Lab – Java – Threads and Synchronization (Easy)

Overview

Write an application that uses threads and synchronization. There is also an exercise to demonstrate deadlock and deadlock prevention.

Create Console Application

Create a new Java console application.

Create a Class that Implements Runnable

Create a class named Test that implements Runnable interface. The run method should just print the name of the current thread. You can get the name of the thread using the following code:

Thread.currentThread().getName()

Main

- At the start of main, print the name of the current thread.
- Create an instance of Test and then call run on that instance.

What is the output?

Main (run test on another thread)

Add code to have Test's run method run on another thread.

What is the output?

Create a Class Derived from Thread

- Create a class named Test2 that is derived from Thread. Override the run method in Test2.
- Add code to main that will execute Test2's run method on another thread.

There should now be three print statements in the output. Are the thread names the same or different?

Create an Anonymous Class that Implements Runnable

In main create an anonymous class that implements Runnable. Add code to main that will execute the run method of the anonymous class.

Run Same Method on Different Threads (No Synchronization)

- Create a static method named showNumbers in the Main class that prints the numbers 1-10 using a loop. Show the thread name of the number that is being printed.
- In main, create 3 threads that run the static method.

What does the output look like? The output will likely show each thread's output grouped together (1-10 followed by 1-10 followed by 1-10).

Run Same Method on Different Threads (No Synchronization/Sleep)

- Add a line to the loop that makes the thread sleep for 500 milliseconds.

What does the output look like? The output will now show each threads output mixed together.

Run Same Method on Different Threads (Synchronization/Sleep)

- Add synchronization code to the showNumbers method that will only allow one thread to print numbers at a time. The synchronization code should encompass the loop. Hint: You can decorate the method header with a keyword.

What does the output look like? The output will show each threads output grouped together.

Show Done Messages in all Threads

- Add a print statement to the end of main that displays the current thread name concatenated with the word done.
- Add a print statement to the end of showNumbers that displays the current thread name concatenated with the word done.

Does the main thread's message appear before or after each thread's done message? The main thread's message will most likely appear first because it does not wait for the other threads to finish.

Make the Main Thread Wait for the Other Threads

- Add to main just before its done print statement that will make it wait for the other threads to finish.

Does the main thread's message appear before or after each thread's done message? It will now appear after each of the other thread's done messages.

Main (deadlock)

Write code that will create a deadlock.

- **Create two static locks in the Main class (lock 1 and lock 2).**
- Create a static method in the Main class with the following header: **static void myMethod()**. It should do the following:
    - Obtain lock 1 (use a synchronized block to obtain a lock).
    - Print a message that states “myMethod obtained lock 1”.
    - Add a sleep statement for 500 milliseconds (right after obtaining lock 1).
    - Obtain lock 2 (after the sleep statement).
    - Print a message that states “myMethod obtained lock 2”
    - Print a message after the lock 2 block that states that it released lock 2.
    - Print a message after the lock 1 block that states that it released lock 1.
- Create a static method in the Main class with the following header: **static void otherMethod()**. It should do the following:
    - Obtain lock 1.
    - Print a message that states “otherMethod obtained lock 2”.
    - Add a sleep statement for 500 milliseconds (right after obtaining lock 1).
    - Obtain lock 2 (after the sleep statement).
    - Print a message that states “otherMethod obtained lock 1”
    - Print a message after the lock 1 block that states that it released lock 2.
    - Print a message after the lock 2 block that states that it released lock 1.
- In main, create and start a thread that runs myMethod.
- In main, create and start a thread that runs otherMethod.

When the program executes it should hang. This is deadlock. It should show only show messages indicating that each method has acquired only one of the locks. You will need to stop the program.

Main (deadlock solution)

Update the code such that deadlock is avoided (you only need to update one of the methods to do this). When the program runs it should now finish. It should show messages that both threads acquired and released both locks.