Faculty of Engineering, Environment and Computing

308SE – Object Oriented Software

Coursework Task Sheet 2019

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Module Title: Object Oriented Software (Semester 2) | | Module Code: 308SE \_1819JANMAY | | |
| Assignment Title: Cw: Client-Server Stock Keeping | | Hand out: Monday, 21 January 2019 (T1) | | |
| Lecturer: Mr. Christopher J. Bass | | Due date: Friday, 29 March 2019 (T10)  (18:00 online Moodle deadline) | | |
| Estimated: 50 hours | % of Module Mark: 50% | Assignment type: Summative – Individual | | |
| Submission arrangements: Submission via Coventry University GitHub and Moodle. | | | | |
| Assessment Regulations:  Coventry University regulations on assessed coursework apply. You are advised to read the guidelines.  Any student requiring an extension or deferral should follow the university process as outlined here:  <https://share.coventry.ac.uk/students/Registry/Pages/Deferrals-and-Extension.aspx>  Keep a safe copy (backed up) of all coursework submitted for reference.  Coversheets are **not** required. | | | | |
| Marks breakdown | | | Max | Awarded |
| Assignment submission | | | 50% of Mm |  |

# Relationship to Module Learning Outcomes

1. Apply UML use-case, activity, sequence and class diagrams to the analysis of programming problems and the design OO solutions.

*Exercises methodical analysis and design through the use of the specified UML diagrams applied to an object-oriented programming problem.*

1. Implement and execute small OO programs (up to 5 classes)

*Requires the implementation and execution of a program using GUIs, constructed from student defined and library classes in Java.*

1. Demonstrate the application of common data structures.

*Requires the use of data structures using the Java Collections library, typical examples are ArrayLists and HashMaps.*

1. Determine the requirements and structure of programs running and communicating across a network.

*This assignment requires a client-server application to be analysed.*

1. Design and develop and test OO programs that can communicate with each other across a network

*This assignment requires a client-server application to be designed and developed.*

# Feedback

Feedback will be provided on the coursework via the feedback section in the Moodle 2 gradebook. Individuals can discuss their feedback in the weekly tutorial slots as required.

# Feedforward

This coursework will develop problem solving skills and programming capabilities, which will be of great benefit for employability and parts of other modules (e.g. 310SE, 330CT, 306SE, 306AAE).

# Individual Assignment Task

## Java Purchase Orders

An engineering company has a purchasing department with the time-critical task of keeping stock room levels of parts available for use on the shop floor. The team spends most of their time managing a large quantity of parts and creating purchase orders when stock levels reach a critical level. In the past, the purchasing manager has used large spreadsheets to manage these stock levels. Now as the department is growing in size, it is clear that a software solution is needed for managing purchase orders.

Some form of database will be required to store the purchase orders and store room parts. You can use text-files, XML-flies, or a relational database management system with SQL for your database solution. The marking for this assignment not specifically concerned with how you implement the database so it is suggested that you choose whichever solution you prefer to implement. The database will require 3 data tables:

Each student should work individually on this coursework. In support of this, each student should solve the problem for a different set of data, based on their individual SID number.

For example, if the last 6 digits of your SID end in 466395 then you should leave out the data rows below with a corresponding 4, 6, 3, 9 and 5 in front of them in your database and program implementation.

A ‘purchase order’ is characterised by:

* Purchase order id. – primary key

1. Date created – date and time
2. Requisitioner – i.e. the name of who made the purchase order
3. Sub-project/department code – i.e. the department who made the purchase order
4. Status – text (e.g. ordered, on-hold, cancelled, dispatched, completed)
5. Expected delivery – date and time
6. Delivery contact – department of who to deliver to
7. Delivery address
8. Delivery description – special instructions for delivery
9. Delivery attention – FAO. person who to deliver to
10. Completed status – yes/no (open/closed)

A ‘store room part’ is characterised by:

* Part id. – primary key
* Manufacturer – text
* Manufacturers part number – text

1. Component description – text
2. Current stock level – whole number
3. Kanban size / low stock level – whole number
4. Safety stock level – whole number
5. Shelf location – text
6. Last order date – date and time
7. Usage rate – how many parts were used last month

A ‘purchase order line’ (link table) is characterised by:

* Line id. – primary key
* Purchase order id. – foreign key (links to ‘purchase order’ primary key)
* Part id. – foreign key (links to ‘store room part’ primary key)
* Quantity – number to order
* Price per unit – number (pounds per unit)

1. Currency – GBP, EURO, USD, etc.
2. Supplier – text
3. Supplier part order code – text

(Note that databases and data-structures are two different concepts. You will have the opportunity to demonstrate the use of both in this coursework).

