Wrapper classes

1. Check if character is a Digit
2. Compare two Strings
3. Convert using valueof method
4. Create Boolean Wrapper usage
5. Convert null to wrapper classes

Pass by value and pass by reference

1. Write a program where a method accepts an integer parameter and tries to change its value. Print the value before and after the method call.
2. Create a method that takes two integer values and swaps them. Show that the original values remain unchanged after the method call.
3. Write a Java program to pass primitive data types to a method and observe whether changes inside the method affect the original variables.

**Call by Reference (Using Objects)**

1. Create a class Box with a variable length. Write a method that modifies the value of length by passing the Box object. Show that the original object is modified.
2. Write a Java program to pass an object to a method and modify its internal fields. Verify that the changes reflect outside the method.
3. Create a class Student with name and marks. Write a method to update the marks of a student. Demonstrate the changes in the original object.
4. Create a program to show that Java is strictly "call by value" even when passing objects (object references are passed by value).
5. Write a program where you assign a new object to a reference passed into a method. Show that the original reference does not change.
6. Explain the difference between passing primitive and non-primitive types to methods in Java with examples.
7. Can you simulate call by reference in Java using a wrapper class or array? Justify with a program.

MultiThreading

1 Write a program to create a thread by extending the Thread class and print numbers from 1 to 5.

2 Create a thread by implementing the Runnable interface that prints the current thread name.

3 Write a program to create two threads, each printing a different message 5 times.

4 Demonstrate the use of Thread.sleep() by pausing execution between numbers from 1 to 3.

5 Create a thread and use Thread.yield() to pause and give chance to another thread.

6 Implement a program where two threads print even and odd numbers respectively.

7 Create a program that starts three threads and sets different priorities for them.

8 Write a program to demonstrate Thread.join() – wait for a thread to finish before proceeding.

9 Show how to stop a thread using a boolean flag.

10 Create a program with multiple threads that access a shared counter without synchronization. Show the race condition.

11 Solve the above problem using synchronized keyword to prevent race condition.

12 Write a Java program using synchronized block to ensure mutual exclusion.

13 Implement a BankAccount class accessed by multiple threads to deposit and withdraw money. Use synchronization.

14 Create a Producer-Consumer problem using wait() and notify().

15 Create a program where one thread prints A-Z and another prints 1-26 alternately.

16 Write a program that demonstrates inter-thread communication using wait() and notifyAll().

17 Create a daemon thread that runs in background and prints time every second.

18 Demonstrate the use of Thread.isAlive() to check thread status.

19 Write a program to demonstrate thread group creation and management.

20 Create a thread that performs a simple task (like multiplication) and returns result using Callable and Future.