Termwork No:1

Title:

Write a menu driven Java program to work with files.

Theory:

File handling is a crucial aspect of many programming tasks, and Java provides a robust set of tools for working with files. In the context of the first term work, a menu-driven Java program has been designed to facilitate various file operations. The utilization of classes such as File, FileReader, and FileWriter enables the reading and writing of file contents efficiently. The program prompts the user to enter source and target file names and ensures their existence through the exists() method, showcasing input validation for enhanced program robustness. The menu-driven approach, a common user interface strategy, empowers users to select desired operations interactively. This term work not only demonstrates practical file handling but also introduces fundamental concepts of user-friendly menu-driven program design.

Exception Handling: Discuss the importance of incorporating try-catch blocks for handling exceptions during file operations, ensuring the program gracefully manages errors.

File Closing: Emphasize the significance of closing file resources using the **close()** method to prevent resource leaks and enhance program efficiency.

Concurrency Considerations: Briefly mention the implications of concurrent file access and how Java provides mechanisms to address potential issues using locks or synchronization.

Termwork No:2

Title:

Write a Java Program to demonstrate the implementation of stream classes in Java.

Theory:

Byte streams in Java, exemplified by ByteArrayInputStream and ByteArrayOutputStream, play a crucial role in handling binary data efficiently. In the second term work, a Java program is designed to demonstrate the implementation of these stream classes. The user is prompted to input a destination file name, introducing interactive user input. The menu-driven program allows the transfer of contents from an input file to the destination file using ByteArrayInputStream, showcasing the power of in-memory byte stream handling. This term work illustrates the efficiency of byte streams in scenarios where raw binary data processing is required, providing a deeper insight into Java's capabilities for handling byte-oriented data.

Buffering Techniques: Discuss the advantages of using buffered streams (**BufferedInputStream** and **BufferedOutputStream**) for improved performance in reading and writing byte streams.

Serialization: Introduce the concept of object serialization, highlighting how it can be applied to save and restore complex data structures to and from byte streams.

Error Handling in Streams: Explain the importance of handling exceptions when working with streams, emphasizing the need for proper error checking and reporting.

Termwork No:3

Title:

Write a Java Program to demonstrate the implementation of reading and writing binary data in Java.

Theory:

The third term work focuses on reading and writing binary data in Java, a fundamental aspect of low-level data manipulation. The program prompts the user for source and destination file names, emphasizing the importance of user interaction. Additionally, it accepts user-defined text to be written to the source file. The program then implements a unique approach by writing every alternate byte from the source to the destination file, demonstrating the versatility of binary data manipulation. Furthermore, the program compares the properties of the source and destination files, encompassing size, permissions, and other metadata. This term work provides a hands-on experience in working with binary data, emphasizing the importance of data integrity and file property comparison.

Data Integrity and Checksums: Discuss strategies for ensuring data integrity during binary data manipulation, such as incorporating checksums or hash functions.

Byte Array Manipulation: Explain the versatility of byte array manipulation and how it can be applied to implement custom data processing routines efficiently.

File Metadata Access: Provide insights into accessing and interpreting file metadata using the **java.nio.file** package, allowing for more detailed property comparisons.

Termwork No:4

Title:

Write a menu-driven Java Program to work with ArrayLists.

Theory:

The fourth term work delves into ArrayList operations, showcasing the flexibility of dynamic arrays in Java. The menu-driven Java program allows users to create ArrayLists of integers and floats with user-specified lengths, encapsulating the principles of user-driven dynamic data structures. Overloaded methods are introduced to add and remove elements, providing a clear illustration of the benefits of method overloading. Additionally, the program includes overloaded methods for performing linear searches on both integer and float ArrayLists, highlighting the adaptability of Java methods to different data types. By creating objects to demonstrate these functionalities, this term work reinforces object-oriented programming concepts, encapsulation, and the modularity of code design. Overall, this term work provides a comprehensive exploration of ArrayList operations in a user-friendly and interactive manner.

Generic Types: Introduce generic types in Java and how they can be applied to create ArrayLists capable of storing elements of any data type, enhancing code flexibility.

Collections Framework: Highlight the broader context of the Java Collections Framework, explaining how ArrayList is just one example of the many data structures provided by Java.

Error Handling in Collections: Discuss the importance of handling exceptions and potential errors when performing operations on ArrayLists, promoting robust code practices.

