SHRI G.S. INSTITUTE OF TECHNOLOGY & SCIENCE,

INDORE INFORMATION TECHNOLOGY DEPARTMENT

Subject Code : IT28001

Subject Name : OBJECT ORIENTE PROGRAMMING

Date of Experiment : 12-08-2024 **Date of Submission :** 26-09-2024

Unit -1

Scenario 1: Library Management System

Scenario Description: You are tasked with developing a Library Management System for a small community library. The system should allow librarians to manage books, members, and loans. The system should keep track of the following:

- **Books:** Each book has a title, author, ISBN number, and availability status.
- **Members:** Each member has a name, membership ID, and a list of borrowed books.
- **Loans:** A member can borrow a book, and the system should record the date of borrowing and return.

Tasks:

1. **Design a class structure** to represent Books, Members, and Loans. Use appropriate data types and variables to represent the attributes.
2. **Implement methods** to:
    - Add new books to the system.

- Register new members.
- Borrow a book (changing the book's status to unavailable and adding it to the member's list).
- Return a book (updating the availability status and removing it from the member's list).

3. **Implement control structures** to check the availability of books before borrowing and to ensure that a member cannot borrow more than a specified number of books.
4. **Use an array** or another suitable data structure to store the list of books and members in the system.
5. **Incorporate final and static keywords** where necessary to define constants such as the maximum number of books a member can borrow.

Scenario 2: Employee Payroll System

Scenario Description: You are developing a payroll system for a company. The system should manage employee details, calculate salaries based on different parameters, and generate monthly payslips.

Tasks:

1. **Create a class hierarchy** to represent different types of employees (e.g., Full-Time, Part-Time, Contract). Include common attributes like name, employee ID, and base salary.
2. **Implement methods** to:

- Calculate the salary for different types of employees considering factors like hours worked (for part-time), overtime, bonuses, etc.
- Generate a payslip that includes the employee's details and salary breakdown.

3. **Use control structures** to ensure that salary calculations are correct based on the type of employee and to apply conditional bonuses based on performance ratings.
4. **Demonstrate the use of loops** to process payroll for multiple employees.
5. **Implement static and final variables** to define constants such as tax rates or company-wide bonuses.

Scenario 3: Online Shopping Cart

Scenario Description: You are building an online shopping cart system for an e-commerce platform. The system should allow customers to browse products, add them to their cart, and proceed to checkout.

Tasks:

1. **Design classes** to represent Products, Shopping Cart, and Customers. Include attributes like product name, price, quantity, and customer details.
2. **Implement methods** to:
   - Add and remove products from the shopping cart.
   - Calculate the total cost of the cart, including any applicable discounts.
   - Display the cart's contents before checkout.

3. **Use control structures** to apply different discount rules based on the total purchase amount or special promotions.
4. **Use an array or list** to store the products in the shopping cart and implement loops to iterate through the cart items for operations like calculating the total price.
5. **Incorporate static and final variables** to represent constants such as tax rates or discount percentages.

Scenario 4: Student Grading System

Scenario Description: You are tasked with creating a Student Grading System for a school. The system should store student information, record grades for various subjects, and calculate overall performance.

Tasks:

1. **Create a class structure** to represent Students and their Grades. Include attributes such as student name, ID, and a list of grades for different subjects.
2. **Implement methods** to:
    - Add grades for a student.
    - Calculate the average grade and determine the overall grade (e.g., A, B, C) based on the average.
    - Display a student's grades and overall performance.
3. **Use control structures** to determine the overall grade based on the average (e.g., if average > 90 then grade = 'A').

4. **Demonstrate the use of loops** to calculate the average grade across multiple subjects for a student.
5. **Use static and final variables** to define grade thresholds (e.g., A = 90-100, B = 80-89).

Scenario 5: Bank Account Management System

Scenario Description: You are developing a Bank Account Management System for a local bank. The system should allow customers to open accounts, deposit and withdraw money, and check their account balance.

