Advanced Programming

Lab Sheet 1 – Recap Exercises

Introduction

Welcome to Advanced Programming!

Advanced Programming is a significant step up from the programming you did last year. You will be working in a far less "step by step" form than you did in labs last year, reflecting your development as an independent learner. Advanced Programming builds upon the concepts and techniques you learned last year, so it is important to make sure that your skills have not atrophied over the summer break. In this lab, you will complete a number of short exercises designed to quickly recap much of what you learned in the first year.

Each of the exercises should only take a short while to complete. If you find yourself struggling on any of the exercises, there is some advice on what you should do at home to rectify the problem. Your lab tutor will be able to offer advice on anything you find difficult, and may offer some additional recommendations for learning you can do outside of class to catch up anything that you haven't properly retained from last year.

Software

Did you buy a new computer over the summer, or wipe your old one ready for the new year? If so, this section has the information you need to get the right software for the term.

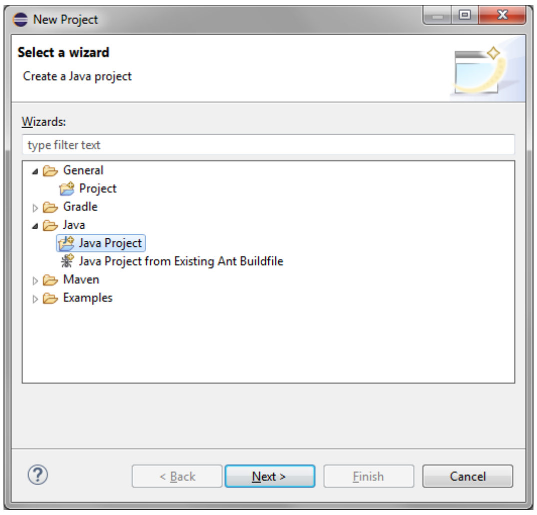
We've upgraded the version of Java we use for teaching from Java 8 to Java 15. Don't worry, everything you learned last year is still applicable! The change does, however, mean that you'll want to install the latest version of eclipse, which is available here. Modern versions of eclipse come complete with a JRE (Java Runtime Environment), which means you no longer need to download and install a JDK (Java Development Kit) from Oracle. The change in versions, plus the resultant need for you to use packages, is discussed in the intro lecture.

Although it is possible to work in your choice of IDE (Integrated Development Environment) if you prefer, I would advise all but particularly strong students to stick with Eclipse, to make moving work to-and-from the lab machines as easy as possible. You don't need any of the exotic eclipse versions: just the standard Eclipse IDE for Java Developers is all you need.

Eclipse Project Setup

Because this is the first lab sheet of the year, and we're using a different version of Java than you're used to, it might be worth a quick recap of the project and class creation process. The screenshots in this section were created using eclipse version 2021-06, which was the most recent version at the time of writing. The process is almost unchanged from what you learned last year, with the exception of needing to select Java 15 as the language version.

You can create a project by Selecting File→New→Project... and finding "Java Project" on the resultant dialog box. The File→New menu remembers frequently-accessed options, so you will likely be able to select that directly in the future. The correct option is depicted in Fig. 1, below.



New Project

Select a wizard

Create a Java project

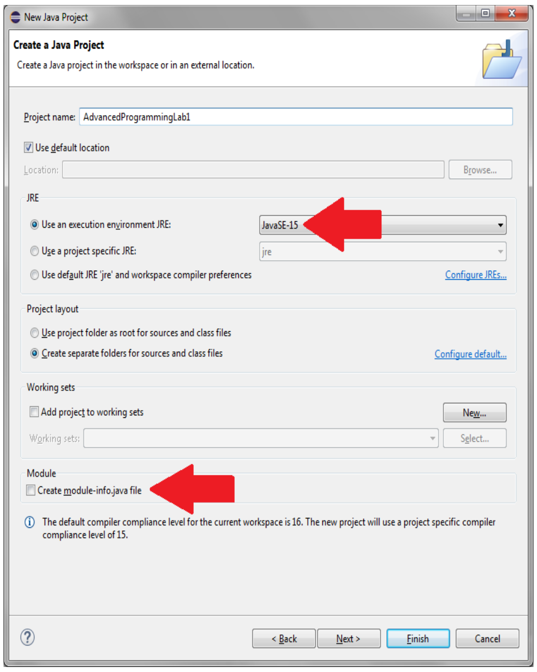
Wizards:

type filter text

- General
  - Project
- Gradle
- Java
  - Java Project
  - Java Project from Existing Ant Buildfile
- Maven
- Examples