Termwork No: 5

Title:

Write a multithreaded Java program to work with numbers.

Theory:

Multithreading in Java is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.Java Multithreading is mostly used in games, animation, etc.

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved in two ways:

1. Process-based Multitasking (Multiprocessing)
2. Thread-based Multitasking (Multithreading)

A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.

Java Thread Methods:

S.N.	Modifier and Type	Method	Description
1)	void	start()	It is used to start the execution of the thread.
2)	void	run()	It is used to do an action for a thread.
3)	static void	sleep()	It sleeps a thread for the specified amount of time.
4)	static Thread	currentThread()	It returns a reference to the currently executing thread object.
5)	void	join()	It waits for a thread to die.

Termwork No: 6

Title:

Write a Java program to demonstrate how the standard operations on a bank account can be synchronized.

Theory:

Synchronization in Java is crucial when multiple threads access shared resources to avoid data inconsistency and potential conflicts. In the context of a bank account, you want to ensure that operations like deposit and withdrawal are atomic and not interrupted by other threads.

In multithreading, when multiple threads are executing simultaneously, thread safety ensures that the shared resources are accessed in a way that avoids data corruption or inconsistencies.

Synchronization in Java is a mechanism to control the access of multiple threads to shared resources. It ensures that only one thread can access the critical section of code at a time.

A critical section is a part of the code where shared resources are accessed. It needs to be synchronized to avoid race conditions and maintain data integrity.

In Java, the synchronized keyword is used to define a block of code or a method that can be accessed by only one thread at a time. It can be applied to methods or blocks.

The wait() and notify() methods in Java are used for inter-thread communication. They enable threads to wait for a specific condition and notify others when the condition is met. They are often used in conjunction with thesynchronized keyword.

Besides using synchronization, other strategies for achieving thread safety include immutability (making objects unmodifiable), using thread-safe data structures, and employing the volatile keyword for shared variables that don't participate in compound actions.

Termwork No: 7

Title: Write a multithreaded Java program to demonstrate the Producer-Consumer problem

Theory: The Producer-Consumer problem is a classic challenge in concurrent programming, illustrating the intricacies of coordinating the activities of two types of threads – producers and consumers – that share a common, fixed-size buffer. The shared resource, typically represented as a buffer or queue, serves as an intermediary for data exchange between producers and consumers. The primary objective is to ensure the smooth and safe operation of these threads in a concurrent environment, avoiding issues like race conditions and deadlocks.

At its core, the problem requires effective synchronization mechanisms to manage access to the shared resource. Synchronization techniques such as locks, semaphores, or condition variables are employed to regulate the interaction between producers and consumers. Producers, responsible for generating data items, attempt to insert them into the buffer. If the buffer is full, producers must wait until space becomes available. Successful insertion prompts the producer to notify waiting consumers. Conversely, consumers retrieve and process items from the buffer. If the buffer is empty, consumers wait until items are available, subsequently notifying waiting producers after consumption.The challenge extends beyond basic functionality, requiring strategies to address potential issues such as starvation and deadlocks. Starvation prevention involves ensuring fairness in scheduling to avoid threads being perpetually blocked. Deadlocks, where threads wait indefinitely for conditions that cannot be met, necessitate careful resource and lock management.

Real-world applications, including print spooling, task scheduling, and job processing, echo the dynamics of the Producer-Consumer problem. The concept provides valuable insights into synchronization techniques, inter-process communication, and the complexities of managing shared resources in multithreaded environments. Successfully addressing this problem requires a delicate balance, emphasizing efficient coordination while steering clear of common pitfalls like race conditions and deadlocks. Mastery of the Producer-Consumer problem is essential for developers navigating the challenges of concurrent programming and ensuring the reliability of multithreaded applications.

Termwork No: 8

Title: 8. Write a Java program to search and display details of book(s) authored by a particular author from a “BOOKS” table. Assume an appropriate structure and attributes for the table.

Theory: The task involves creating a Java program to search and display details of books authored by a specific author from a "BOOKS" table, assuming an appropriate structure and attributes for the table.

The Java program utilizes JDBC (Java Database Connectivity) to interact with the database. It begins by establishing a connection to the database using the DriverManager class. The assumed "BOOKS" table likely includes fields such as bookID, title, author, publisher, and publicationYear.The program constructs a SQL query to retrieve book details based on the specified author. It utilizes a PreparedStatement to safely set the author parameter in the query. Upon execution, the ResultSet is processed to retrieve and display book details, including book ID, title, author, publisher, and publication year.

