Core Java Exercises

1. Hello World Program

- **Objective:** Understand the basic structure of a Java program.
- **Task:** Write a Java program that prints "Hello, World!" to the console.
- **Instructions:**
    - Create a class named *HelloWorld*.
    - Implement the *main* method.
    - Use *System.out.println()* to display the message.

2. Simple Calculator

- **Objective:** Practice arithmetic operations and user input.
- **Task:** Develop a calculator that performs addition, subtraction, multiplication, and division.
- **Instructions:**
    - Prompt the user to enter two numbers.
    - Ask the user to choose an operation.
    - Display the result of the operation.

3. Even or Odd Checker

- **Objective:** Utilize conditional statements.
- **Task:** Determine if a number entered by the user is even or odd.
- **Instructions:**
    - Prompt the user for an integer.
    - Use the modulus operator *%* to check divisibility by 2.
    - Display whether the number is even or odd.

4. Leap Year Checker

- **Objective:** Apply nested conditional logic.
- **Task:** Check if a given year is a leap year.
- **Instructions:**
    - Prompt the user to enter a year.

- A year is a leap year if it's divisible by 4 but not by 100, unless it's also divisible by 400.
- Display the result accordingly.

5. Multiplication Table

- **Objective:** Implement loops.
- **Task:** Print the multiplication table for a number up to 10.
- **Instructions:**
  - Prompt the user for a number.
  - Use a for *loop* to iterate from 1 to 10.
  - Multiply the input number by the loop counter and display the result.

6. Data Type Demonstration

- **Objective:** Understand Java's primitive data types.
- **Task:** Declare variables of different primitive types and display their values.
- **Instructions:**
  - Declare variables of types *int, float, double, char,* and *boolean*.
  - Assign appropriate values to each.
  - Use *System.out.println()* to display each variable.

7. Type Casting Example

- **Objective:** Practice type casting between different data types.
- **Task:** Convert a *double* to an *int* and vice versa.
- **Instructions:**
  - Declare a *double* variable with a decimal value.
  - Cast it to an *int* and display the result.
  - Declare an *int* variable and cast it to a *double*, then display.

8. Operator Precedence

- **Objective:** Explore how Java evaluates expressions.
- **Task:** Evaluate and display the result of complex expressions.
- **Instructions:**
  - Write expressions combining multiple operators, e.g., *int result = 10 + 5 * 2;*.

- Display the result and explain the order of operations.

9. Grade Calculator

- **Objective:** Use conditional statements to determine grades.
- **Task:** Assign grades based on marks entered by the user.
- **Instructions:**
  - Prompt the user for marks out of 100.
  - Use *if-else* statements to assign grades:
    - 90-100: A
    - 80-89: B
    - 70-79: C
    - 60-69: D
    - Below 60: F
  - Display the assigned grade.

10. Number Guessing Game

- **Objective:** Implement loops and conditional logic.
- **Task:** Create a game where the user guesses a randomly generated number.
- **Instructions:**
  - Generate a random number between 1 and 100.
  - Prompt the user to guess the number.
  - Provide feedback if the guess is too high or too low.
  - Continue until the user guesses correctly.

11. Factorial Calculator

- **Objective:** Use loops to perform repetitive calculations.
- **Task:** Calculate the factorial of a number entered by the user.
- **Instructions:**
  - Prompt the user for a non-negative integer.
  - Use a *for* loop to calculate the factorial.
  - Display the result.

12. Method Overloading

- **Objective:** Understand method overloading in Java.
- **Task:** Create multiple methods with the same name but different parameters.
- **Instructions:**
    - Define methods named *add* that accept:
        - Two integers.
        - Two doubles.
        - Three integers.
    - Each method should return the sum of its parameters.
    - Call each method and display the results.

13. Recursive Fibonacci

- **Objective:** Implement recursion.
- **Task:** Calculate the nth Fibonacci number using recursion.
- **Instructions:**
    - Prompt the user for a positive integer *n*.
    - Define a recursive method *fibonacci(int n)* that returns the nth Fibonacci number.
    - Display the result.

