Multithreading in Java

Task 1: Create a Thread by Extending the Thread Class

Description:
Write a class called `MyPrinterThread` that prints the message `Hello from Thread!` 5 times with a delay of 1 second between each message.

Expected Output:

```
Hello from Thread!
... (repeated 5 times, 1 second apart)
```

Task 2: Create a Thread by Implementing Runnable

Description:
Create a class `RunnableCounter` that implements `Runnable`. Print numbers from 1 to 10 using a `for` loop in the `run()` method.

Task 3: Thread with Sleep and Join

Description:
Create two threads:

- Thread1 should print numbers 1 to 5 with a 1-second delay.
- Main thread should wait for Thread1 to finish using `join()` before printing "Main thread done."

Task 4: Check Thread State using isAlive()

Description:

- Start a thread and check whether it is alive before and after `start()` and `join()` calls.

Expected Output:

```
Before start: false
After start: true
After join: false
```

Task 5: Thread Priority Demo

Description:
Create three threads and set their priorities to `MIN_PRIORITY`, `NORM_PRIORITY`, and `MAX_PRIORITY`. Print the priority of each thread and observe the execution order.

```
t1.setPriority(Thread.MIN_PRIORITY);  // 1
t2.setPriority(Thread.NORM_PRIORITY); // 5
t3.setPriority(Thread.MAX_PRIORITY);  // 10
```

Synchronization Tasks

Task 6: Shared Counter Without Synchronization

Description:
Create a shared counter variable. Create two threads that increment this counter 1000 times each without using synchronization.

Expected Result:
Final value may be **less than 2000** due to race condition.

Task 7: Fix the Shared Counter with Synchronized Block

Description:
Modify Task 6 using a synchronized block or method to ensure data consistency.

Expected Result:
Final count should always be **2000**.

Task 8: Bank Account Withdrawal Simulation

Description:
Create a class `BankAccount` with a `balance` variable and a method `withdraw(int amount)` which should be synchronized.
Start two threads simulating two users withdrawing money from the same account.

Goal:
Avoid overdrawing the balance. Use `synchronized` to ensure thread safety.

Inter-thread Communication Tasks

Task 9: Producer-Consumer (1 Item Buffer)

Description:
Create a producer thread that produces numbers 1 to 5 and a consumer thread that consumes them. Use `wait()` and `notify()` for communication.

Constraints:

- Only one item can be stored at a time.
- The producer should wait if the buffer is full.
- The consumer should wait if the buffer is empty.

Task 10: Chat Simulation

Description:
Simulate a chat system where one thread sends messages and another thread receives them using `wait()` and `notify()`.

Sample Output:

```
Sender: Hi!
Receiver: Hello!
Sender: How are you?
Receiver: I’m fine.
```

Use synchronized methods to ensure communication alternates between send and receive.

Task 11: Thread Race Simulation

Description:
Create 3 threads named `Tortoise`, `Rabbit`, and `Dog`. Each thread prints its name and "running..." in a loop of 5 times, with a random sleep time.

Observe how threads interleave.

Task 12: Countdown Timer using Threads

Description:
Create a thread that counts down from 10 to 1, pausing 1 second between numbers, and prints "Time's up!" at the end.