PROGRAMMING FUNDAMENTALS—SOFTWARE I

Competencies:
430.2.5: Classes and Interfaces *- The graduate designs software solutions with appropriate classes, objects, methods, and interfaces to achieve specific goals.*
430.2.6: Object-Oriented Principles *- The graduate implements object-oriented design principles (e.g., inheritance, encapsulation, and abstraction) in developing applications for ensuring the application's scalability.*
430.2.7: Application Development *- The graduate produces applications using Java programming language constructs to meet business requirements.*
430.2.8: Exception Handling *- The graduate incorporates simple exception handling in application development for improving user experience and application stability.*
430.2.9: User Interface Development *- The graduate develops user interfaces to meet project requirements.*

Task 1: Object-Oriented Application Development

Introduction:

Throughout your career in software design and development, you will be asked to create applications with various features and functionality based on business requirements. When a new system is developed, typically the process begins with a business analyst gathering and writing these business requirements, with the assistance of subject matter experts from the business. Then a system analyst works with several application team members and others to formulate a solution based on the requirements. As a developer, you would then create a design document from the solution and finally develop the system based on your design document.

For this assessment, you will create a Java application using the solution statements provided in the requirements section.

The skills you showcase in your completed application will be useful in responding to technical interview questions for future employment. This application may also be added to your portfolio to show to future employers.

Your submission should include a zip file with all the necessary code files to compile, support, and run your application.

Scenario:

You are working for a small manufacturing organization that has outgrown its current inventory system. They have been using a spreadsheet program to manually enter inventory additions, deletions, and other data from a paper-based system but would now like you to develop a more sophisticated inventory program.

They have provided you with a mock-up of the user interface to use in the design and development of the system (see the attached "GUI Mock-Up") and a class diagram to assist you in your work (see the attached "UML Class Diagram"). The organization also has specific business requirements that must be included as part of the application. A system analyst from your company created the solution statements outlined in the requirements section based on the manufacturing organization's business requirements. You will use these solution statements to develop your application.

Requirements:

Your submission must be your original work. No more than a combined total of 30% of the submission and no more than a 10% match to any one individual source can be directly quoted or closely paraphrased from sources, even if cited correctly. Use the Turnitin Originality Report available in Taskstream as a guide for this measure of originality.

You must use the rubric to direct the creation of your submission because it provides detailed criteria that will be used to evaluate your work. Each requirement below may be evaluated by more than one rubric aspect.

The rubric aspect titles may contain hyperlinks to relevant portions of the course.

I. User Interface

Create a JavaFX application with a graphical user interface (GUI) based on the attached "GUI Mock-Up". Write code to display *each* of the following screens in the GUI:

A. A main screen, showing the following controls:
- buttons for "Add", "Modify", "Delete", "Search" for parts and products, and "Exit"
- lists for parts and products
- text boxes for searching for parts and products
- title labels for parts, products, and the application title

B. An add part screen, showing the following controls:
- radio buttons for "In-House" and "Outsourced" parts
- buttons for "Save" and "Cancel"
- text fields for ID, name, inventory level, price, max and min values, and company name or machine ID
- labels for ID, name, inventory level, price/cost, max and min values, the application title, and company name or machine ID

C. A modify part screen, with fields that populate with presaved data, showing the following controls:
- radio buttons for "In-House" and "Outsourced" parts
- buttons for "Save" and "Cancel"
- text fields for ID, name, inventory level, price, max and min values, and company name or machine ID
- labels for ID, name, inventory level, price, max and min values, the application title, and company name or machine ID

D. An add product screen, showing the following controls:
- buttons for "Save", "Cancel", "Add" part, and "Delete" part
- text fields for ID, name, inventory level, price, and max and min values
- labels for ID, name, inventory level, price, max and min values, and the application
- a list for associated parts and their products
- a "Search" button and a text field with an associated list for displaying the results of the search

E. A modify product screen, with fields that populate with presaved data, showing the following controls:
- buttons for "Save", "Cancel", "Add" part, and "Delete" part
- text fields for ID, name, inventory level, price, and max and min values
- labels for ID, name, inventory level, price, max and min values, and the application
- a list for associated parts and their products
- a "Search" button and a text field with associated list for displaying the results of the search

II. Application

Now that you've created the GUI, write code to create the class structure provided in the attached "UML (unified modeling language) Class Diagram". Enable *each* of the following capabilities in the application:

F. Using the attached "UML Class Diagram", create appropriate classes and instance variables with the following criteria:
- five classes with the 16 associated instance variables
- variables are only accessible through getter methods
- variables are only modifiable through setter methods

G. Add the following functionalities to the main screen, using the methods provided in the attached "UML Class Diagram":
- redirect the user to the "Add Part", "Modify Part", "Add Product", or "Modify Product" screens
- delete a selected part or product from the list
- search for a part or product and display matching results
- exit the main screen

H. Add the following functionalities to the part screens, using the methods provided in the attached "UML Class Diagram":
   1. "Add Part" screen
      - select "In-House" or "Outsourced"
      - enter name, inventory level, price, max and min values, and company name or machine ID
      - save the data and then redirect to the main screen
      - cancel or exit out of this screen and go back to the main screen
   2. "Modify Part" screen
      - select "In-House" or "Outsourced"
      - modify or change data values
      - save modifications to the data and then redirect to the main screen
      - cancel or exit out of this screen and go back to the main screen

I. Add the following functionalities to the product screens, using the methods provided in the attached "UML Class Diagram":
   1. "Add Product" screen
      - enter name, inventory level, price, max and min values, and company name or machine ID
      - save the data and then redirect to the main screen
      - associate one or more parts with a product
      - remove or disassociate a part from a product
      - cancel or exit out of this screen and go back to the main screen
   2. "Modify Product" screen
      - modify or change data values
      - save modifications to the data and then redirect to the main screen
      - associate one or more parts with a product
      - remove or disassociate a part from a product
      - cancel or exit out of this screen and go back to the main screen

J. Write code to implement exception controls with custom error messages for *each* of the following sets:
   1. Set 1
      - entering an inventory value that exceeds the minimum or maximum value for that part or product
      - preventing the minimum field from having a value above the maximum field
      - preventing the maximum field from having a value below the minimum field
      - ensuring that a product must always have at least one part
   2. Set 2
      - preventing the user from deleting a product that has a part assigned to it
      - including a confirm dialogue for all "Delete" and "Cancel" buttons
      - ensuring that the price of a product cannot be less than the cost of the parts
      - ensuring that a product must have a name, price, category, and inventory level (default 0)

File Attachments:
GUI Mock-Up link opens in new window
UML Class Diagram link opens in new window

Web Links:
1. **GYP Task 1 Rubric link opens in new window**