Advanced Java Programming Lab Sheet
IV Year /I Part
Faculty: CSIT

Lab sheet 1

Objectives:

1. Arrays, for-each Loops, Interface, Inheritance
2. Overloading, Interface, Inheritance
3. Overloading, Inheritance, Overriding
4. Try, Catch, Finally
5. Throws and Throw
6. Creating Exception Class
7. Thread States, Writing Multithreaded Programs
8. Thread Priorities
9. Working with Files
10. Reading and Writing Objects

Questions:

1. Define an interface called Shape with methods calculateArea() and displayInfo(). Create three classes that implement this interface: Circle, Rectangle, and Triangle. Each class should calculate its area using different formulas based on their respective shapes (Pi * r^2 for the circle, length * width for the rectangle, and (base * height) / 2 for the triangle).
   Use for-each loops to store these shapes in an array, calculate and display their areas, and call the displayInfo() method for each shape.

2. Create a class hierarchy for a library system. Define a base class called Item with attributes title and itemID. Derive two subclasses from Item: Book and DVD. Implement overloaded constructors for each class to allow different ways of initializing the attributes. Additionally, define an interface called Borrowable with methods checkOut() and returnItem(). Implement this interface in both Book and DVD classes with custom implementations of the methods to manage borrowing and returning. Write a program that demonstrates the creation and management of items in the library, including checking out and returning items.

3. Create a class Vehicle with attributes brand and model. Derive two subclasses from Vehicle: Car and Bicycle. Implement overloaded constructors in each class to set the brand and model attributes in various ways. Define a method printInfo() in the base class Vehicle that displays the brand and model. Override the printInfo() method in both Car and Bicycle classes to include additional information specific to each type of vehicle. Finally, create objects of both Car and Bicycle, call the printInfo() method on each, and display their complete information.

4. Create a Java program that reads an integer from the user. Use a try-catch block to handle potential exceptions when parsing the user input. If the input is a valid integer, print the square of that integer. If an exception occurs, display an error message, and then execute a finally block to close any resources or perform cleanup tasks.

5. Define a method called divide that takes two integers as arguments and divides them. Handle division by zero using a try-catch block and throw a custom exception called DivisionByZeroException when such an error occurs. In the catch block, catch the exception, print an error message, and then rethrow the exception. In your main program, call the divide method with different inputs and catch and handle the exceptions appropriately.

6. Create a custom exception class called NegativeValueException that extends the Exception class. This exception should be thrown when a negative value is encountered. Write a program that prompts the user to enter a positive integer, and if a negative value is provided, throw the NegativeValueException. Use a try-catch block to handle this exception and display an error message. Ensure that the custom exception is appropriately caught and handled.

7. Create a Java program that simulates a simple ticket booking system using multiple threads. Define a Ticket class with a book method that simulates booking a ticket by sleeping for a random time (representing the time taken for the booking process). Create multiple threads, each representing a user trying to book a ticket. Start these threads concurrently and observe the thread states (e.g., NEW, RUNNABLE, BLOCKED, TERMINATED) using appropriate methods. Ensure that only one user can book a ticket at a time to prevent overbooking.

8. Write a Java program that demonstrates thread priorities. Create three threads with different priorities: low, normal, and high. Each thread should print a message with its priority. Start all the threads and observe the execution order. Discuss the impact of thread priorities on thread scheduling and execution.

9. Create a Java program that reads a text file named "input.txt" and counts the number of words in the file. Implement the following steps:
    - Open and read the contents of the file.
    - Count the number of words in the file.
    - Display the total word.

10. Create a Java program that demonstrates object serialization and deserialization. Define a class Student with attributes like name, roll number, and grade. Write methods to serialize and deserialize objects of this class. Perform the following tasks:

    - Create several Student objects and write them to a file named "students.ser" using object serialization.
    - Read the objects from the file and display their details.