|  |  |  |
| --- | --- | --- |
| **CMP5332 Object-Oriented Programming Checklist** | **Faculty of Computing, Engineering and the Built Environment** | New Logo Tiny |

|  |  |
| --- | --- |
| Please fill in your name and student ID in the table below. | |
| **Student Name** |  |
| **Student Number** |  |
| **Course and Year** | 2023 |
| **Module Code** | CMP5332 |
| **Module Title** | Object Oriented Programming |
| **Module Leader** | Dr. Abdel-Rahman Tawil |
| **Assessment item:** | Interactive Library System |

|  |  |  |
| --- | --- | --- |
| Students are required to complete this software implementation checklist for their Interactive Library System developed in java. You need to select only the features that were implemented in your code. You can select all/some features from any marking range as long as they have been implemented in the code submitted for this assessment. For example, you can select all/some features from the marking range 40%-49% and all/some features under the range 70%-79% etc. This implementation checklist should be submitted with the PowerPoint documentation for this coursework.  **Important notice**: This checklist will assist the tutors when marking your code, hence, you should only select the feature requirements that are implemented in your code. Even if some features are not working correctly, you can still select them as long as there is evidence in your code showing the implementation attempt. However, it is not acceptable for a student to claim the implementation of features that were not attempted/implemented in the game. False claims are a clear indication that the student does not understand the submitted code, hence, the submission will be investigated further for plagiarism, and the tutor marking the assessment may invite the student to explain all/parts of the submitted code. | |  |
| **Checklist Interactive Library System** | |  |
| **Achieving a mark of 40% to maximum of 49%** | |  |
| The game **must** implement **all** the following: | | Pending |
| * Add new Patrons (members) to the system. System should store at least the following information for each member: ID, Name, Phone Number and List of Books Borrowed. |  |  |
| * List all Books stored in the system. |  | Handling multiple copies of a book |
| * List all Patrons stored in the system. |  |  |
| * Issue Books to Patrons. When a book is issued to a Patron a Loan object must be created holding a reference to the Book being issued, to the Patron borrowing the Book and the Due Date of the Loan. This object should be added to the Book and the book should be added to the Patron’s list of borrowed Books. |  |  |
| * Return issued books. When a member returns a book the status of the returned book should be updated to reflect its availability. Also book must be removed from the Patron’s list of borrowed Books. |  |  |
| * Save the status of the system to the backend storage (i.e. text file storage) when the system is closed. The library data should be stored in three different files (books.txt, patrons.txt and loans.txt). A sample format to save the different properties for each object is given in the **Sample Prototype Application** section above. When the system starts it should load the status of the library from the text files to the memory. |  |  |
| **Achieving a mark of 50% to maximum of 59%** | |  |
| The game **must** implement **all** the above and the following: | |  |
| * Add a publisher property to the Book object and make the appropriate changes to the program to ensure that this information can be captured when a new Book is created. Also ensure that this information will be stored to and correctly loaded from the file storage. |  |  |
| * Add an email property to the Patron object and make the appropriate changes to the program to ensure that this information can be captured when a new Patron is created. Also ensure that this information will be stored to and correctly loaded from the file storage. |  |  |
| * Implement Unit Tests to validate and demonstrate that the above changes made to the Book and Patron classes work as expected. |  |  |
| **Achieving a mark of 60% to maximum of 69%** | |  |
| The game **must** implement **all** the above and the following: | |  |
| * Extend the prototype implementation for the GUI application provided to allow for the following basic functionalities:   + Display a popup window that will show the Patron details if a Book is on loan.   + List all Patrons and their details including the number of books they have on loan.   + Display a popup window that will show the Books details if a Patron has Books on loan.   + Display a popup window when the “Add” submenu of the “Patrons” menu item is selected. The popup should display a form that allows the addition of a new Patron to the system. |  |  |
| * Extend the functionality of the library system to allow for storing data to the file storage after the execution of commands that change the state of the system (e.g. “addbook”, “renewbook”). If the system fails to store the data on the file storage due to an error (e.g. file is already in used or corrupted), the program must inform the user and rollback any changes made to the system prior to the error. You can change the file permission to “read-only” in order to test this functionality. |  | **Now perisits for all commands** |
|  |  |  |
| **Achieving a mark of 70% to maximum of 79%** | |  |
| The game **must** implement **all** the above and the following: | |  |
| * Remove (hide) existing books from the system. When a book is removed, it should not appear in the books view. Instead of completely deleting a book, use a Boolean property in the Book class to indicate whether the book is deleted. Change the affected functions appropriately to return only the books that are not deleted. |  |  |
| * Remove (hide) existing Patrons from the system. When a patron is removed, it should not appear in the patrons view. Instead of completely deleting a patron, use a Boolean property in the Patron class to indicate that the patron is deleted. Change the affected functions appropriately to return only the books that are not deleted. |  |  |
| * Impose a limit on the maximum number of books that a patron can borrow and use that during issuing books for a patron. |  |  |
| * Extend the implementation for the GUI application to add the Delete functionality for both Books and Patrons using the GUI. |  |  |
| * Add Javadoc documentation for the newly created methods. |  |  |
| **Achieving a mark of 80% and over** | |  |
| The game **must** implement **all** the above and the following: |  |  |
| * Keep record of all loan history for Patrons. To implement this functionality you need to introduce a Boolean property in the Loan class to indicate whether the loan is terminated and a property that will hold the date that the book was returned. You have the flexibility to decide on how to complete the implementation of this functionality. |  |  |
| * Functionalities:   + Implement the Borrow functionality to borrow a book using the GUI.   + Implement the Renew functionality to allow for renewing a book loan using the GUI.   + Implement the Return functionality to return a book using the GUI. |  |  |