Proper resource management is ensured by closing the PreparedStatement and the database connection within a try-with-resources block. The provided example assumes a MySQL database, and adjustments may be needed for different databases.This program serves as a practical illustration of database connectivity in Java, showcasing how to seamlessly integrate Java applications with relational databases for data retrieval and display based on specified criteria. The Java program presented addresses the task of searching and displaying book details by a specific author from a "BOOKS" table, demonstrating an effective use of JDBC for database interaction. By establishing a connection to the database, constructing a parameterized SQL query, and employing a PreparedStatement for secure execution, the program ensures robustness and security in handling database operations. The ResultSet processing and subsequent display of book details showcase the seamless integration of Java with relational databases. Through this concise and well-structured example, developers gain insights into the practical implementation of database queries in Java, emphasizing readability, efficiency, and adherence to best practices in database connectivity.

Termwork No: 9

Title: Program to demonstrate transaction processing. Assume an appropriate database/table

Theory: The presented Java program serves as a demonstration of transaction processing, highlighting the importance of maintaining the ACID (Atomicity, Consistency, Isolation, Durability) properties in database operations. Focusing on a hypothetical "BANK" database and an "ACCOUNTS" table, the program establishes a connection using JDBC to interact with the database. Within the transaction, the setAutoCommit(false) method is employed to allow multiple SQL operations to be treated as a single unit of work.

Two sample operations, representing a withdrawal and a deposit, are encapsulated within the transaction. The updateBalance method, utilizing a PreparedStatement, ensures the atomicity of the database updates. If any part of the transaction encounters an SQLException, the program rolls back the entire transaction to maintain consistency.The program emphasizes proper resource management by committing the transaction only when all operations succeed, ensuring the durability of the changes. Conversely, in case of an exception, the entire transaction is rolled back to preserve the database's integrity.

This example provides insights into implementing transactional behavior in Java applications, showcasing how to handle complex database operations while ensuring data consistency and reliability. Developers can adapt this foundational understanding to more complex scenarios, enhancing the robustness of their applications in real-world database interactions.

In this illustrative Java program, transaction processing is exemplified through a practical scenario in a banking context. The code effectively encapsulates a series of database operations within a transaction, reflecting the intricate dance of withdrawals and deposits on a hypothetical "ACCOUNTS" table in a "BANK" database. By initiating a transaction and carefully orchestrating database updates, the program adheres to the principles of atomicity, consistency, isolation, and durability. Notably, the graceful handling of exceptions ensures that the entire transaction is rolled back in the event of any unforeseen issues, underscoring the significance of data integrity. This program serves as a valuable reference for developers seeking to implement robust and secure transaction processing in Java, providing a foundational understanding of how to handle complex interactions with relational databases.

Termwork No: 10

Title: Write a program to demonstrate RMI.

Theory: This RMI (Remote Method Invocation) program in Java exemplifies the principles of distributed computing through a "Calculator" application. The architecture involves a remote interface (CalculatorInterface), an implementation class (CalculatorImp), and a client class (CalculatorClient)The remote interface defines the methods that the server provides to remote clients, while the implementation class extends UnicastInterface to offer the actual functionality. The server side orchestrates the registration of the remote object in the RMI registry, making it accessible to remote invocations. On the client side, the CalculatorClient class looks up the remote object in the registry and invokes its methods seamlessly. This illustrative example showcases the simplicity and power of RMI, allowing for the creation of distributed systems where objects residing on different Java Virtual Machines can communicate and collaborate as if they were local. In addition to its simplicity, the RMI example underscores the foundational concepts of distributed systems, including the principles of transparency and encapsulation. RMI abstracts away the complexities of network communication, enabling developers to design applications that leverage remote services with minimal effort. The transparent nature of RMI allows programmers to interact with remote objects in a manner akin to local objects, thereby promoting a high level of abstraction. This abstraction fosters a modular and scalable architecture, facilitating the creation of robust distributed systems. As such, this RMI demonstration not only showcases the technical capabilities of Java in the realm of distributed computing but also emphasizes the importance of abstraction and transparency for building maintainable and extensible distributed applications.