SUBDOMAIN 430.1 - INTRODUCTION TO PROGRAMMING

Competency 430.1.2: Algorithm Design and Development - *The graduate designs and develops algorithms for problem solving and implements those algorithms using appropriate program code.*
Competency 430.1.3: Use of Data Structures - *The graduate develops working programs that use appropriate data structures for problem solving.*
Competency 430.1.4: Modeling Systems Using Unified Modeling Language (UML) - *The graduate develops and interprets Unified Modeling Language (UML) diagrams which model object-oriented designs.*
Competency 430.1.5: Object-Oriented Concepts - *The graduate applies object-oriented concepts, develops object-oriented designs, and uses object-oriented programming techniques.*
Competency 430.1.6: Software Testing and Troubleshooting - *The graduate applies software testing and troubleshooting strategies to determine programming errors and recommend appropriate solutions.*

Introduction:

As a competent programmer, your ability to design and develop algorithms, your proficient use of data structures, and your ability to use the Unified Modeling Language (UML) to communicate and develop object-oriented designs will help you design and develop applications to meet customer requirements. A solid understanding of object-oriented concepts will help you develop applications that are maintainable and extensible. Strong competence in software testing and troubleshooting will allow you to validate and verify your applications to ensure you are delivering a quality product that meets all requirements.

You will need to develop a Java application to meet the requirements of this assignment. It is recommended that you use NetBeans as your integrated development environment. To download this program, follow the instructions found at the "Netbeans Download" web link below.

Use the following scenario to complete this task:

You are eager to demonstrate to management your ability to develop a client/server application with a MySQL backend database. Your prototype should interface with a database that holds student records. Each record will contain the student's first name, last name, student ID, GPA, mentor, and student status, along with other information specific to a student type. There are three types of students: full-time undergraduate, part-time undergraduate, and full-time graduate.

If management pursues this idea, the application must be easy to maintain and easy to extend. For these reasons, your application must be developed using an object-oriented design.

Your application should provide functionality to update, add, and delete students from the database. The application should also query the database to print out a selected student's information or all students' information. The functional and design requirements of the system are outlined in part A of the task.

Task:

Note: For this task, you will submit a working student record managing system application, UML Use Case diagrams with descriptions of the functional requirements, UML Class diagrams, UML Sequence diagrams, and a test plan with test cases.

Note: Before you begin, follow the steps in the "Installing MySQL and Using NetBeans" document. Use the attached "Registrar.sql" script to create your registrar database and create the student table schema.

A. Build a student record managing system application by doing the following:
   1. Create a Student inheritance hierarchy. This should include the following requirements:
      - Create the Student class. Student will be the super class and an abstract class. At least one method on Student will be abstract.
      - Student class holds the attributes first name, last name, student ID, GPA, status (resident or nonresident), and mentor. All attributes will be inherited by the subclasses.
      - Create appropriate overloaded constructors for the super class, Student.
      - Student has an abstract method, calculateTuition(), which will be implemented in the subclasses Undergraduate, Graduate, and Part-time, making use of the polymorphism principle.
      - Create the subclasses Undergraduate, Graduate, and Part-time, which inherit from the Student class. Each subclass will encapsulate a specialization of Student.
      - Create an abstract method, update(). When this method is called, the database entry for the student is updated.
      - Create an abstract method, add() method. When this method is called, the database entry for the student is created.
      - Create an abstract method, delete(). When this method is called, the database entry for the student is deleted.
      - Create an abstract method, query(). When this method is called, a query is made to the database to retrieve the information and is then printed to the screen.
   2. Create subclasses. This should include the following requirements:
      - Create the subclass Undergraduate, which inherits from Student. Undergraduate has the additional attribute, "level" (freshman, sophomore, junior, senior).
      - Create the subclass Graduate, which inherits from Student. Graduate has additional attributes, "thesisTitle", and "thesisAdvisor".
      - Create the subclass Part-time, which inherits from Student. Part-time students are working adults. Part-time students have an attribute, "company", which is the name of their sponsoring employer.
      - Incorporate the technique of information hiding by making appropriate attributes private and creating getters and setters to access and modify each private attribute.
      - Create overloaded constructors for each Student type. The super class constructor should be used by each subclass's constructors to set those attributes found in the super class. Create enough constructors for each class to initialize the instance variables (attributes) of an object, either by initial values passed into the constructor or default values used if none is passed in. All values passed in should be verified for validity.
   3. Implement the following methods using polymorphism. This should include the following requirements:
      - Implement calcuateTuition() method as follows:
        ~ Status of resident: undergraduate tuition = number of credit hours × 200
        ~ Status of nonresident: undergraduate tuition = number of credit hours × 400
        ~ Status of resident: undergraduate part time = number of credit hours × 250

- ~ Status of nonresident: undergraduate part time = number of credit hours × 450
- ~ Status of resident: graduate = number of credit hours × 300
- ~ Status of nonresident: graduate = number of credit hours × 350

- Student objects should know how to display the information. Override the toString() method on each subclass. Use inheritance to display attributes provided by super class by calling super.toString() method.

