|  |
| --- |
|  |
| **Assignment Brief 2022/23** |
| **Unit Title:** Introduction to Programming |
| **Unit Code:** 6G4Z1901 |
| **Level:** 4 |
| **Assignment Title:** Record Management System Using Java Programming |
| **Unit Leader:** Connah Kendrick |
| **Contact Details:** Connah.Kendrick@mmu.ac.uk |
| **No of Elements in Assignment:** One element. |
| **Submission Date:** 5th May, 9 pm |
| **Submission Instructions:**   1. A single zip file is to be submitted via Moodle. This zipped folder should contain the following:    1. A sub-folder containing the code and other required files, e.g. input text files.    2. A sub-folder containing the design and testing documentation files (PDF or Word files). 2. The zip file should be named last name, first name and student ID, separated by \_, e.g., "Smith\_John\_1023452.zip".   Note that submissions which are incomplete or do not compile will be subject to penalty. (identify physical and electronic hand-in arrangements if both are required) |
| **Feedback Return Information:** via Moodle, four weeks after submission.(identify mark return and feedback return if handled separately) |
| **Plagiarism Notice:**  Plagiarism is a severe issue, and steps are taken to ensure submissions are original work. It can be defined as submitting code (full or partially) that is not your own, e.g., submitting online repositories or using other people's methods and code segments within your code.  To avoid plagiarism, ensure you only submit your code. Students are encouraged to discuss and talk about their work. However, direct code sharing and copying from the screen fall under plagiarism. Due to the nature of plagiarism, unless a clear indication of the original work is known, both will be classed as plagiarised. Owing to this, it is recommended that computers and laptops be supervised and locked to prevent on-screen copying when unsupervised. Similarly, steps should be taken so pen drives, external hard drives, and online backups (e.g., Dropbox, GitHub) should be secured and not public.  We use an online submission system that compares and assesses similarity against all submissions and online repositories. Cases of plagiarism are penalised with a 0% grade for the assignment. However, an alternative assignment will be given for the opportunity of resubmission during the summer, capped at 40%(pass) of the grade. |
| **Assignment Task Overview:**  Design, implement and test a record management system. The system should be capable of displaying, adding, removing, and editing records via text or file input. The student should choose what type of record is stored, such as music, films, students (for a university database) or customers (in a business setting), based on their interests. Each record must have multiple pieces of the relevant information associated with it, e.g., for music records, a sensible set of fields would be album name, singer, year, and publisher. Similarly, we may expect to see ISBN, title, author, and year for book records using relevant variables.  Assessment Levels: In each case, the achievement of a given level requires that all lower levels also be fulfilled (so, for example, a student aiming for a mark of 65% must complete all the 40-49% **and** 50-59% criteria **before** addressing the criteria in 60-69%). We strongly advise you to complete, test and save your assignment at each level before you advance and make regular backups. This way, you can be sure to have an earlier version to submit. This will prevent last-minute coding errors from lowering your submission. We do not cover using version management software like GitHub in the course. You may use a version manager, given any repositories are kept private. For more information on this, see the plagiarism section above.  In each case, the criteria listed are the minimum requirements for that mark band. Additional marks within a band can be obtained by improving the quality of your code (e.g., indentation, comments, etc.) and by implementing some of the features required for the next level.  Note: There is no restriction on the number of Java classes. However, a good solution would require three or more classes. Using external packages is not required, as we only mark your code; if an external package is used, it is recommended to include it with the submission and any instructions for ensuring its functionality. Similarly, a Graphical User Interface is not required but can be done. Brief overview |
| **Unit Learning Outcomes Assessed:**  State what Unit Learning Outcomes are being covered by the assessment, delete as required.  LO1: Apply the main structuring features of the chosen high-level programming language(s) to solve a variety of problems  LO2: Design well-structured solutions to issues of varying complexity using appropriate methods  LO3: Test well-structured solutions to a variety of problems using the appropriate techniques and a high-level programming language  LO4: Apply an object-oriented design to model a real-world (type) problem  LO5: Use abstraction to create efficient, reusable code, demonstrating an understanding of the complex interactions between the CPU and a high-level programming language. |
| **Apprenticeship Standard Learning Outcomes Covered:**  See KSB mappings on the Moodle page. |
| **Negotiated Assessment:**  If you or your employer wish to pursue a negotiated assessment, then your supervisor should submit a maximum 2-page summary of the proposed alternative assessment to the unit tutor listed on the cover sheet of this assessment.  In writing your summary, please ensure that the learning outcomes listed on the front sheet of this assessment are covered and that the scale, complexity, and level of the work proposed is broadly equivalent with this assessment.  It might be that you are working on a project in the right area, but that the project you are working on is much larger or more complex than this assessment. In this case, it might be possible to submit work which relates to part of the project you are working on, e.g., a subset of the functionality of a piece of software or similar.  If you are not sure of the suitability of an alternative assessment, then your supervisor should speak directly with the unit leader, ideally by phone or Microsoft teams, so that they can quickly establish the feasibility of the negotiated alternative. |
| **Assignment Details and Instructions:**  Record systems are fundamental to a wide range of companies, from restaurants for keeping track of orders to stores to keeping track of stock. These systems follow a similar creation process but require programming knowledge to design, implement and tailor to individual requirements. These systems reduce human workload and allow for redundancy in case of failures, e.g., loss of written documents while helping minimise human error.  The system should allow users to display, create, delete, edit, and search records in its most straightforward format. To access the higher mark bands, you should also consider providing information that can be taken by analysing the records, such as record count, customer loyalty or stock counts. In the assignment, you should make a record system for a subject of your choice; the system should have some real-world implications and feature multiple input fields and one unique ID field; for example, for banking, input fields could be:   * Customer ID (unique to each customer) * Customer name * Accounts * Money available   Note the unique field should be used to prevent duplicate inputs and aid in differentiating between fields with similar values, e.g., customers with identical names or differing book editions, like the ISBN system for books and ID/ number card system for students.  Specification  This assignment requires you to design and implement a Java application using an appropriate IDE to create a record management program for an area of your choice. You must also document the design and testing of the program. The minimum specifications for the assignment are as follows:   1. The program should run from eclipse and print a list of commands: 'display', 'create', 'delete', 'search' and 'exit'. 2. From the initial state, the user will select an option and interact with the program accordingly. Upon completion of the action, the system should return to the initial state allowing for the use of the other options.    1. At a pass level (40% upwards), you should be able to input records and print all records out manually. The programme should exit without error when the 'exit' option is selected.    2. At a 2-2 level (50% upwards), you should be able to create your records via manual input (e.g., via the keyboard whilst being prompted by the user). You should also be able to delete records from the system.    3. A 2-1 level (60% upwards) should (a) prevent duplicate items from being added using the unique field. (b) Search should be implemented at a field level. E.g., for book records, when searching, the input of "Authors" would only search the author fields.    4. A 1st level (70% upwards), You should have added at least two additional features beyond the specification up to this point. These should be specific to the types of records in your application and should be recorded in your design documentation.    5. For above 80%, code should be above what has been asked, providing intuitive user information, error handling and further additional improvements beyond the specification of the task.    6. Please see the marking grid for a complete breakdown of how the code will be graded. 3. All submissions should include a user guide with test values to demonstrate your system's work and all the features you would like to demonstrate. 4. Submission must include your design process, e.g., class diagrams, pseudocode, and demonstration of refinement (bug fixing, testing tables). 5. Submission must include test cases and the evidence logs of the tests; the test should highlight the working configurations of the above criteria, e.g., advanced searching. These should be in a word or PDF document. 6. Although not required, you should look through and highlight where you think you are currently graded in the marking rubric. |
| **Resources**   * Lecture slides, online lecture captures and Moodle videos. * Textbooks: Charatan Q. and Kans A. (2019) Java in two semesters. Shiffman D. and The nature of code. * Completed portfolio exercises and challenge exercises. * Additional weekly resources on Moodle. * Moodle Quizzes. * Codingbat.com. * Lynda.com. |
| **Group Work Guidelines (If applicable, see Moodle):**  N/A |
| **Unit Specification**:  See Moodle |