14. Array Sum and Average

- **Objective:** Work with arrays and perform calculations.
- **Task:** Calculate the sum and average of elements in an array.
- **Instructions:**
    - Prompt the user to enter the number of elements.
    - Read the elements into an array.
    - Calculate and display the sum and average.

15. String Reversal

- **Objective:** Manipulate strings.
- **Task:** Reverse a string entered by the user.
- **Instructions:**

- Prompt the user for a string.
- Use a loop or *StringBuilder* to reverse the string.
- Display the reversed string.

16. Palindrome Checker

- **Objective:** Combine string manipulation and conditional logic.
- **Task:** Check if a string is a palindrome.
- **Instructions:**
  - Prompt the user for a string.
  - Remove any non-alphanumeric characters and convert to lowercase.
  - Check if the string reads the same forwards and backwards.
  - Display the result.

17. Class and Object Creation

- **Objective:** Understand classes and objects.
- **Task:** Create a *Car* class with attributes and methods.
- **Instructions:**
  - Define attributes: *make, model, year.*
  - Implement a method *displayDetails()* to print car information.
  - Create objects of the *Car* class and call the method.

18. Inheritance Example

- **Objective:** Implement inheritance.
- **Task:** Create a base class *Animal* and a subclass *Dog*.
- **Instructions:**
  - *Animal* class should have a method *makeSound()*.
  - *Dog* class should override *makeSound()* to print "Bark".
  - Instantiate both classes and call their methods.

19. Interface Implementation

- **Objective:** Use interfaces in Java.
- **Task:** Define an interface *Playable* with a method *play()*.

- **Instructions:**
    - Implement the interface in classes *Guitar* and *Piano*.
    - Each class should provide its own implementation of *play().*
    - Instantiate the classes and call the method.

20. Try-Catch Example

- **Objective:** Handle exceptions gracefully.
- **Task:** Handle division by zero using try-catch.
- **Instructions:**
    - Prompt the user for two integers.
    - Attempt to divide the first by the second.
    - Catch any *ArithmeticException* and display an appropriate message.

21. Custom Exception

- **Objective:** Create and use custom exceptions.
- **Task:** Define a custom exception *InvalidAgeException*.
- **Instructions:**
    - Throw *InvalidAgeException* if the user's age is less than 18.
    - Catch the exception and display a message.

22. File Writing

- **Objective:** Write data to a file.
- **Task:** Write user input to a text file.
- **Instructions:**
    - Prompt the user for a string.
    - Write the string to a file named *output.txt*.
    - Confirm that the data has been written.

23. File Reading

- **Objective:** Read data from a file.
- **Task:** Read and display the contents of output.txt.
- **Instructions:**

- Open *output.txt* for reading.
- Read each line and display it on the console.

24. ArrayList Example

- **Objective:** Use dynamic arrays.
- **Task:** Manage a list of student names.
- **Instructions:**
  - Create an *ArrayList* to store names.
  - Allow the user to add names to the list.
  - Display all names entered.

25. HashMap Example

- **Objective:** Use key-value pairs.
- **Task:** Map student IDs to names.
- **Instructions:**
  - Create a *HashMap* with Integer keys and String values.
  - Allow the user to add entries.
  - Retrieve and display a name based on an entered ID.

26. Thread Creation

- **Objective:** Implement multithreading.
- **Task:** Create and run two threads that print messages.
- **Instructions:**
  - Define a class that extends *Thread* or implements *Runnable*.
  - In the *run()* method, print a message multiple times.
  - Start both threads and observe the output.

27. Lambda Expressions

- **Objective:** Use functional programming features.
- **Task:** Sort a list of strings using a lambda expression.
- **Instructions:**
  - Create a *List* of strings.

- Use *Collections.sort()* with a lambda to sort the list.
- Display the sorted list.

28. Stream API

- **Objective:** Process collections using streams.
- **Task:** Filter and display even numbers from a list.
- **Instructions:**
  - Create a *List* of integers.
  - Use the Stream API to filter even numbers.
  - Collect and display the result.