Your main task is to develop a Java client-server application for the purchasing team to use on a daily basis to help them manage the stock levels in the stock rooms.

By ‘client-server’, it is meant that you should develop both the client application and server program and that they should communicate with each other over a TCP network connection. Java provides the ‘Socket’ and the ‘ServerSocket’ library classes as part of the ‘java.net’ package for this.

The database must sit on the server-side so that multiple clients can connect and access the data synchronously. A simple GUI should be developed for the client-side application so that multiple purchasers can operate the system with ease. You are expected to use one of the Java GUI frameworks for building your GUI; choose between AWT or Swing.

A stock control administrator using the system should be able to:

* Open the client GUI and connect to the central server and database. (do **not** need to implement a login system, assume it is handled by another part of the system).
* View a list of all purchase orders.
* View a list of all store room parts.
* Open an existing purchase order.
* Create, view, modify, or delete a purchase order and purchase order details.
* Create, view, modify, or delete store room parts.
* Send a purchase order to the printer.
* Extra features could include (but not limited to):
  + Optionally view all purchase orders from a particular supplier.
  + Optionally view all purchase orders for a particular part.
  + Optionally view all purchase orders for an expected delivery date.
  + Optionally view all purchase orders which are open and past the expected date.
  + Optionally view all purchase orders sorted by delivery date.
  + Optionally view parts which are low stock or below the safety stock level.
  + Optionally view a list of parts which are in a specific shelf location.
  + Or similar sorting and filtering operations.

**Before** writing the program a methodical analysis and design must be completed including UML:

Analysis:

1. *Console diagram* for the GUI. Show how a user interacts through the GUI. Keep the GUI as simple as possible (all in the same frame – i.e. no menu system). Label components where necessary. (can produce this in MS Word).
2. UML *use case diagram* at high-level to show the actors and their main goals for the system.
3. Detailed *use case description* for one major use case (can produce this in MS Word).
4. High-level UML *activity diagram* to show interaction for one major use case, use swimlanes.
5. UML *activity diagram* at mid-level showing the event loop and possible events occurring in your client application after the program is run and presented with your main GUI screen.
6. Perform *textual analysis* on your use case description and identify a list of potential classes. (UML case tool or MS Word).

Design:

1. UML *class diagram* consistent with the application code (show the class design, attributes, methods, and any class relationships).
2. UML *sequence diagram* for one major use case to show object-interaction. The diagram must include a pseudocode narrative on the left-hand side.

All diagrams must be produced in their appropriate UML models as required. Include your name and SID in a note in the top left of each diagram. Store all UML models together in a single UMLStudio or Visual Paradigm file similar to as shown in figure 1.

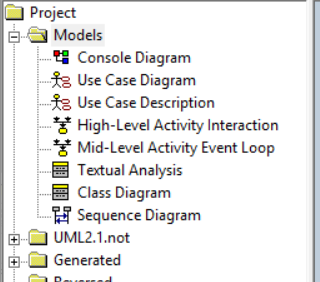


Figure 1 – UMLStudio Models Folder

**After** writing your program, you must create a screencast video clip (approx. 3 minutes in length).

In the clip, you must demonstrate your program running with test data and describe the stability, features and functionality of your client-server application. Point out any shortcomings, bugs, logic errors or limitations in your solution.

For a pass your program and video should meet and demonstrate the following requirements to a suitable standard:

1. The client program builds and runs without errors.
2. The server program builds and runs without errors.
3. An operational client-side GUI has been implemented with suitable controls allowing for transfer of data between user and client (use of Event Handling).
4. The program is in close correspondence to the UML diagrams.
5. The client program can successfully connect to and exchange data with the server.
6. The server program can successfully store and retrieve data from the database.
7. The basic program functionality is achieved.
8. Use of data structures (such as ArrayLists) for temporary storage and manipulation of data.
9. Neatly written program code with consideration to the use of tabbing, suitable naming conventions, structured appropriately.
10. Access to NetBeans project and source code is provided through CU GitHub repository.

