3.4.4.1 Constructor

- Write a constructor that takes a single parameter, a *File* reference (not a String) to the file it should use. Mark the constructor as throwing FileNotFoundException (reasonable, if it can't load its data).
- Create and populate option sub-lists for each of these columns: experience level, course type, platform, completion status, dropout reason. Ensure only non-duplicated items are added to the lists, e.g., "Tech" and "Non-Tech" appear many times in the file, but we only want *one entry* for each in the sub-list. These will be tiny lists, under ten elements usually, so instantiate them accordingly, with a starting capacity of just five elements.
- Then read all data from the file, creating *CourseData* objects and storing them. Note that for sub-list data, it will store *indexes*, and not *strings*, to save space and object overhead.

[1] I recommend you let your IDE stub required methods and bring in JavaDoc, when you initially code the interface implementation. Ask in class if you want a demonstration.

3.4.4.2 calcFilteredAverages

This is the most challenging method in the class. It receives arguments for indexes into the option fields, filters based on those choices, then creates and returns a CourseStats object representing statistics about the results. Ask in class if you don't fully understand the method requirements or *CourseStats* usage.

3.4.5 Generics

When working with generics, it is likely you will generate Xlint warnings to fix. *Do not suppress these* unless you are explicitly given permission; the inclination to suppress is a bad/scary habit to get into.

3.4.6 Use the Interface(s)

Use the provided interface(s) for the project. Your class(es) should implement these and must exactly follow the signatures in the interfaces. Add helper methods to your classes, as needed. *Do not alter the interface in any way; the original interface must work with your code, or your submission will be rejected and returned for rework, with a 5% penalty.* Consider marking such files as read-only in your OS.

3.4.7 General Coding

As always, keep access minimal (e.g., prefer private to public, constants to variables). Use statics when they make sense (e.g., for constants and methods that don't need instance data). Write helper methods when they reduce code duplication. If methods get too large, break them up where possible (rule of thumb: one method should take up no more than two pages on screen).

4 Document

4.1 Follow the Style Guide

Follow the Course Style Guide, which is linked in the Reference section of the Modules list in Canvas. Failure to do so will result in the loss of points.

4.2 Write JavaDoc

Write complete JavaDoc for these classes: *OnlineCourseDatabase*, *CourseData*, *CourseStats*. This means that an -Xdoclint:all comes back with no errors or warnings[2]. Unfixed warnings will cause deductions.

5 Test

5.1 Write Unit Tests

Write unit tests for the *OnlineCourseDatabase* class. Create a JUnit5 test class; ensure that all methods are fully tested. Run your tests with code coverage; for full credit, tests must cover 100% of statements.

5.2 Verify Code Meets Acceptance Criteria

Work will be rejected and returned for rework if it doesn't meet these acceptance criteria:

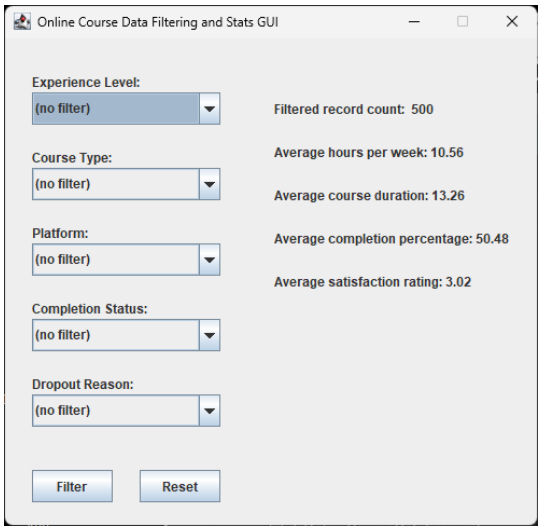
- …compiles with all required Xlint and Xdoclint command line additions.
- …runs without throwing any exceptions.
- …has no failing unit tests.

[2] We went over, in class, the process for adding the required compile command line parameters, what is required to build the entire project at once, and where to find the warnings that might be generated. Ask, if you'd like to review this process; you'll need it all quarter.

