Chapter by chapter Project requirements: Indicate whether or not you accomplished these requirements for every chapter. For Chapters 3, 4, 7, 8, 9, 11, 13 and 14 you must state which programs where these requirements were implemented.

Chapter 1: Basic procedural programming with no syntax errors

* **Accomplishment**
  + Established a foundational structure for the project using procedural programming principles.
* **Implementation**
  + The main class, OrderEaseApp, follows a logical structure that sets up the GUI and handles user interactions sequentially.
  + Methods such as displayMenu() and placeOrder() perform specific tasks in a clear, procedural manner, ensuring each action is completed before moving to the next.

Chapter 2: Proper use of variables and major types such as integer, double, Boolean, Final, use of print and println including concatenation. Basic use of GUI dialog boxes. Arithmetic statements and assignment statements. Keyboard input and GUI input would be best practice.

* **Accomplishment**
  + Correct use of various data types, variables, and GUI input techniques to manage the application's data and interactions.
* **Implementation**
  + **Variables and Data Types**
    - Different data types (e.g., String, int, double, boolean) are used to represent menu item details, table numbers, and order totals.
  + **GUI Input**
    - User input is collected through JOptionPane dialogs (e.g., when guests place orders or log in as servers/admins). This approach ensures that all input is processed correctly and matches the expected data types.

Chapter 3: Use of getters (accessors) and setters (mutators). Use of methods, parameters, arguments, return statements. Use of classes, static and nonstatic methods. Appropriate use of instance methods. Use of instance fields. Use of constructors.

* **Accomplishment**
  + Leveraged object-oriented principles to create a modular and extensible system.
* **Implementation**
  + **Classes and Objects**
    - The system is divided into distinct classes such as MenuItem, Order, Table, Server, and Restaurant, each with its own attributes and methods.
  + **Encapsulation**
    - Attributes in each class are private, with public getters and setters provided to access and modify the data. This design maintains data integrity and prevents unintended changes from outside the class.
  + **Inheritance and Polymorphism**
    - While the project does not directly use inheritance for subclasses, the design allows for future extension, such as creating specialized menu item subclasses.

Chapter 4: Appropriate use of scope. Use of constructors with parameters. Use overloading constructors. Use of the this reference. Use of static fields. Packages (math or similar: see this chapter for this information), constants. Use of classes, such as nested or inner classes.

* **Accomplishment**
  + Correctly defined scopes and constructors, with appropriate use of static fields for class-level data.
* **Implementation**
  + **Constructors**
    - Each class has constructors that initialize its fields. For example, the MenuItem constructor initializes the item name, description, price, and availability.
  + **Scope**
    - Variables are declared with appropriate access modifiers (private, public) to control visibility and access. Local variables within methods are used for temporary data storage.
  + **Static Fields**
    - Static fields, such as serialVersionUID in serializable classes, ensure proper deserialization and data integrity.

Chapter 5: Appropriate use of nested if statements.

* **Accomplishment**
  + Used selection statements for decision-making processes based on user inputs and system states.
* **Implementation**
  + **if-else Statements**
    - Used throughout the project to handle different conditions. For example, checking if a menu item is available before adding it to an order.
  + **Nested if Statements**
    - Complex conditions, such as verifying server availability or valid login credentials, are managed using nested if statements, ensuring the program logic handles all possible scenarios.

Chapter 6: Use of looping, especially using for and while loops. Nested looping is preferred when possible.

* **Accomplishment**
  + Implemented looping constructs to manage repetitive tasks and data processing.
* **Implementation**
  + **for Loops**
    - Used in methods like displayMenu() to iterate through all menu items and display their details.
  + **Enhanced for Loop**
    - Applied in the Restaurant class to aggregate data and perform operations on lists of tables, servers, and menu items.
  + **while Loops**
    - Used for input validation, prompting users repeatedly until valid data is provided.

Chapter 7: Use of string and string methods. Using the equals clause when comparing two strings. The correct use of the length methods. Converting strings to numbers.

* **Accomplishment**
  + Managed string operations effectively for user inputs and display formatting.
* **Implementation**
  + **String Comparisons**
    - Methods like equalsIgnoreCase() ensure that user input matches menu items and server names regardless of case sensitivity.
  + **StringBuilder**
    - Used to construct dynamic messages displayed to the user, such as order summaries and notifications.

Chapter 8: Your programs must use arrays and looping to create and/or access the arrays. The use of multi-dimensional or parallel arrays is required.

* **Accomplishment**
  + Utilized arrays and collections to manage dynamic lists of data, such as menu items, tables, and servers.
* **Implementation**
  + **ArrayLists**
    - The project uses ArrayLists to store dynamic data collections, allowing easy addition, removal, and traversal of elements.
  + **Multi-Dimensional Arrays**
    - Although not directly used, the concept is applied by managing multiple related collections (e.g., tables, servers, and menu) to handle complex data relationships.

Chapter 9: The use of inheritance and the super class in your programs is required. Use of the abstract class and implementation

* **Accomplishment**
  + While not directly implementing inheritance, the project is designed for future extensibility.
* **Implementation**
  + **Abstract Class and Inheritance Potential**
    - The MenuItem class could be extended to create specialized menu item types, such as Beverage or MainCourse, demonstrating a clear path for using inheritance and abstract classes in future iterations.

Chapter 10: You must use try catch blocks as much as possible throughout your programs

* **Accomplishment**
  + Incorporated robust error handling to manage unexpected situations and ensure a smooth user experience.
* **Implementation**
  + **try-catch Blocks**
    - Used in methods like saveData() and loadData() to handle I/O exceptions and inform the user of any issues encountered.
  + **Input Validation**
    - Ensures user inputs (e.g., table numbers, menu item names) are correctly processed, with informative error messages for invalid entries.

Chapter 11: You must create and use at least one file using correct I/O syntax and logic

* **Accomplishment**
  + Implemented file input and output operations for data persistence and retrieval.
* **Implementation**
  + **Serialization**
    - The Restaurant class uses serialization to save and load data, preserving the state of the restaurant between sessions.
  + **File Operations**
    - Methods like saveData() and loadData() handle file operations, ensuring that restaurant data (tables, servers, menu items) is stored and retrieved accurately.

Chapter 12: Recursion is only required if you are submitting Yummy’s or Sammy’s.

**INSTRUCTED NOT TO INCLUDE CHAPTER 12**

Chapter 13: Include any of linked lists or Generic Methods within your project.

* **Accomplishment**
  + Utilized dynamic data structures and generic methods to handle flexible data management.
* **Implementation**
  + **Dynamic Collections**
    - ArrayLists are used in place of traditional arrays for flexible data storage and manipulation, such as managing orders and menu items dynamically.
  + **Generic Methods**
    - Methods that operate on these lists ensure type safety and code reusability, providing a solid foundation for data management.

Chapter 14: Programs are expected to have button, event listeners, checkboxes or option buttons. You will need to use the JFrame, JLabel and other swing components as a part of your programs.

* **Accomplishment**
  + Developed a user-friendly graphical interface using Java Swing components, providing an intuitive and interactive user experience.
* **Implementation**
  + **Swing Components**
    - Used JFrame, JPanel, JButton, JOptionPane, and JTextArea to create a cohesive user interface for different roles (guest, server, admin).
  + **Event Handling**
    - Implemented ActionListener interfaces for all buttons, ensuring responsive and dynamic interactions based on user inputs.