29. Records

- **Objective:** Use the *record* keyword for immutable data structures (Java 16+).
- **Task:** Create a record to represent a *Person* with *name* and *age*.
- **Instructions:**
  - Define a record named *Person*.
  - Create instances and print them.
  - Use records in a *List* and filter based on age using Streams.

30. Pattern Matching for *switch* (Java 21)

- **Objective:** Simplify conditional logic with pattern matching in enhanced switch expressions.
- **Task:** Determine the type of an object and respond accordingly.
- **Instructions:**
  - Create a method that accepts *Object* as input.
  - Use a *switch* expression to check if the object is *Integer, String, Double*, etc.
  - Print a message based on the object's type.

31. Basic JDBC Connection

- **Objective:** Connect Java with a relational database.
- **Task:** Connect to a local MySQL/SQLite database and retrieve data.
- **Instructions:**

- Set up a database with a *students* table.
- Write code to load the JDBC driver, create a connection, execute a *SELECT* query, and print results.

32. Insert and Update Operations in JDBC

- **Objective:** Perform insert/update SQL queries from Java.
- **Task:** Add and modify student data using JDBC.
- **Instructions:**
  - Create a *StudentDAO* class.
  - Implement methods to insert new records and update student details.
  - Use *PreparedStatement* for parameterized queries.

33. Transaction Handling in JDBC

- **Objective:** Use JDBC transactions.
- **Task:** Simulate a money transfer between two accounts.
- **Instructions:**
  - Create *accounts* table with balances.
  - Implement a transfer method with *Connection.setAutoCommit(false)*.
  - Commit if both debit and credit succeed, else rollback.

34. Create and Use Java Modules

- **Objective:** Understand Java's module system.
- **Task:** Create two modules: *com.greetings* and *com.utils*.
- **Instructions:**
  - Define a module-info.java file in both modules.
  - Export a utility class from *com.utils* and use it in *com.greetings*.
  - Compile and run using the module path.

35. TCP Client-Server Chat

- **Objective:** Use Java sockets for TCP communication.
- **Task:** Implement a simple TCP chat system.
- **Instructions:**
  - Create a *ServerSocket* that listens for connections.

- Accept client connections and use *InputStream* and *OutputStream* for two-way communication.
- Run server and client in different terminals.

36. HTTP Client API (Java 11+)

- **Objective:** Make HTTP requests from Java.
- **Task:** Fetch data from a public API (e.g., GitHub).
- **Instructions:**
  - Use *HttpClient* and *HttpRequest*.
  - Print the response status and body.
  - Optional: Parse JSON response using Jackson or Gson.

37. Using *javap* to Inspect Bytecode

- **Objective:** Explore compiled *.class* files.
- **Task:** Compile a Java class and inspect its bytecode using *javap*.
- **Instructions:**
  - Create a class with a method.
  - Compile it and run *javap -c ClassName*.
  - Interpret the bytecode output.

38. Decompile a Class File

- **Objective:** Reverse engineer compiled Java bytecode.
- **Task:** Use a tool like *JD-GUI* or *CFR* to decompile a *.class* file.
- **Instructions:**
  - Write a simple Java program and compile it.
  - Open the *.class* file in a decompiler.
  - Analyze the decompiled source.

39. Reflection in Java

- **Objective:** Use Java Reflection API.
- **Task:** Load a class and invoke methods dynamically.
- **Instructions:**

- Use *Class.forName()*, *getDeclaredMethods()*, and *invoke()* to call a method without directly referencing it in code.
- Print the method names and parameters.

40. Virtual Threads (Java 21)

- **Objective:** Use lightweight threads for scalable concurrency.
- **Task:** Launch 100,000 virtual threads that each print a message.
- **Instructions:**
  - Use *Thread.startVirtualThread(() -> { ... })*.
  - Measure performance versus traditional threads.

41. Executor Service and Callable

- **Objective:** Use concurrency utilities.
- **Task:** Execute multiple *Callable* tasks that return results.
- **Instructions:**
  - Use *Executors.newFixedThreadPool()* and *submit()* to execute callables.
  - Collect results using *Future.get()*.