4. Override the following methods. This should include the following requirements:
   - For each subclass, implement method query(). When this method is called, a query is made to the database to retrieve the information and then the information is printed to the screen. If appropriate, use toString() within the print() method.
   - For each subclass, implement method update(). When this method is called, the database entry for the student is updated.
   - For each subclass, implement method add() method. When this method is called, the database entry for the student is created.
   - For each subclass, implement method delete(). When this method is called, the database entry for the student is deleted.
5. Create an application that tests the following requirements:
   - Application should allow the user to add, update, delete, and query any student type.

Note: The application can have a graphical user interface (GUI) or can be run from the command prompt. It should be clear how to use the application. Prompts should be clearly worded and mechanics of the prompt/receive input should work well. If necessary, provide a short user manual that explains how to use the application.

B. Create UML Use Case Diagrams that could be inserted into a Software Requirement Specification (SRS) document that captures all functional requirements for this application. Use the task description in part A to elicit the requirements. You will only be required to document the functional requirements and provide Use Case diagrams. You do not have to complete a complete SRS document.
   1. Document the functional requirements with UML use case diagrams for *each* requirement. For *each* functional requirement provide the following:
      - Requirement number and title
      - Description of the functionality
      - Input
      - Results of processing or output
      - Error handling or recovering requirements outlined
      - UML use case diagrams (It is very important that your UML use case diagrams are complete and proper UML 2.0 Modeling notation is used.)

C. Create UML Class and Sequence Diagrams that could be inserted into a Software Design Specification (SDS) document. (It is very important that your UML diagrams are complete and proper UML 2.0 Modeling notation is used. Your design must be an object-oriented design.) You will only be required to submit the class and sequence diagrams and not have to complete a SDS document.
   1. Your UML Diagrams should include the following:
      a. Comprehensive Class Diagrams: Show all classes and all relationships among all classes. These should include the following:
         - Relationships: Your object oriented design will require you to diagram all class relationships using correct UML 2.0 Modeling notation which minimally includes:

- Inheritance
- Association Relationships

- Multiplicity: Note any multiplicity in the relationships (e.g., 1-1, 1-many, etc.) using correct UML 2.0 Modeling notation.
- Attributes: For each class, all attributes with types (e.g., int, double, etc.) and access control (e.g., private, public, protected) need to be noted using proper UML 2.0 Modeling notation.
- Methods: Methods with signatures and access control should be provided. All methods have to be indicated, including all constructors, setters, and getters.
- Abstract Classes and Interfaces: Any abstract classes and interfaces used in your design must be properly indicated using UML 2.0 Modeling notation.

b. Sequence Diagrams: Model the object interactions required for *each* functional scenario. For *each* Use Case provided in the functional requirements section, at least **one** sequence diagram should be provided to model the interaction between the objects inside the application.

D. Develop a test plan that includes test cases that test all functional requirements of the system.

Note: The test plan is the overall approach to how the application will be tested. The test cases are the steps a tester will take to test a single behavior. Pass/Fail criteria need to be provided for each test case; all test cases should have been run, and the Pass/Fail section should be completed. For each functional requirement, one or more test cases should be provided.

Note: All compiler and runtime errors must be corrected before you submit. Your application should correctly satisfy all requirements. Your application should have passed all test cases before being submitted for evaluation.

1. The test plan should include at least the following sections:
   - Title Page
   - Introduction: Brief description (*suggested length of 1–2 paragraphs*) of the application under test
   - Overview of Testing Strategy: See part D1a.
   - Resource Requirements: Discuss hardware and software required to complete the test plan.
   - List of Features to Test: A simple table is sufficient. If features are numbered, the feature number can be used in your test cases.
   - Acceptance Criteria: Identify the standards that must be met when considering if the application is ready for release. In your case, this means that the application should be ready to be submitted for grading.
   - Test Cases: Should be submitted as an appendix to the test plan

   a. In the "Overview of Testing Strategy" section, compare white box testing, black box testing, unit testing, and integration testing and how they can be used to verify your application.
      - Identify which testing strategy you will use.
2. Execute the test plan, including running all test cases.
   a. Discuss the results of *each* test case in the "Test Cases" section. (A Pass/Fail value should be indicated as a result for each test case.)

   Note: For an example, see the attached "Sample Test Case."

Note: If you choose to use a test plan and/or test case template found on the Internet, please ensure you identify the source using APA citation.

E. Create a working Java application that meets all requirements.

Note: Include all .java and .class files along with any test data used. Your application will be verified to ensure all requirements were met. If you created this application using the NetBeans integrated development environment, please zip your entire NetBeans project folder. This will include all .java and .class files.

F. If you use sources, include all in-text citations and references in APA format.

*Note: Submit all .java and .class files in one zipped folder. If you developed this application using NetBeans, it is recommended that you zip and submit the entire NetBeans project folder. Submit all word-processing documents that contain your use case diagrams, class diagrams, sequence diagram diagrams and Test Plan as *.rtf (Rich Text Format) files.*

Note: For definitions of terms commonly used in the rubric, see the Rubric Terms web link included in the Evaluation Procedures section.

Note: When using sources to support ideas and elements in a paper or project, the submission MUST include APA formatted in-text citations with a corresponding reference list for any direct quotes or paraphrasing. It is not necessary to list sources that were consulted if they have not been quoted or paraphrased in the text of the paper or project

Note: No more than a combined total of 30% of a submission can be directly quoted or closely paraphrased from sources, even if cited correctly. For tips on using APA style, please refer to the APA Handout web link included in the General Instructions section.