For a first your program and video demonstration should additionally meet the following requirements to a high standard:

1. The full program functionality is achieved. Excellent functionality. Covers all requirements. No bugs or logic errors.
2. The GUI is refined in its layout and design (use of multiple components from the GUI framework and layouts).
3. The client program is protected from entering invalid data it will not crash the program.
4. The server is protected from clients sending bad data it will not crash the server or corrupt the database.
5. Exception handling is implemented.
6. Use of custom defined classes to reduce the system complexity and promote an object-oriented design (demonstrates Abstraction, Encapsulation, Information Hiding, etc.).
7. The server program is multi-threaded.
8. The server can accept connections and exchange data with multiple clients at the same time.
9. Excellent use of classes and class design. Makes use of standard library features.
10. Excellent use of data structures. Use of Java collections e.g. ArrayList<> or HashMap<>, use of generics, iterators etc.
11. Evidence of regular commits and consistent, detailed and meaningful commit messages
12. No linting errors.
13. Full Javadoc documentation for your code.

All Java code files must contain your name and SID as part of the Javadoc comments at the top.

# Assessment

The individual assignment is a summative assessment worth 50% of the module mark.

# Submission

1. Submission via Coventry Uni. GitHub: <https://github.coventry.ac.uk/308SE-1718JANMAY>
2. Ensure your repository is named as such: <your username>-<your project name> e.g.: aa6164-CwCheeseClub.
3. Ensure your repository is private.
4. Make regular commits before the deadline due date. Commits after the due date will not be marked.
5. Include your full NetBeans project, your Java source code files (.java), and other material (e.g. test database, sample data, password, and instructions to setup if required).
6. Upload your UML diagrams to Moodle.
7. Upload a link to your GitHub source code repository on Moodle.
8. Upload a link to your video evidence on Moodle.

You can submit anytime before the deadline.

Please note that any source code file or built file may be scanned by plagiarism detecting software.

Late submission will incur a mark of zero.

# Assessment Marking Grid

Marks for the assignment will be awarded out of 10 for each of the following criterion, and then averaged and weighted accordingly.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criteria**  **Mark** | **UML Diagrams** | **Stability, Features and Functionality** | **Connection client-server and database access on the server-side** | **Use of Classes, Standard Libraries and Data Structures** | **Code Standards, Neatness and Documentation** |
| **> 8. Outstanding** | Outstanding UML. Well presented.  Covers high and low level details. | Outstanding functionality.  Covers all requirements with possible extra features added. | Outstanding connection code. Multi-threaded. Synchronized methods. No logic errors with testing. Possible advanced thread pooling or broadcasting. Outstanding database code with full CRUD. | Outstanding use of classes.  Elegant class design with single responsibility principle (SRP) ~10 classes.  Makes extensive use of standard library features.  Data structures with outstanding code e.g. Java collection of custom objects. | Outstanding tidy code.  Good practices being used and Javadoc documentation.  Code contains no linting errors or warnings. |
| **> 7. Excellent (1st).** | Excellent UML diagrams.  Neat and fully correct in all respects. | Excellent functionality.  Covers all requirements. No bugs or logic errors. | Excellent connection code. Multi-threaded server allows connections at the same time.  Excellent database code with full create, read, update, delete (CRUD) functionality. | Excellent use of classes and class design. Identified additional classes from the problem description ~7-8 classes. Makes use of standard library features.  Data structures with excellent code. Use of Java collections e.g. ArrayList<> or HashMap<>, use of generics, iterators. | Excellent tidy code.  Good practices being used e.g. private attribute, meaningful identifiers and Javadoc documentation.  Code contains no linting errors. |
| **> 6. Strong (2:1).** | Strong UML diagrams.  No errors. | Strong functionality.  No bugs under normal use. | Strong connection code. Exchange of data as objects.  Database with strong read and write code. | Strong use of classes ~4-5 classes with clear responsibilities.  Data structures with strong code. Use of Java collections e.g. ArrayList<>. | Neat code. Lower camel case used on variable identifiers. Code may contain linting errors. |
| **> 5. Good (2:2).** | All UML diagrams completed.  Possible errors. | Program runs.  Covers some requirements.  Few bugs under normal use. | Good connection and successful exchange of data as strings. Database with server-side access. | Good use of classes.  ~4 classes are used.  Data structures with good code. | Mostly tidy code.  Naming okay e.g. upper camel case used on class names and correct tabbing. |
| **> 4. Okay (3rd).** | Most UML diagrams completed, some errors. | Program runs.  Could have bugs or crashes under normal use. | Connection established between client and server.  Basic data storage but maybe on client-side. | Some class design ~3 classes.  Basic use of data structures e.g. arrays with basic code. | Passable tidy code.  Possibly poor naming and tabbing. |
| **< 4. Poor (Fail).** | UML poor or missing. | Program does not run or there are compiler errors. | Connection code poor or missing. No client-server. No database code. | No class design ~2 classes.  No data structures used. | Untidy code. Poor structure and difficult to read. Poor naming, tabbing, and bad practices. |