Tasks:

1. **Design classes** to represent different types of bank accounts (e.g., Savings, Checking). Include attributes like account number, account holder's name, balance, and interest rate.
2. **Implement methods** to:
    - Deposit money into an account.
    - Withdraw money from an account (ensuring that the withdrawal does not exceed the available balance).
    - Calculate interest for savings accounts based on the balance.
3. **Use control structures** to validate transactions (e.g., preventing withdrawals that exceed the balance).
4. **Implement loops** to generate a report of all transactions for an account over a given period.

5. **Incorporate static and final variables** to define constants such as minimum balance requirements or fixed interest rates.

Unit –II

Scenario 1: Hospital Management System

Scenario Description: You are tasked with developing a Hospital Management System that handles various aspects such as patient information, doctor assignments, and appointment scheduling. The system should allow hospital staff to manage patient records, assign doctors to patients, and schedule appointments.

Tasks:

1. **Identify and define classes** for different entities in the system, such as Patient, Doctor, and Appointment. Define attributes and methods for each class, ensuring that the design represents real-world entities and their relationships.
2. **Implement encapsulation** by defining private attributes for the classes and providing public methods to access and modify these attributes.
3. **Create constructors** to initialize objects of each class with necessary attributes (e.g., a Patient object with a name, age, and medical history).
4. **Implement method overloading** to handle different types of appointment scheduling (e.g., scheduling with just a date, or with a date and time).
5. **Demonstrate relationships** between the classes:
    - **Association:** Show the relationship between Doctor and Patient.
    - **Aggregation:** A Doctor may have multiple Appointments.
    - **Composition:** A Patient has a MedicalHistory that is integral to the Patient object.

Scenario 2: E-Commerce Platform

Scenario Description: You are developing an E-Commerce platform that manages products, orders, and customers. The system should allow customers to browse products, place orders, and view order histories.

Tasks:

1. **Identify and define classes** for Product, Customer, and Order. Define the appropriate attributes and methods for each class, keeping in mind the relationships between them.
2. **Implement data hiding and encapsulation** by using private attributes for the classes and providing public getter and setter methods.
3. **Create constructors** for initializing Product, Customer, and Order objects with relevant attributes (e.g., a Product with a name, price, and stock quantity).
4. **Demonstrate method overloading** in the Order class to handle different types of order creation (e.g., an order with a single product or multiple products).
5. **Demonstrate relationships**:
    - **Association:** Between Customer and Order.
    - **Aggregation:** An Order contains multiple Product objects.
    - **Generalization/Specialization:** Create subclasses of Product (e.g., Electronics, Clothing) that specialize the Product class.

Scenario 3: University Enrollment System

Scenario Description: You are developing a University Enrollment System that manages students, courses, and faculty members. The system should allow students to enroll in courses, faculty members to manage course materials, and the university to track student progress.

Tasks:

1. **Identify and define classes** for Student, Course, and Faculty. Each class should have relevant attributes and methods.

2. **Implement encapsulation** by making class attributes private and providing public methods to access and modify them.
3. **Create constructors** for initializing objects, such as a Student with a name, ID, and list of enrolled courses.
4. **Implement method overloading** in the Course class to handle different ways of enrolling students (e.g., enrolling a student by ID or by student object).
5. **Demonstrate relationships**:
    - **Association:** Between Student and Course.
    - **Aggregation:** A Course contains multiple Student objects.
    - **Composition:** A Course has a Syllabus that is part of the Course object.
    - **Generalization/Specialization:** Create subclasses of Course (e.g., UndergraduateCourse, GraduateCourse).

Scenario 4: Hotel Booking System

Scenario Description: You are developing a Hotel Booking System that manages guests, room bookings, and payments. The system should allow guests to book rooms, make payments, and view booking details.

Tasks:

1. **Identify and define classes** such as Guest, Room, and Booking. Determine the necessary attributes and methods for each class.
2. **Implement encapsulation** by using private variables for the classes and providing public methods for accessing and updating these variables.
3. **Create constructors** to initialize Guest, Room, and Booking objects with relevant information.

