COMP1815 (2021/22)	**JVM Programming Languages**	**Contribution: 100% of course**
Course Leader: **Dr Markus Wolf**	**Practical Coursework (Hackathon)**	**Deadline Date:** **Thursday 16/12/2021**

Coursework Submission Requirements

- **An electronic copy of your work for this coursework must be fully uploaded on the Deadline Date of** Thursday 16/12/2021 **using the link on the coursework Moodle page for COMP1815.**
- **For this coursework you must submit a single PDF document. In general, any text in the document must not be an image (i.e. must not be scanned) and would normally be generated from other documents (e.g. MS Office using "Save As .. PDF"). An exception to this is hand written mathematical notation, but when scanning do ensure the file size is not excessive.**
- **For this coursework you must also upload a single** ZIP **file containing supporting evidence.**
- **There are limits on the file size (see the relevant course Moodle page).**
- Make sure that any files you upload are virus-free and not protected by a password or corrupted otherwise they will be treated as null submissions.
- Your work will not be printed in colour. Please ensure that any pages with colour are acceptable when printed in Black and White.
- **You must NOT submit a paper copy of this coursework.**
- **All courseworks must be submitted as above. Under no circumstances can they be accepted by academic staff**

The University website has details of the current Coursework Regulations, including details of penalties for late submission, procedures for Extenuating Circumstances, and penalties for Assessment Offences.
See http://www2.gre.ac.uk/current-students/regs

Coursework Specification

This assignment consists of two parts:

- **Part A (implementation) will be completed in a group**
- **Part B (report) must be completed individually**

Please read the entire coursework specification before starting your work.

Timetable Clash Detection System

You have been asked to work on a University Timetable Clash Detection System. The system should load teaching activities for a particular programme of study and check whether there are any clashes in the timetable.

The following information needs to be captured:

- A programme of study can be undergraduate or postgraduate
  - An undergraduate programme runs over three years
  - A postgraduate programme runs over one year
- Each year of study consists of two terms
  - Term 1 – September to December
  - Term 2 – January to April
- A programme of study consists of modules
  - Modules can be compulsory or optional
  - In total, a student will study four modules each term (can be a combination of compulsory and optional modules)
- For each module there can be one or more activities
  - Activities can be lecture, lab, tutorial, seminar or workshop
  - Activities start on the full hour
  - Activities must not start before 9:00 or finish after 21:00
  - Activities can only be scheduled Monday to Friday

When checking for clashes, you would need to consider the year of study (e.g., a year 2 module can run at the same time as a year 1 module without creating a clash).

In a complete timetabling system, students and staff would have to be allocated to activities, but for this assignment we are ignoring these.

The system will need to provide the following functionality:

- Set up/manage modules
- Set up/manage programmes of study
- Timetable activities for modules, according to the rules specified above
- List activities for a particular programme, year and term
- Check for potential clashes (the system must highlight the potential clash)

Currently, the only type of user of the system will be an administrator.

For examples of timetables, you can have a look at the University's programme timetables at https://timetable.gre.ac.uk/.

There are strict requirements about the technologies and architecture to be implemented. These are detailed in the deliverables section.

Deliverables

Part A – Implementation (70%)

Organise yourselves into groups consisting of 3 or 4 students. Exceptionally, you can work individually, but please approach the tutor if you are thinking of doing this. If you cannot find a group to join, your tutors will allocate you to a group.

a) **Java GUI for Project Management – Java (10%):**

   Use Java to create a desktop application with a graphical user interface which enables users to set up and manage modules, programmes of study and timetable activities. It should also be possible to view the timetable of a programme by year and term.

