Assignment

General Instructions:

- This assignment consists of two questions; both are mandatory.
- Question 2 is mini-project based and students can attempt anyone of the provided project options.
- Students are required to submit a report (hardcopy) by or before the submission deadline.

Assessment: Assessment will be based on two components:

1. Viva based on Mini-project

2. Demo and Assignment report.

Note: The marking will be relative where students will be ranked based on the quality of the mini-project. The specific grade/marks assigned to any student will depend on their rank.

Q1. File Handling and Stream Handling

Create a Java program that performs the following tasks:

Create a directory named "Demo" in the current working directory.

Create a text file named "example.txt" inside the "Demo" directory.

Write some sample text content to the file.

Read the contents of the file and display them on the console.

Update the content of the file by appending new text.

Read the updated contents of the file and display them on the console.

Delete the file "example.txt".

Delete the directory "Demo".

Extend the program to perform the following additional tasks:

Create a new text file named "byteExample.txt" and write some text using byte streams.

Read the contents of "byteExample.txt" using byte streams and display them on the console.

Create a new text file named "characterExample.txt" and write some text using character streams.

Read the contents of "characterExample.txt" using character streams and display them on the console.

Ensure error handling and exception handling mechanisms are implemented throughout the program. Handle situations such as file or directory not found, I/O exceptions, etc.

Document the code with appropriate comments and provide clear instructions on how to run the program.

Prepare a brief report explaining the file handling concepts used in the program, along with screenshots of the program's execution.

Note: Make sure to research and utilize the appropriate classes and methods from the **java.io** package for file handling and stream handling operations.

Q2. Mini Project based on JAVA AWT and SWING

Mini-Project: Student Management System

Design and implement a Student Management System using Java AWT and Swing. The system should allow users to perform the following operations:

- Add a new student: Allow the user to enter the student's details such as name, roll number, course, and marks in various subjects. Validate the input and store the student's information in a data structure.
- View student details: Display the details of all the students currently stored in the data structure. Show the student's name, roll number, course, and total marks.
- Search for a student: Provide a search functionality to find a student by their roll number or name. Display the student's details if found, or show a message if the student is not found.
- Update student details: Allow the user to update the details of a specific student. Provide options to modify the student's name, course, or marks in various subjects.
- Delete a student: Allow the user to delete a student from the data structure based on their roll number or name.
- Calculate statistics: Calculate and display statistics such as the average marks of all students, the highest and lowest marks, and the overall pass percentage.
- GUI Design: Design a user-friendly graphical user interface (GUI) using Java AWT and Swing components such as buttons, labels, text fields, tables, etc., to facilitate user interaction.
- Error Handling: Implement appropriate error handling and validation mechanisms to handle invalid inputs, such as alphabets in numeric fields or empty fields.
- Data Persistence: Implement file handling to store and retrieve student data from a file. Use file operations to load existing data when the program starts and save the data when changes are made.
- Documentation: Provide clear documentation with comments in the code and create a user manual explaining how to use the system.

Feel free to enhance this mini-project by adding additional features, such as sorting students based on marks, generating reports, or implementing authentication for secure access. You can also focus on improving the GUI design and adding validations to ensure data integrity.

Mini-Project: Student Mess Management System

Design and implement a Student Mess Management System using Java AWT and Swing. The system should allow users to perform the following operations:

- Add a new student: Allow the user to enter the student's details such as name, roll number, Branch and Phone number, Fee payment, validity. Validate the input and store the student's information in a data structure.
- View student details: Display the details of all the students currently stored in the data structure. Show the student's name, roll number, Fee payment status
- Search for a student: Provide a search functionality to find a student by their roll number or name. Display the student's details if found, or show a message if the student is not found.
- Update student details: Allow the user to update the details of a specific student. Provide options to modify the student's name, payment status.
- Delete a student: Allow the user to delete a student from the data structure based on their roll number or name.

- GUI Design: Design a user-friendly graphical user interface (GUI) using Java AWT and Swing components such as buttons, labels, text fields, tables, etc., to facilitate user interaction.
- Error Handling: Implement appropriate error handling and validation mechanisms to handle invalid inputs, such as alphabets in numeric fields or empty fields.
- Data Persistence: Implement file handling to store and retrieve student data from a file. Use file operations to load existing data when the program starts and save the data when changes are made.
- Documentation: Provide clear documentation with comments in the code and create a user manual explaining how to use the system.

Mini-Project: Sudoku Solver

Design and implement a Sudoku Solver using Java AWT and Swing. The program should allow users to enter a partially filled Sudoku grid and solve it using an algorithm. Here's an outline of the project:

- GUI Design: Design a graphical user interface (GUI) using Java AWT and Swing components to represent the Sudoku grid. Create a grid layout consisting of 9x9 text fields where users can enter the initial values of the Sudoku puzzle.
- Input Validation: Implement validation mechanisms to ensure that users can only enter numbers from 1 to 9 in the text fields. Validate the input to ensure that it represents a solvable Sudoku puzzle.
- Solving Algorithm: Implement a backtracking algorithm to solve the Sudoku puzzle. Use a recursive approach to fill in the empty cells of the grid one by one, making sure that the numbers entered follow the Sudoku rules (no repetition in the same row, column, or 3x3 sub-grid).
- Solve Button: Add a "Solve" button to the GUI that triggers the solving algorithm when clicked. Display the solved Sudoku grid in the text fields of the GUI.
- Clear Button: Add a "Clear" button to the GUI that resets the Sudoku grid to its initial state, clearing all the text fields.
- Error Handling: Implement error handling to handle cases where an invalid or unsolvable Sudoku puzzle is entered. Provide appropriate error messages or feedback to the user.
- Documentation: Document the code with comments, explaining the implementation details and the solving algorithm. Create a user manual or README file explaining how to use the Sudoku Solver.
- Extra Features (Optional): Add a "Generate" button to the GUI that generates a new Sudoku puzzle for the user to solve. Implement difficulty levels (easy, medium, hard) to generate Sudoku puzzles of varying complexity.

Mini-Project: Music Player

Design and implement a music player application using Java AWT and Swing. The application should allow users to play, pause, stop, and navigate through music tracks. Here's an outline of the project:

- GUI Design: Design a graphical user interface (GUI) using Java AWT and Swing components to represent the music player. Include buttons for play, pause, stop, next, previous, and a slider to display the playback progress.

- Music File Management: Implement functionality to load music files from a specified directory. Store the information about the loaded music files, such as the file path, title, artist, and duration.
- Music Playback: Utilize Java's audio capabilities (e.g., **javax.sound.sampled**) to play music files. Implement controls to play, pause, stop, and adjust the volume of the currently playing track. Display the current track's title, artist, and duration in the GUI.
- Playlist Management: Allow users to create playlists and add music files to them. Implement functionality to save and load playlists from files.
- Track Navigation: Enable users to navigate through tracks in a playlist using next and previous buttons. Implement a progress slider that shows the playback progress and allows users to jump to a specific position in the track.
- Error Handling: Implement error handling mechanisms for cases such as unsupported audio formats or missing audio files. Provide appropriate error messages or feedback to the user.
- Documentation: Document the code with comments, explaining the implementation details and the music player logic. Create a user manual or README file explaining how to use the music player application.
- Extra Features (Optional): Implement a shuffle mode that plays tracks randomly from the playlist. Add a repeat mode to repeat the current track or the entire playlist. Implement a search functionality to search for tracks by title or artist.

Mini-Project: Chat Application

Design and implement a chat application using Java AWT and Swing. The application should allow users to communicate with each other in real-time. Here's an outline of the project:

- GUI Design: Design a graphical user interface (GUI) using Java AWT and Swing components to represent the chat application. Include input fields for users to enter their messages and a chat log area to display the conversation.
- Client-Server Architecture: Implement a client-server architecture where multiple clients can connect to a server and communicate with each other. Establish socket connections between the clients and the server using Java's **Socket** and **ServerSocket** classes.
- Connection Establishment: Allow users to enter a username or nickname when connecting to the server. Implement functionality to connect to the server and join the chat room.
- Chat Communication: Enable users to send messages to the server, which will be relayed to all connected clients. Display the messages in the chat log area of each client's GUI in real-time.
- User Interface: Implement features such as scrolling to view previous messages, displaying timestamps, and distinguishing between users' messages.
- Private Messaging: Allow users to send private messages to a specific user by selecting their name or using a command. Ensure that private messages are only visible to the sender and the recipient.
- Error Handling: Implement error handling mechanisms for cases such as connection failures or invalid inputs. Provide appropriate error messages or feedback to the user.

- Documentation: Document the code with comments, explaining the implementation details and the chat application logic. Create a user manual or README file explaining how to use the chat application.
- Extra Features (Optional): Implement additional features such as chat room moderation, user authentication, or emoticon support. Add file sharing functionality, allowing users to send and receive files through the chat application.

Mini-Project: Image Editing Tool

Design and implement an image editing tool using Java AWT and Swing. The tool should allow users to perform various image manipulation operations. Here's an outline of the project:

- GUI Design: Design a graphical user interface (GUI) using Java AWT and Swing components to display and manipulate images. Include options to open, save, and display images in the GUI.
- Image Operations: Implement various image editing operations such as rotation, scaling, cropping, and flipping. Allow users to adjust image properties like brightness, contrast, saturation, and hue. Provide image filters and effects such as grayscale, sepia, blur, sharpen, and more. Implement drawing tools such as brushes, shapes, and text overlay.
- Undo and Redo: Implement an undo and redo functionality to allow users to revert or reapply image edits.
- Image Format Support: Support multiple image formats (e.g., JPEG, PNG, GIF) for opening and saving images. Utilize the appropriate libraries or classes in Java for image handling and manipulation.
- Keyboard and Mouse Interactions: Handle keyboard and mouse interactions for drawing, selecting, and manipulating images.
- Error Handling: Implement error handling mechanisms for cases such as invalid image formats or unsupported operations. Provide appropriate error messages or feedback to the user.
- Documentation: Document the code with comments, explaining the implementation details and the various image editing operations. Create a user manual or README file explaining how to use the image editing tool.
- Extra Features (Optional): Implement image layering and blending modes to create composite images. Add image transformation operations like perspective distortion or morphing. Provide image enhancement features such as auto-adjustments or histogram equalization. Add image annotation features like adding text, arrows, or shapes to images.