4. **Demonstrate method overloading** in the Booking class to handle different ways of creating bookings (e.g., booking by room type or by specific room number).
5. **Demonstrate relationships**:
    - **Association:** Between Guest and Booking.
    - **Aggregation:** A Booking may include multiple Room objects.
    - **Composition:** A Booking has a Payment object that is essential to the Booking.
    - **Generalization/Specialization:** Create subclasses of Room (e.g., SingleRoom, DoubleRoom) that inherit from the Room class.

Scenario 5: Banking System

Scenario Description: You are tasked with developing a Banking System that handles customer accounts, transactions, and loans. The system should allow customers to manage their accounts, perform transactions, and apply for loans.

Tasks:

1. **Identify and define classes** for Customer, Account, and Transaction. Each class should have relevant attributes and methods.
2. **Implement data hiding and encapsulation** by using private attributes and providing public methods for accessing and modifying them.
3. **Create constructors** to initialize Customer, Account, and Transaction objects with necessary attributes.
4. **Implement method overloading** in the Transaction class to handle different types of transactions (e.g., deposit, withdrawal).
5. **Demonstrate relationships**:
    - **Association:** Between Customer and Account.
    - **Aggregation:** A Customer may have multiple Account objects.

- **Composition:** An Account has a TransactionHistory that is integral to the account.
- **Generalization/Specialization:** Create subclasses of Account (e.g., SavingsAccount, CheckingAccount) that inherit from the Account class.

UNIT-III

Scenario 1: Automated Vehicle Management System

Scenario Description: You are developing an Automated Vehicle Management System for a transportation company. The system needs to manage different types of vehicles (e.g., cars, trucks, buses) and their operations such as driving, refueling, and maintenance.

Tasks:

1. **Identify and define a base class** Vehicle that includes common attributes and methods like fuelType, maxSpeed, and drive(). Then, create subclasses for specific vehicle types such as Car, Truck, and Bus.
2. **Implement hierarchical inheritance** by designing the class structure such that all specific vehicle types inherit from the Vehicle base class. Demonstrate how different vehicles share common functionality while also having unique behaviors.
3. **Use the super keyword** to call methods and constructors from the base class within the subclasses. For example, initialize common attributes in the base class and unique attributes in the subclass.
4. **Override methods** in the subclasses to demonstrate different behaviors (e.g., overriding the drive() method to reflect different driving styles for cars, trucks, and buses).
5. **Create an abstract class or interface** for a feature like Maintenance that all vehicle types must implement. Discuss the difference between using an abstract class and an interface for this purpose.
6. **Model the class hierarchy** using UML to show the relationships between the base class, subclasses, and interfaces.

Scenario 2: Online Education Platform

Scenario Description: You are tasked with building an Online Education Platform that manages different types of courses (e.g., Science, Arts, Technology) and their associated content, including lessons and assignments. The platform should support both instructors and students.

Tasks:

1. **Design a base class** Course that includes common attributes and methods like courseName, courseCode, and startCourse(). Create subclasses for specific course types such as ScienceCourse, ArtsCourse, and TechnologyCourse.
2. **Implement multi-level inheritance** by adding another level to the hierarchy, such as having a ProgrammingCourse class that inherits from TechnologyCourse. Demonstrate how attributes and methods are inherited across multiple levels.
3. **Use final with inheritance** to prevent further inheritance of a class or to prevent method overriding. For example, prevent the ProgrammingCourse class from being subclassed further.
4. **Override methods** in the subclasses to customize behavior for different course types (e.g., overriding the startCourse() method to include specific course materials).
5. **Implement interfaces** for features like Interactive that all courses should implement (e.g., methods for live sessions or quizzes). Compare the use of interfaces vs. abstract classes for this scenario.
6. **Model the class hierarchy and relationships** using UML, highlighting the inheritance structure, overridden methods, and interfaces.

Scenario 3: Smart Home Automation System

Scenario Description: You are developing a Smart Home Automation System that manages various home devices (e.g., lights, thermostats, security cameras) and allows for automation rules (e.g., turning off lights when no one is home).