   The application should look pleasant and be easy to use.

b) **Object-Oriented Design - Kotlin (10%)**

   Create domain and entity classes that create an object-oriented structure supporting the Java GUI application. You should apply separation of concern to ensure that the Java GUI application contains only the user-interface related functionality, and all other responsibility is assigned to the domain and entity classes. These classes should be implemented in Kotlin and integrated into the Java GUI

c) **Persistence and Lambda – Kotlin (10%)**

   Implement persistence for the module, programme and activity data, which makes it possible to save this data. It is up to you to decide how you wish to save the information (e.g. save it to file or to database). You should use Lambda expressions to manage the collections of data. This should be implemented in Kotlin and integrated into Java GUI.

d) **Clash Detection - Kotlin (10%)**

   Implement an object-oriented component in Kotlin that can check for potential clashes in a given programme. When clashes are identified, details of the clash should be provided.

e) **Integration – Kotlin/Java (10%)**

   Integrate the clash detection component into the programme management process, so that when a programme timetable is edited, the system automatically checks for clashes in the background. A user should also be notified when a clash happens and details of the clash should be provided..

f) **Clash Detection - Scala (10%)**

You implemented the clash detection algorithm using an object-oriented approach in Kotlin. Now implement a clash detection algorithm using a functional approach in Scala.

g) **Integration – Scala/Kotlin/Java (10%)**

Integrate the Scala implementation of the clash detection algorithm into the Java/Kotlin application you developed. Ideally, it should be possible to choose whether the object-oriented or functional implementation should be used at run-time.

Part B – Report (30%)

The report is to be completed **individually.**

Write a report consisting of **all** the following sections:

- **Section 1. (10%)** – (approx. 700-1,000 words) On this module you have been taught object-oriented and functional programming paradigms, using three different languages (i.e. Kotlin, Scala and Clojure). Critically compare your experience with these different paradigms and languages. You should include discussion of their suitability to different problems and what you perceive their strengths and weaknesses to be.
- **Section 2. (10%)**– (approx. 700-1,000 words) An evaluation of the evolution of your application. You should discuss any problems you had during implementation. You should be critical (both positive and negative) of your implementation. Be prepared to suggest alternatives. You should also include a reflection on how it was to work in a group and of your role within the team. Discuss lessons learnt, what you think went well and what you think could have been improved and how.
- **Section 3. (5%)** Screenshots demonstrating each of the features that you have implemented. Give captions or annotations to explain which features are being demonstrated.
- **Section 4. (5%)** Code listing of any code files you have written. You do not need to include generated code. Please clearly label the code, so it indicates the source file and programming language.

Demonstration (0% but see below)

The demonstration is carried out by the group.

You are required to prepare a brief video showing your implementation (one video per group). The duration should be approximately 10 minutes. There will be a dedicated Panopto coursework submission area – more details on this will be available on Moodle.

Your group must also attend a brief Q&A session with your tutor where you will be asked questions about your implementation and the code.

Failure to demonstrate will result in a failed assessment. You will be allocated a time slot closer to the submission deadline.

Schedule of Submission

Deliverable	**Date**	**Type**	**Group/Individual**
Hackathon	11/11/2021	On campus/MS Team (for students studying online)	Group
Code Q&A	09/12/2021	On campus/MS Team (for students studying online)	Group
Video Demonstration	16/12/2021	Submission on Panopto	Group
Report and Code	16/12/2021	Submission on Moodle	Individual

Each group is encouraged to create a group on MS Teams to enable working as a group. Although you can use this to share files, you are encouraged to use a version control system to support the work as a group.

The hackathon will provide you with an opportunity to engage in an intensive group programming session to progress the implementation of the coursework. The tutors will set aside time blocks on that day when they are able to drop in and advise groups. Details on this will be published on the module's Moodle page.

Notes on the Implementation

You **MUST** upload a ZIP file containing all of your source code (i.e. the folders containing the IntelliJ projects). If the resulting file is too large, then you can delete compiled code and libraries, but do not remove any source code files.

You **MUST** clearly reference code borrowed from sources other than the lecture notes and tutorial examples (e.g. from a book, the web, a fellow student).

Please include a file called “Group List.txt”, which lists all the members of the group, including the name and id of each member.

Notes on the Report

The document should be submitted separately as a PDF document.

Notes on the Demonstration

You will demonstrate the implemented product as a group to your tutor by preparing a 10-minute video. **If you fail to demonstrate your work you will automatically fail the coursework**, irrespective of the quality of the submission. The video must be uploaded via the module’s Panopto assessment submission link. There should be only one video submission per group.

The video you upload on Panopto should follow the follow naming convention – “Surname1,Surname2, etc.”