Figure 1: Eclipse Project Type Selection Dialog

Once you have selected "Java Project" and clicked "Next", you'll see the familiar project creation dialog. As well as specifying the name of the project to create, there are two settings to keep an eye out for here: the Java version to use, and the option to create a module. You wish to select Java version 15, and to disable the creation of module-info.java, as depicted in Fig. 2, below. In some configurations, eclipse pops up a "create/don't create" dialog for this instead. Don't create one.



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: AdvancedProgrammingLab1

[✓] Use default location

Location:

JRE

[✓] Use an execution environment JRE: JavaSE-15

Use a project specific JRE: jre

Use default JRE 'jre' and workspace compiler preferences

Project layout

Use project folder as root for sources and class files

[✓] Create separate folders for sources and class files

Working sets

[ ] Add project to working sets

Working sets:

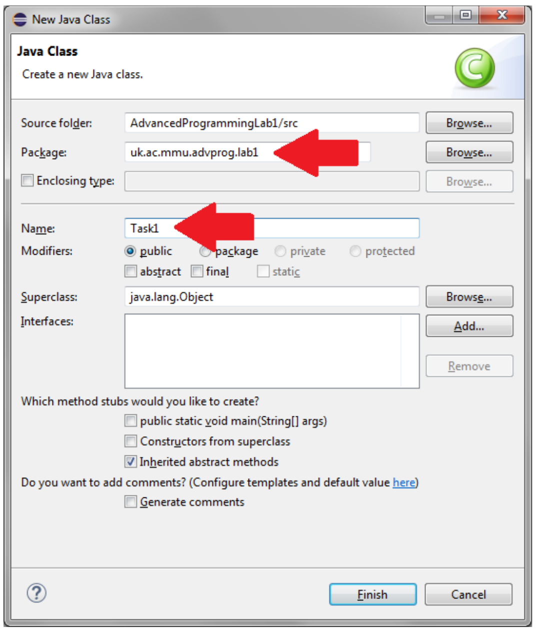
Module

[ ] Create module-info.java file

The default compiler compliance level for the current workspace is 16. The new project will use a project specific compiler compliance level of 15.

Figure 2: Eclipse Project Creation Dialog

Once you have created a Java project, the next step is to add a class to the project's "src" directory. To do this, expand the project in the Package Explorer panel, which is on the left of your eclipse window by default, then right click the "src" directory, and select New→Class. This will show the class creation dialog. You need both to name the class *and to specify the package that it will form part of*. Once you have created a class inside the package, you can avoid repeating the typing by right clicking on the package instead of "src" for future classes. An example is depicted in Fig. 3, below.



New Java Class

Java Class

Create a new Java class.

Source folder: AdvancedProgrammingLab1/src — Browse...

Package: uk.ac.mmu.advprog.lab1 — Browse...

[ ] Enclosing type: — Browse...

Name: Task1

Modifiers: [✓ public] package private protected

[ ] abstract [ ] final [ ] static

Superclass: java.lang.Object — Browse...

Interfaces: — Add... — Remove

Which method stubs would you like to create?

[ ] public static void main(String[] args)

[ ] Constructors from superclass

[✓] Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

[ ] Generate comments

Finish — Cancel

Figure 3: Eclipse Class Creation Dialog

Task 1: Variables and Conditionals

Create an eclipse Java project and add a class with a main() method. Add some variables near the start of the method for storing a person's name and age. Take a look at the various railcards that are available for use on various train operators at railcard.co.uk and write a program that checks the user's age and displays a personalised message telling them what (if any) railcards they should be eligible for.