**Assessment Marking Criteria**

| **Grade Range** | **Coding (40%)** | **OO Design (20%)** | **Design and testing (20%)** | **Functionality and Interface (20%)** |
| --- | --- | --- | --- | --- |
| **Demonstrate the ability to use high-level programming language constructs to solve complex problems** | **Demonstrated the ability to implement programming solutions incorporating the principles of object-oriented design** | **Demonstrate the ability to understand requirements, design accurate algorithms and use testing for checking accuracy and stepwise refinement of the code** | **Demonstrate the ability to create functional programmes with interactive user interfaces** |
| 86%-100% | As below, plus additional (more imaginative) features – beyond the task specification. | As below, plus exceptional use of object-oriented design to achieve maximum abstraction with minimal code length –beyond the task specification. | As below, plus professional-level design and testing documentation –beyond the task specification. | As below, plus a rich user interface – beyond the task specification. |
| 70%-85% | As below, plus some additional features. e.g., outputs are printed alphabetically or numerically. | Excellent use of classes and methods to achieve optimal efficiency, abstraction, and reusability of the entire code. | As below, plus testing for each method incorporating error handling. | As below, plus an intuitive user interface listing items in an order. |
| 60%-69% | As below, plus some additional key features, duplicate checking using unique values and advanced search using filters. | As below, plus multiple interdependent classes are used without redundancy in methods and variables. | As below, plus the pseudocode for each method. | As below plus, a user-friendly Interface gives warnings when entering duplicate fields and allows search by specific fields. |
| 50%-59% | As below, plus the correct results for keyboard input, key features, and items added are typed, and the ability to delete objects is given. | All classes and methods work correctly. Multiple interdependent methods were used. | As below, plus all top-level test cases. | As below, plus the ability to load and export text files. |
| 40%-49% | The programme compiles, loads, displays correct results, and has the option to exit the program. Key features are the ability to input new items and print all items out. | All classes and methods work as expected, except for minor errors (at most, in one or two methods). Two or more classes using none-static context. | Pseudocode accurately describes the overall algorithm with some test cases. | As below, plus the functionality to enter dynamic input through the keyboard |
| 35%-39% | The programme compiles, loads, and operates with minor errors but allows entry of most record fields and displays near-correct results. | Almost all classes and methods work as expected, but there are one or two minor errors. At least two classes, a record and a main, with some attempts at initialisation and class methods. | A correct pseudocode description of the overall algorithm with some test cases. | No anomalies in the start/termination behaviour. |
| 20%-34% | The programme compiles, loads, and operates with significant errors. | Most classes and methods work as expected, but there are a few errors. Single class, with some additional methods. | A mostly accurate description of the overall algorithm with some test cases. | Minor anomalies in the start/termination behaviour. |
| 0%-19% | The programme has significant errors or does not compile/load at all. | Significant design and implementation errors. Single class with only the main method. | Incorrect description of the algorithm with few or no test cases. | Significant anomalies in the start/termination behaviour. |

Include either a marking grid based on the table below, OR objective assessment criteria for each of the degree classes, e.g. To qualify for a mark of 50% or greater, the program must have the ability to load and process text files in addition to the direct processing of text ……. Raheel used this approach quite effectively last year and I am sure would be happy to share with other members of the team.