Tasks:

1. **Define a base class** SmartDevice that includes common attributes like deviceName, status, and methods like turnOn() and turnOff(). Create subclasses for specific device types such as Light, Thermostat, and SecurityCamera.
2. **Implement polymorphism** by designing methods that can operate on the base class SmartDevice but behave differently based on the actual subclass (e.g., turning on a light vs. turning on a thermostat). Demonstrate both compile-time and run-time polymorphism.
3. **Use abstract classes** to create a common interface for all devices but leave the implementation details to the specific device classes. For example, create an abstract method scheduleAutomation() that each device must implement differently.
4. **Override methods** in the subclasses to tailor the behavior of each device (e.g., overriding turnOn() for a Light to adjust brightness).
5. **Implement multiple inheritance** using interfaces, such as a BatteryPowered interface that some devices implement in addition to their main functionality. Discuss the role of interfaces in this context.
6. **Model the relationships** between classes and interfaces using UML, showing how the Smart Home Automation System manages different types of devices and their behaviors.

Scenario 4: Banking System with Multiple Account Types

Scenario Description: You are developing a Banking System that supports multiple account types such as Savings Account, Checking Account, and Business Account. Each account type has different features, such as interest calculation, overdraft protection, and transaction limits.

Tasks:

1. **Create a base class** BankAccount with attributes like accountNumber, balance, and methods like deposit(), withdraw(), and calculateInterest(). Develop subclasses for SavingsAccount, CheckingAccount, and BusinessAccount.
2. **Implement method overriding** in subclasses to provide specific implementations for methods like calculateInterest() (e.g., different interest rates for different account types).

3. **Use hierarchical inheritance** to manage common functionality across different account types while allowing for specialization in each subclass.
4. **Create abstract classes or interfaces** to represent operations that are common across different accounts but require specific implementations, such as OverdraftProtection. Discuss when to use abstract classes vs. interfaces.
5. **Demonstrate multiple inheritance** by implementing an interface like Taxable, which some accounts (e.g., BusinessAccount) might need to implement in addition to their main class hierarchy.
6. **Model the class hierarchy and relationships** using UML, illustrating how different account types inherit from the base class and implement additional features through interfaces.

Scenario 5: Hotel Reservation System

Scenario Description: You are developing a Hotel Reservation System that manages different types of rooms (e.g., Standard Room, Deluxe Room, Suite) and their reservations. The system should also handle various services offered by the hotel, such as room service, laundry, and spa.

Tasks:

1. **Design a base class** Room with attributes like roomNumber, roomType, and methods like bookRoom() and checkout(). Create subclasses for StandardRoom, DeluxeRoom, and Suite.
2. **Implement multiple inheritance** using interfaces to add additional functionalities such as RoomService and LaundryService. Demonstrate how a Suite might implement both interfaces, while a StandardRoom might only implement RoomService.
3. **Use the super keyword** to access and modify the base class's attributes and methods in subclasses. For example, initialize room features in the base class and customize them in the subclass.

4. **Override methods** in subclasses to reflect different booking processes or amenities (e.g., a Suite might include complimentary services).
5. **Demonstrate polymorphism** by implementing methods that work with the base class Room but behave differently based on the actual room type (e.g., checking in guests to different room types).
6. **Model the class structure and relationships** using UML, showing how different room types inherit from the base class and implement additional services through interfaces.

Unit –IV

Scenario 1: Inventory Management System

Scenario Description: You are developing an Inventory Management System for a retail store. The system manages products, stock levels, orders, and suppliers. The system should handle multiple tasks such as updating stock, placing orders, and managing supplier information concurrently. It also needs to be robust enough to handle exceptions such as out-of-stock situations, supplier delays, and invalid product entries.

Tasks:

1. **Create and organize packages**:
    - Define packages like inventory, orders, and suppliers.
    - Implement classes like Product, Order, and Supplier within these packages.
    - Demonstrate how to import and use classes from different packages.
2. **Implement exception handling**:
    - Handle exceptions for cases like OutOfStockException and InvalidProductException.
    - Use try, catch, throw, throws, and finally to manage these exceptions.
    - Create custom exception subclasses to provide more specific error handling.
3. **Develop a multithreaded system**:
    - Create threads to handle different tasks such as processing orders and updating inventory simultaneously.
    - Use thread priorities to prioritize critical tasks (e.g., updating stock before placing new orders).
    - Implement thread synchronization to ensure consistent updates to stock levels.
4. **Demonstrate thread lifecycle**:

- Show how threads move through different states (e.g., New, Runnable, Blocked, Terminated) in the context of processing an order.

