**Report on Restaurant Management System Implementation**

**Introduction**

The Restaurant Management System is a Java-based application designed to manage employees, customers, menu items, and orders in a restaurant. The system leverages key Object-Oriented Programming (OOP) concepts such as Inheritance, Encapsulation, Polymorphism, and Abstraction. Additionally, it incorporates file handling for saving order details and exception handling to ensure robustness. This report explains how these concepts and techniques are applied, along with the challenges faced during implementation.

**OOP Concepts in the Implementation**

**1. Inheritance:**The system uses inheritance to create a hierarchy of employees. The Employee class is an abstract base class, while Manager and Waiter are derived classes. This allows shared attributes (e.g., empId, name, salary) and methods (e.g., calculateBonus) to be defined in the base class, while specific behaviors (e.g., bonus calculation) are implemented in the derived classes.

**Example:** The Manager class overrides the calculateBonus method to provide a 20% bonus, while the Waiter class provides a 10% bonus.

**2. Encapsulation:**Encapsulation is achieved by making the attributes of classes private and providing public getter and setter methods. For example, in the Employee class, attributes like empId, name, and salary are private, and their values are accessed or modified through methods like getSalary() and getName().

This ensures data integrity and prevents unauthorized access to class attributes.

**3. Polymorphism:**Polymorphism is demonstrated through method overriding and the use of interfaces. For instance, the printDetails method is overridden in the Employee, Customer, and Order classes to provide specific implementations.

The Printable interface is used to enforce a common behavior (printDetails) across different classes, allowing polymorphic behavior when calling this method on objects of different types.

**4. Abstraction:**Abstraction is achieved through the use of abstract classes and interfaces. The Employee class is abstract, meaning it cannot be instantiated directly. It provides a template for derived classes like Manager and Waiter.

The Printable interface abstracts the printDetails method, allowing any class implementing it to define its own behavior.

**Role of File Handling and Exception Handling**

**1. File Handling:**The system uses file handling to save order details to a text file (orders.txt). The saveOrdersToFile method in the Main class writes customer and order information to the file using a PrintWriter.

This feature ensures that order data is persisted and can be reviewed later, even after the program terminates.

**2. Exception Handling:**Custom exceptions (InvalidInputException) are used to handle invalid user inputs, such as empty names, negative salaries, or invalid menu selections. These exceptions ensure that the program does not crash due to invalid data.

Try-catch blocks are used to handle IOException during file operations, ensuring that errors like file access issues are gracefully managed.

**Challenges Faced and Solutions**

**1. Challenge: Managing Complex User Input:** The system requires extensive user input for adding employees, customers, menu items, and orders. Validating this input was challenging.

Solution: Custom exceptions (InvalidInputException) and helper methods (getIntInput, getDoubleInput) were implemented to validate and handle user input effectively.

**2. Challenge: Ensuring Data Consistency:** Ensuring that orders are linked to valid customers and menu items was a challenge.

Solution: The system checks for the existence of customers and menu items before allowing orders to be placed. This prevents inconsistencies in the data.

**3. Challenge: File Handling Errors:** Writing to a file can fail due to various reasons, such as permission issues or file locks.

Solution: The saveOrdersToFile method uses a try-catch block to handle IOException, ensuring that the program does not crash and provides meaningful error messages.

#### ****Conclusion:****

The Restaurant Management System effectively utilizes OOP principles to create a modular and maintainable design. File handling ensures data persistence, while exception handling enhances the robustness of the system. Despite challenges like managing user input and ensuring data consistency, the system provides a functional and user-friendly solution for restaurant management. Future improvements could include a graphical user interface (GUI) and integration with a database for more efficient data management.