ITE 2005-ADVANCED JAVA PROGRAMMING

ASSESSMENT NO. 02

DEADLINE: 15-02-2022

1. Implement the producer consumer problem for the bounded buffer of size ‘n’.

2. Lets assume the case of 2 threads (denoted A and B) that communicate using a circular array (a basic one-dimensional array "int[]") . More precisely, thread A generates and writes data into this array, and thread B reads these data from the array. Reading and the updating of variables for holding the start value and quantity are performed in synchronized blocks on the same object.

   If the shared array is full, then thread A waits until thread B finish reading data from it. If the array is empty, then thread B waits until thread A finish writing some data into it. Since the array can't be full and empty at the same time, it is impossible for both threads to be waiting at the same time.

   Also, since thread A can only write in the free part of the array, and thread B can only read from the occupied part of the array, then the threads will never access the same cell at the same time, i.e. the threads are working on 2 independent parts of the array. Implement the above scenario and display the output at each sequence.

3. Write a method that takes a string and returns the number of unique characters in the string. It is expected that a string with the same character sequence may be passed several times to the method. Since the counting operation can be time consuming, the method should cache the results, so that when the method is given a string previously encountered, it will simply retrieve the stored result. Use collections and maps where appropriate.
4. Make a Map that associates the following employee IDs with names. Keys and values of Maps can be any Object type, so in real life you would probably have the key be a String and the associated value be a Person or Employee object. To make things simpler on this exercise, you can use String for both the ID and the name, rather than bothering to create a Person or Employee class. The point here is to associate keys with values, then retrieve values later based on keys.

ID	Name
a1234	Steve Jobs
a1235	Scott McNealy
a1236	Jeff Bezos
a1237	Larry Ellison
a1238	Bill Gates

   Make test cases where you test several valid and invalid ID’s and print the associated name.

5. Make a coin-flipping class that implements Runnable. The run method should flip 1000 coins and print out whenever it sees 3 or more consecutive heads. Make a task queue, and put 5 separate instances of the Runnable class in the queue. In the printouts, you can use Thread.currentThread().getName() to identify the thread. You are following variation 1 of the basic threading approach (separate classes that implement Runnable), so your code will look something like this (or, you could call execute from a loop):

```
public class Foo implements Runnable {
  public void run() { loop, flip coins, check for 3+ heads in a row }
}
-----------------------------------------------------
public class Driver {
  public static void main(String[] args) {
    ExecutorService tasks = ...
    tasks.execute(new Foo()); // Multiple instances of Foo
    tasks.execute(new Foo());
    tasks.execute(new Foo());
  }
}
```

6. Make an "infinite" stream that generates random doubles between 0 and 10. Use it to
   • Print 5 random doubles
   • Make a List of 10 random doubles
   • Make an array of 20 random doubles
   Note: in general, if you are dealing with numbers, DoubleStream is preferred over Stream because DoubleStream uses primitives and has more convenient methods (e.g., min, max, sum, average). In this case, however, use Stream because it is hard to turn a DoubleStream into a List and because it is hard to print a double[] but easy to print a Double[] (e.g., pass the array to Arrays.asList and print the resultant List). So, for this part of the exercises, use Stream.generate, not DoubleStream.generate