5. **Model inter-thread communication**:
    - Implement a system where threads communicate to update inventory and notify when stock levels fall below a certain threshold.
    - Use daemon threads for background tasks like daily inventory audits.

Scenario 2: Online Banking System

Scenario Description: You are building an Online Banking System that handles multiple account operations such as deposits, withdrawals, and transfers. The system must be able to perform these operations concurrently without any race conditions. It should also manage exceptions like insufficient funds, invalid account details, and transaction failures gracefully.

Tasks:

1. **Design and implement packages**:
    - Create packages like accounts, transactions, and customers.
    - Implement classes such as BankAccount, Transaction, and Customer within these packages.
    - Demonstrate how to define, create, and access packages, and properly set the CLASSPATH.
2. **Handle exceptions effectively**:
    - Use exception handling for scenarios like InsufficientFundsException and InvalidAccountException.
    - Demonstrate the use of custom exceptions to enhance the robustness of the system.
    - Implement exception hierarchy and utilize try, catch, throw, throws, and finally blocks.

3. **Create a multithreaded environment**:
    - Develop threads to handle multiple transactions concurrently, such as deposits and withdrawals.
    - Prioritize threads handling critical operations, such as large transfers.
    - Synchronize threads to avoid issues like double withdrawal from the same account.
4. **Understand and implement thread lifecycle**:
    - Demonstrate how threads are created, started, and terminated in the context of processing a bank transaction.
    - Show how to properly manage thread states using appropriate methods.
5. **Implement inter-thread communication**:
    - Use threads to monitor account balances and trigger alerts when certain conditions are met (e.g., low balance).
    - Create daemon threads for background processes like daily transaction logging.

Scenario 3: E-commerce Platform

Scenario Description: You are tasked with developing an E-commerce Platform that handles product listings, customer orders, and payment processing. The platform needs to support multiple users simultaneously, managing their interactions with the system efficiently. It should also handle various exceptions such as payment failures, out-of-stock products, and invalid user inputs.

Tasks:

1. **Organize the system into packages**:
    - Define packages such as products, orders, and payments.
    - Implement classes like Product, Order, and PaymentProcessor within these packages.

- Demonstrate the correct usage of CLASSPATH and how to import and access classes from these packages.

2. **Implement robust exception handling**:
    - Create and manage exceptions like PaymentFailureException and ProductUnavailableException.
    - Use try, catch, throw, throws, and finally to manage these exceptions and ensure the system remains robust.
    - Develop custom exceptions that provide specific error messages for better user feedback.
3. **Implement multithreading for concurrent operations**:
    - Develop threads to handle simultaneous user actions such as adding items to the cart, placing orders, and processing payments.
    - Set thread priorities for time-sensitive tasks like payment processing.
    - Use thread synchronization to manage access to shared resources like product inventory.
4. **Demonstrate thread lifecycle management**:
    - Illustrate the thread lifecycle by managing order processing tasks, showing how threads are started, managed, and terminated.
    - Use appropriate thread methods to control the execution flow and state of threads.
5. **Implement inter-thread communication**:
    - Develop a system where threads communicate to ensure consistent updates to order status and inventory levels.
    - Use daemon threads for tasks like order history maintenance and periodic sales reporting.

Scenario 4: Ride-Sharing Application

Scenario Description: You are developing a Ride-Sharing Application that allows users to book rides, drivers to accept bookings, and the system to manage real-time ride data. The application needs to handle multiple ride requests simultaneously, manage driver assignments, and handle exceptions such as no available drivers, invalid booking details, and payment issues.

