**COMP1202 - Programming 1**

**Coursework 2**

**Name: BRANDON TING WEE KANG**

**Student ID: 33399085**

**Choice: B. Food Ordering System**

**Part 1. Relations Tree**

**Draw a simple tree that shows the relation between the classes. The diagram must show the class type (abstract or concrete), and the relationship type (implements, extends, or composed of).**

**Your overall implementation should include the following features of OOP:**

**1. Inheritance**

**2. Polymorphism**

**3. Interface**

**4. Abstract class**

Answer:

To create a relations tree for the given Java code, consider the following structure:

1. **MenuItem (Abstract Class)**: Extended by **FoodItem** and **DrinkItem**.
2. **Order (Concrete Class)**: Composed of **User**, **Restaurant**, **CartItem**, and **SpecialOffer**.
3. **Cart (Concrete Class)**: Composed of **CartItem**.
4. **CartItem (Concrete Class)**: Contains **MenuItem**.
5. **Restaurant (Concrete Class)**: Composed of **MenuItem**, **SpecialOffer**, and **Inventory**.
6. **Inventory (Concrete Class)**: Contains **MenuItem**.
7. **SpecialOffer (Concrete Class)**: Used in **Order**.
8. **User (Concrete Class)**: Contains **Order**.

The relationships are as follows:

* **MenuItem** is an abstract class.
* **FoodItem** and **DrinkItem** extend **MenuItem**, demonstrating inheritance.
* **Order**, **Cart**, **CartItem**, **Restaurant**, **Inventory**, **SpecialOffer**, and **User** are concrete classes.
* Composition is shown in classes like **Order**, which is composed of **User**, **Restaurant**, **CartItem**, and **SpecialOffer**.

**Part 4. Storytelling**

**Every programmer learns from mistakes. Write a story about:**

* **What problems did you encounter during the development of this coursework; what makes you stuck for many hours trying to figure out what's wrong.**
* **How did you manage to solve the problems.**
* **What are the things that you are really proud of about your program; imagine this is a business pitching session.**

Answer:

1. **Complexity and Design**: Developing a program like this can become complex quickly due to the interaction of multiple classes and components. Designing a clear and modular architecture from the beginning is crucial. Break down the problem into smaller, manageable pieces, and create well-defined classes and interfaces to represent different concepts (e.g., **Order**, **Restaurant**, **MenuItem**).

**Solution**: Use Object-Oriented Programming principles to structure your code, like encapsulation, inheritance, and abstraction. This makes the code more organized and easier to maintain.

1. **Handling User Input**: Interacting with users through a command-line interface can be challenging, especially when dealing with various input options and error handling.

**Solution**: Implement robust input validation and error handling mechanisms to guide users and prevent unexpected inputs. Utilize loops and conditionals to repeatedly prompt users until valid input is received.

1. **Testing and Debugging**: Debugging can be time-consuming, especially when issues arise from interactions between different components.

**Solution**: Use debugging tools available in your development environment to step through code and identify issues. Write unit tests for individual functions and components to catch errors early in the development process.

1. **Data Management**: Managing data, such as inventory, user orders, and order history, requires careful consideration of data structures and storage mechanisms.

**Solution**: Use appropriate data structures (e.g., lists, maps) and consider data persistence options (e.g., databases, file storage) based on the project's requirements. Ensure data consistency and integrity.

1. **Handling Special Offers**: Implementing special offers and discounts can be complex, especially when they need to be applied correctly to an order.

**Solution**: Create a clear and extensible mechanism for applying offers to orders. Test various scenarios to ensure that discounts are applied accurately.

1. **Code Maintenance**: As the project grows, maintaining and updating the code can become challenging.

**Solution**: Keep your code well-documented, use meaningful variable and function names, and follow coding best practices. Consider version control systems like Git to track changes and collaborate with others.

1. **User Experience**: Consider the overall user experience when interacting with your program. Ensure that it is intuitive and user-friendly.

**Solution**: Gather feedback from users (if possible) and make iterative improvements to the user interface and experience. Consider adding features like user authentication and personalized recommendations.

**Proud Points About the Program (Pitch)**:

* **Modular Design**: The program is structured using Object-Oriented Programming principles, making it modular and easy to extend with new features or components in the future.
* **User-Friendly Interface**: The command-line interface provides a straightforward way for users to interact with the restaurant's menu, place orders, and view order history.
* **Inventory Management**: The program effectively manages inventory levels, ensuring that items are available in the desired quantity before allowing users to place orders.
* **Special Offers**: Special offers and discounts are implemented, allowing users to benefit from promotions while ordering.
* **Data Persistence**: Although not included in the provided code, future enhancements could include data persistence, allowing the program to store and retrieve user order history and restaurant data.
* **Error Handling**: The program includes error handling and validation to guide users and prevent unexpected inputs.
* **Scalability**: The modular design and well-structured code make it easier to scale the program by adding more features, expanding the menu, or accommodating multiple restaurants.
* **Collaboration**: If this is a collaborative effort, teamwork and version control with tools like Git contribute to efficient development and maintenance.

Bugs