You should include the surnames of all group members in the video name, so we can easily identify the group.

During the Q&A session you are expected to talk knowledgeably and self-critically about your code.

If you are unable to answer questions about the product you have developed, you will be submitted for plagiarism.

A schedule for the Q&A sessions will be made available on the module’s Moodle page closer to the submission deadline.

Formative Assessment

Groups are encouraged to show the progress made with the implementation in the labs on a weekly basis to get feedback on how the work progresses and, on the design, and implementation decisions taken at every stage of the assignment.

There are also weekly lab exercises, which train you in the knowledge required to complete the assessment. Completed exercises do not need to be uploaded but shown to the tutor for feedback.

Grading Criteria

Please note that the PASS GRADE FOR THIS MODULE IS 40% REGARDLESS OF WHETHER YOU ARE A LEVEL 6 OR LEVEL 7 STUDENT.

The implementation which consists of the group work accounts for 70% and the individual report accounts for 30%. There are multiple functionalities which comprise the development of the system. Each functionality section contributes 10% to the overall grade. The mark for each is awarded taking into consideration the quality and completeness of the implementation, as well as the assessment criteria specified below. Just because you implemented a particular requirement does not mean that you automatically get the full marks. The full marks are only awarded if the requirement has been implemented to outstanding quality, including software design, code quality, user interface, error handling, validation, etc. A poorly structured but working implementation of a requirement would attract a pass mark.

To achieve a pass (40%) you must have made a serious attempt to implement at least four functionality sections. The implementation must show some signs of attempting to focus on the assessment criteria given in this document. A relevant report must also be submitted.

To achieve a 2:2 mark (above 50%) you must have made a serious attempt to implement at least five functionality sections. They must attempt to focus on the assessment criteria given in this document. A good report must also be submitted.

To achieve a 2:1 mark (above 60%) you must have made a serious attempt to implement at least six functionality sections. They must address most assessment criteria given in this document. A very good report must also be submitted.

To achieve a first class (above 70%) you must implement all requirements to a very high standard, or most to an outstanding level, in accordance with the assessment criteria given in this document. Submit an excellent report. Successfully meet most assessment criteria outlined below.

To achieve a very high mark (80% and above) you must implement all implementation requirements to an outstanding standard in accordance with the assessment criteria given in this document. Submit an outstanding report. Successfully meet all assessment criteria outlined below.

NOTE: Failure to do your Demonstration will normally result in you being awarded 0% for the coursework. Even if your implementation and report are excellent and would be awarded a mark of 80% but you don't do a Demonstration then you may score 0% for the coursework.

Assessment Criteria

The Implementation

The following assessment criteria are used to determine the quality of your implementation and should be addressed in the development process:

- If you have incorporated features that were not explicitly asked for, but which add value to the application, they may be considered if you draw our attention to them.
- The application should look pleasant and be easy to use.
- Code structure – does your code demonstrate low coupling and high cohesion? Have you avoided hard coding (i.e. is your code stateless)? Have you reused external components? Have you minimised code duplication? How much impact would a further change of persistence medium have on your application?
- Quality of Design – how flexible is your application? How easy would it be to add in new functionality, or alter the presentation layer, or change the data source?
- Robustness of the application. Have you properly handled errors and validated input? Is there evidence of testing?
- Quality of code –
    - Is the code clear to read, well laid out and easy to understand?
    - Is the code self-documenting? Have you used sensible naming standards?
    - Is your code structure logical?
    - Have you commented appropriately?

The Report

The document should be clear, accurate, complete, and concise. State any assumptions you have made.

- Are all the required sections included and completed properly?
- Does the report give an accurate reflection of what you have achieved?
- Is the report clear and easy read? Does it follow the structure specified?
- Is the evaluation realistic and does it show that you have really thought about your implementation and the specified issues as well as how they may be enhanced to be ready for live deployment. Do you show insight into the complexities of software development?

Demonstration

- You should be able to demonstrate the implementation level achieved in a clear logical and unambiguous manner without encountering errors. You must be able to show knowledge of your code and design choices.

Summative Feedback

Feedback on the final submission will be provided in written format and made available on Moodle within 15 working days of submission.