Tasks:

1. **Create and manage packages**:
    - Define packages such as rides, users, and payments.
    - Implement classes like Ride, Driver, and Payment within these packages.
    - Demonstrate how to define, create, access packages, and manage the CLASSPATH for this project.
2. **Implement robust exception handling**:
    - Handle exceptions like NoDriverAvailableException and InvalidRideRequestException.
    - Use try, catch, throw, throws, and finally to manage these exceptions and maintain system robustness.
    - Create custom exception subclasses to provide specific handling for different error conditions.
3. **Develop a multithreaded environment**:
    - Create threads to handle multiple ride bookings and driver assignments concurrently.
    - Assign thread priorities to ensure that critical tasks like ride booking and payment processing are handled promptly.
    - Synchronize threads to prevent race conditions in driver assignment and ride status updates.

4. **Demonstrate thread lifecycle**:
    - Show how threads are used to manage different stages of a ride, from booking to completion.
    - Demonstrate the thread lifecycle in the context of accepting and completing rides.
5. **Implement inter-thread communication**:
    - Use threads to monitor ride requests and assign available drivers in real time.
    - Implement daemon threads for background tasks like ride history tracking and system health checks.

Scenario 5: Hotel Reservation System

Scenario Description: You are developing a Hotel Reservation System that allows customers to book rooms, the hotel to manage room availability, and the system to handle payments. The system needs to support concurrent room bookings and manage tasks like room assignment and payment processing. It should also handle exceptions like overbooking, invalid payment details, and booking cancellations.

Tasks:

1. **Organize the system into packages**:
    - Define packages such as rooms, reservations, and payments.
    - Implement classes like Room, Reservation, and Payment within these packages.
    - Demonstrate how to set up and manage packages and CLASSPATH in the project.
2. **Implement robust exception handling**:
    - Handle exceptions like OverbookingException and InvalidPaymentException.
    - Use try, catch, throw, throws, and finally to manage these exceptions and maintain the system's robustness.

- Create custom exception subclasses for specific scenarios like booking cancellations.

3. **Develop a multithreaded environment**:
    - Create threads to handle multiple room bookings, cancellations, and payment processing simultaneously.
    - Set thread priorities to ensure that critical tasks like booking confirmation and payment processing are handled efficiently.
    - Synchronize threads to avoid conflicts in room assignments and ensure consistent updates to room availability.
4. **Demonstrate thread lifecycle**:
    - Show how threads manage different stages of a reservation, from booking to check-out.
    - Use thread methods to control the execution flow and state of threads during the reservation process.
5. **Implement inter-thread communication**:
    - Use threads to manage real-time updates to room availability and booking status.
    - Implement daemon threads for background tasks like room availability audits and reservation history management.

Unit-V

Scenario 1: File Management System

Scenario Description: You are developing a File Management System that allows users to perform various file operations such as reading, writing, and updating files. The system should support both text (character streams) and binary (byte streams) files. Additionally, it should handle large files efficiently using random access files.

Tasks:

1. **Implement basic file operations**:
    - Write a program to read from and write to text files using character streams (FileReader and FileWriter).
    - Write a program to read from and write to binary files using byte streams (FileInputStream and FileOutputStream).
2. **Handle random access files**:
    - Create a program that uses RandomAccessFile to read and write at specific positions within a large file.
    - Demonstrate how to seek to a particular position in the file and update content.
3. **Work with directories**:
    - Implement a program that lists all files in a directory and allows users to choose a file for reading or writing.
    - Extend the program to allow the creation and deletion of files within a directory.
4. **Error handling in file operations**:
    - Write a program that includes robust exception handling for common file-related errors, such as FileNotFoundException, IOException, and EOFException.
5. **Integrate file operations with collections**:

- Create a program that reads a list of names from a text file and stores them in a List or Set collection.
- Implement functionality to add, remove, and update names in the collection, and then write the updated list back to the file.

Scenario 2: Digital Library System

Scenario Description: You are tasked with creating a Digital Library System that manages eBooks. The system should allow users to upload, download, and read eBooks, which can be stored as text files. Additionally, the system should provide random access to specific chapters or sections within an eBook.

Tasks:

1. **Implement file operations for eBooks**:
   - Write a program to upload (write) and download (read) eBooks using character streams.
   - Implement functionality to read eBooks as binary files for faster access.
2. **Random access to eBooks**:
   - Develop a program that allows users to jump to a specific chapter or section in an eBook using RandomAccessFile.
   - Implement a table of contents feature that maps chapter titles to file positions for quick access.
3. **Integrate collections for book management**:
   - Use a Map to associate each eBook with its metadata (e.g., title, author, file location).
   - Implement functionality to search for books by title or author and display their metadata.

