Short Answer:

Answer the following questions with complete sentences in your own words. You are encouraged to conduct your own research online or through other methods before answering the questions. If you research online, please consult multiple sources before you write down your answers. You are expected to be able to explain your answers in detail (Provide examples to each question).

1. What is Thread and What is Process? What are the differences between them?
2. How to create threads in Java?
3. Runnable or Thread, which do you prefer to use? Why?
4. What are the differences between start() and run()?
5. What happens if start() is invoked on a thread twice? What if run() is invoked on a thread twice?
6. What is the Thread Life Cycle in Java? Explain how to get to each stage.
7. Explain the join() method in Java Thread.
8. What are the wait(), sleep(), yield() methods? What are the differences between them?
9. What is notify()?
10. What is Daemon thread in Java? Why do we need it?
11. What is thread interference? Give an example.
12. What is memory consistency error? Give an example.
13. What are the ways of Thread Synchronization?
14. What is Deadlock? How to resolve it?

Coding Questions:

Write code in Java to solve following problems. Please write your own answers. You are highly encouraged to present more than one way to answer the questions. Please follow best practice when you write the code so that it would be easily readable, maintainable, and efficient. Clearly state your assumptions if you have any. You may discuss with others on the questions, but please write your own code.

1. Create the main() method to reproduce the Counter Thread interference issue, run several times and explain the result.

2. Use synchronized method to make the Counter work as expected.

3. Write a Java Program that implements a multithread application that has three threads. The first thread generates a random integer every second and if the number is even the second thread prints the square of the number. If the number is odd, the third thread will print the cube of the number.

4. How to solve the following deadlock problem.

```
public class DeadlockExample {

        public static void main(String[] args) {
```

```
    // TODO Auto-generated method stub

    Key key1 = new Key();
    Key key2 = new Key();

    Thread fan = new Thread(()->{
      synchronized(key1) {
        key1.setAvailable(false);
                                    System.out.println("fan has key1");

        try {
          Thread.sleep(1000);         } catch (InterruptedException e) {           //
              TODO Auto-generated catch block e.printStackTrace();
        }

        synchronized(key2) {
          key2.setAvailable(false); System.out.println("fan has key2");
        }
      }
    });


    Thread landon = new Thread(()->{
      synchronized(key2) {
        key2.setAvailable(false);
                                    System.out.println("landon has key2");
                                    synchronized(key1) {
          key1.setAvailable(false); System.out.println("landon has key1");
        }
      }
    });

    fan.start();
    landon.start();

          }

}

 class Key{
          private boolean available;
          Key(){
    this.available = true;
          }

          public void setAvailable(boolean available) {
    this.available = available;
          }
```

}

5. Write a java code for the producer consumer problem.
The problem describes two processes, the producer and the consumer, who share a common, fixed-size buffer used as a queue. The producer's job is to generate data, put it into the buffer, and start again. At the same time, the consumer is consuming the data (i.e., removing it from the buffer), one piece at a time. The problem is to make sure that the producer won't try to add data into the buffer if it's full and that the consumer won't try to remove data from an empty buffer. (Do not copy & paste code from online! It will be very helpful for you to understand synchronization in multi-threading env)
• The producer will produce random integers between 1 - 100 and the max size of the queue is 10. Once the Queue is full, the producer stops producing random numbers and waits for the consumer to consume all the numbers in the queue.
• The consumer will take one integer from the queue at a time, print it with the consumer name, until all the numbers are consumed.
- Try to implement with only one producer and one consumer
- Try to implement with one producer and two consumer