6 Demonstrate

Instantiate an *OnlineCourseDatabase*, specifying a File pointing at the project-provided data file (onlineCourseDataset.csv). Do not alter the data file provided; your code must work with the project version, or the project will be rejected and returned for rework, with a 5% penalty.

Then instantiate the *CourseDataFilteringGui*, with the *OnlineCourseDatabase* reference as an argument. If everything is working correctly, the GUI should come up, be populated with the correct options, and result in correct stats for the filtering specified by the user. Here's what should appear at startup:



7 Earn Extra Credit: Branch Coverage (1%)

In addition to 100% code statement coverage, ensure that *branch* coverage of the tested production classes is also 100%.

(continues…)

8 Measure Success

Area	Value	Evaluation
Design docs – initial	5%	Did you submit initial design documents, and were they iterated/approved?
Design docs – final	5%	Are these good representations of the final version of the project?
ArrayList	10%	Was the author's ArrayList code adapted to suit our needs? Is the toArray method coded and working? Is the class iterable?
CourseData	5%	Is the record created as specified?
CourseStats	5%	Is the record created as specified?
OnlineCourseDB – general	5%	Are good coding guidelines followed? Is access minimized, code duplication minimized? Are constants used whenever possible? Are methods of reasonable size? Are preconditions implemented where they should be?
OnlineCourseDB – constructor	15%	Does the constructor do everything requested? Are exceptions handled as requested? Is all data loaded correctly?
OnlineCourseDB – getCourseRecordAt & getCourseRecordString	5%	Does getCourseRecordAt return the specified course record? Do getCourseRecordString return the expected information, in the requested format?
OnlineCourseDB – get*Options	5%	Do these methods return the requested data?
OnlineCourseDB – calcFilteredAveragese	10%	Does the method return the correct data for all filtered and non-filtered scenarios? Is it coded in a maintainable way, with helpful styling?
Generics	5%	Is the Xlint run clean on all classes?
Demo	5%	Is the database created properly, and passed to the GUI? Does the GUI appear, populated and working?
Unit tests	5%	Were test classes created for the requested class? Were all methods tested? Do all tests run and pass? Are proper verification methods in place? Are tests organized as requested per unit methodology (typically, one test class per production class, one test method per production method)?
Test line coverage	5%	Do tests cover 100% of the lines in the production class?
JavaDoc	5%	Did you use JavaDoc notation, and use it properly? Is the Xdoclint run clean?
Style	5%	Were elements of style (including the Style Guide) followed?
Extra credit: branch	1%	Do coverage results indicate 100% branch coverage?
Total	**101%**	

9 Submit Work

Follow the course's Submission Guide when submitting your project. There are helpful checklists and reminders there that will help you earn the most possible points from your work.

(continues…)

10 Suggested Plan of Attack

10.1 Getting Started

- Copy provided classes and interfaces into the source folder.
- Copy provided data files into the project folder.

10.2 ArrayList

- Copy the author *ArrayList* class (chapter 15) into the project.
- Make requested changes. Verify they are working correctly; consider writing unit tests just for the portions you write. If you don't know exactly what they should do, please ask.

10.3 Records

- Create the two records, *CourseData* and *CourseStats*.

10.4 OnlineCourseDatabase

- Create the class.
- Implement the provided interface; stub interface-required methods (or let your IDE do that work for you, copying in the JavaDoc while doing so).
- Write the constructor in two parts, (1) gathering the option sub-list data, and (2) loading the course results data. Verify each part before proceeding.
- Write additional methods.
- Write unit tests. It's recommended to do this as you code, rather than waiting until the end. Ensure you're on a solid foundation before moving on with coding.
- Double check code from a maintenance standpoint. Is there repetitive code that should be factored out? Are methods a reasonable length? Have other coding guidelines been observed?
- Write/check JavaDoc. Do style check and a general code cleanup pass.
- Run Xlint and Xdoclint checks. These are super important now that we're working with generics.
- Do extra credit if you have time.

10.5 Main

In a runnable main method, write instantiate the *OnlineCourseDatabase* and pass it to the *CourseDataFilteringGui* constructor. Verify that results are as expected.[3]

[3] How will you know what "correct" means, here? What oracle will you use for manual and unit testing? Let's discuss this in class if you'd like.