Too Easy?

Well it is the first exercise of the year! I don't want to go too hard on you...

Some ideas for expanding this task include:

- Have the program prompt the user to type their name and age in
- Add variables to allow the program to cover all of the railcards listed on the site

Too Hard?

If you're struggling with this exercise, that means that you're struggling with the very basics of Java, and that you need to do a lot of work at home to get back up to the standard that was required to pass last year's programming unit. I'd suggest:

- Going back over the lecture materials from the first few weeks last year

- Reading the first few chapters of one of the Java books in the library. Pick whichever one explains things in a way you're most comfortable with. All of the introductory Java books will cover this sort of thing.

- Re-doing some of the lab exercises from the end of the first term last year, when you were just starting to work in Eclipse instead of Processing.

TASK 2: CLASSES AND METHODS

Add another class with a main method to your eclipse project. Add a second new class to the project named UnitMark. This class is going to hold information about the marks a student has achieved in a unit at University.

We're going to simplify the problem a little, and assume that every unit a student studies has just two pieces of coursework (and no exams! Yay!), and that they are weighted 50%/50%. Add some instance variables at the top of the class to hold each of the two marks. Add a constructor to the class that accepts both of the unit marks as parameters, and have the constructor set the value of the instance variables appropriately.

Add a method to the UnitMark class that returns a double. Name the method calculateUnitGrade(). This method should use the two instance variables to calculate the student's overall grade on the unit.

Add another method to the UnitMark class which returns a String. Name the method calculateUnitClassification() and have it return either I, II(I), II(II), III or Fail, if a student gets 70+, 60-70, 50-60, 40-50 or <40, respectively.

In the class with the main method, create 4 UnitMark objects with various grades, and print each of the unit grades and unit classifications to demonstrate that your UnitMark class is working correctly.

Too Easy?

Some suggestions for expanding the task include:

- Find a way to allow units to have different numbers of assessments
- Find a way to allow the assessment weightings to be customised

Too Hard?

If you're struggling with this exercise, it likely means that you haven't quite got the hang of using classes correctly. You'll need to work on this at home quite quickly, because almost everything we'll be doing this term will involve having various classes work together.

I'd suggest:

- Take a look at the relevant chapters in this free e-book, which does a great job of explaining object orientation.
- Going back over the Defenderz portfolio exercise from last year, which was the first time you were introduced to classes and methods. The TiledRoom exercise was also highly relevant.

Task 3: Arrays and Loops

Building on the work in Task 2, add a Student class to your eclipse project. The student needs an instance variable to hold their name and a constructor to allow the name to be set. Add another instance variable named marks, that holds an array of UnitMark objects. Add a setMarks() method to the Student class that accepts an array of UnitMarks as a parameter and sets the instance variable. Add a calculateOverallGrade() method to the student class that uses a for loop to go through the array and calculate the average of all the student's unit grades.

In your main method, create some Students, create some arrays of UnitMarks, assign them to students, and check that the correct average grade is returned for each of the students you have created by printing them to the console.

Too Easy?

That's a good sign! If you'd like to extend this exercise, then I'd suggest:

- Add a method to return the student's overall degree classification for the year

- Many Universities have rules that round up students who narrowly missed a degree classification overall, but who did well in a number of their units. Tweak your classification calculator to award the higher grade to students whose average falls within 2% of a classification boundary, but who achieved the higher classification in at least half of their units. This should not apply to the 40% boundary: students who fail, fail.

Too Hard?

We'll be using a lot of arrays and loops throughout the year, so you'll need to get to a point where you're comfortable working with arrays and lists, particularly in looping through them.

If you just need a quick re-cap on arrays and loops, w3schools have a nice one-page reference here.

If you need some more in-depth content, then chapters 7 and 8 of my preferred introductory Java e-book do a very good job of talking you through things. It is available here.