4. **Error handling in eBook operations**:
    - Ensure the system handles errors gracefully, such as missing eBook files or corrupted data, by implementing exception handling.
5. **Extend the system with AWT for a GUI**:
    - Create a simple AWT-based user interface for browsing the digital library, uploading new books, and reading existing ones.
    - Implement buttons for file operations like "Upload", "Download", "Read", and "Search".

Scenario 3: Online Quiz Applet

Scenario Description: You are developing an Online Quiz system using Java Applets. The applet will display multiple-choice questions, accept user inputs, and display results. The applet should be able to receive parameters like the number of questions and the time limit for the quiz from the web page that hosts it.

Tasks:

1. **Create a basic applet for the quiz**:
    - Write a simple applet that displays a welcome message and the first question.
    - Implement the applet's lifecycle methods: init(), start(), stop(), and destroy().
2. **Handle parameters passed to the applet**:
    - Write a program that passes parameters like the number of questions and time limit to the applet.
    - Access these parameters in the init() method and use them to configure the quiz.
3. **Extend the applet with interactive components**:
    - Use AWT components like Label, Button, Checkbox, and TextField to create the quiz interface.

- Implement event handling to record user responses and navigate between questions.

4. **Error handling in applets**:
    - Implement error handling to manage scenarios like missing parameters or invalid user inputs (e.g., non-numeric values for the time limit).
5. **Difference between applets and applications**:
    - Write a small comparison program demonstrating the differences between a standalone Java application and an applet performing the same task (e.g., displaying quiz questions).

Scenario 4: Employee Management System with AWT

Scenario Description: You are developing an Employee Management System with a graphical user interface (GUI) using Java AWT. The system should allow the management of employee records, including adding, updating, and viewing employee information.

Tasks:

1. **Design the GUI with AWT**:
    - Create an AWT-based interface that includes forms for entering employee details (e.g., name, ID, department).
    - Use AWT components such as Label, TextField, Button, and TextArea to design the form.
2. **Handle file operations**:
    - Implement file operations to save employee details to a text file and retrieve them.
    - Write a program to load employee data from the file when the application starts and display it in the GUI.
3. **Integrate collections for employee management**:
    - Use a Map to associate employee IDs with their details.

- Implement functionality to add, update, and remove employee records from the collection and synchronize these changes with the file.

4. **Implement error handling**:
    - Ensure that the system handles errors like duplicate employee IDs, missing fields, and file I/O errors gracefully.
5. **Enhance with random access files**:
    - Extend the system to allow random access to specific employee records using RandomAccessFile.
    - Implement functionality to update specific records without re-writing the entire file.

Scenario 5: Personal Finance Manager

Scenario Description: You are developing a Personal Finance Manager that tracks income, expenses, and savings. The system should allow users to record transactions, categorize them, and view reports. The application should provide both text-based and graphical interfaces, with file operations to save and load data.

Tasks:

1. **Create a package structure**:
    - Organize the application into packages such as transactions, reports, and user.
    - Implement classes within these packages to handle different aspects of personal finance management.
2. **Implement file operations**:
    - Write a program to save transaction records to a file using character streams.
    - Implement functionality to load transactions from a file when the application starts.
3. **Design a GUI with AWT**:

- Create a simple AWT-based interface for entering transactions, viewing summaries, and generating reports.
- Use AWT components like Frame, Button, TextField, and List to build the interface.

4. **Use collections and maps**:
   - Store transaction records in a List and categorize them using a Map.
   - Implement features to filter transactions by category and date, and display summaries.
5. **Error handling and robustness**:
   - Implement robust exception handling to manage file errors, invalid input, and empty fields.
   - Ensure the system can recover from common issues without crashing.
6. **Random access to transactions**:
   - Extend the program to allow random access to specific transactions using RandomAccessFile, enabling quick updates or deletions without processing the entire file.