1. (20 points) Answer each of questions in 1-2 sentences

(a) (5 points) What is the Java Monitor Pattern, and what is it used for?

(b) (5 points) Why are Java programmers generally advised to use `notifyAll()` rather than `notify()` in their programs?

(c) (5 points) Explain the difference between *balking* and *guarded suspension* approaches to implementing state-dependent actions.

(d) (5 points) What is “lock striping”? 
2. (20 points)

(a) (5 points) What is “nested monitor lockout”?

(b) (7 points) Consider the following implementation of a thread-safe class of unbounded buffer of strings.

```java
public class StringBuffer {
    private final ArrayList<String> strings;
    StringBuffer () { strings = new ArrayList<String>(); }

    public synchronized void put (String newString) {
        strings.add(newString);
        notifyAll();
    }

    public synchronized String take() throws InterruptedException {
        while (strings.size() == 0) { wait(); }
        String returnString = strings.get(0);
        strings.remove(0);
        return returnString;
    }
}
```

Now suppose we wish to implement a thread-safe string buffer that does not insert empty strings as follows.

```java
public class StringBufferNonEmpty {
    private final StringBuffer buffer;
    StringBufferNonEmpty() { buffer = new StringBuffer(); }

    public synchronized void put (String newString) {
        if (!newString.equals("")) buffer.put(newString);
    }

    public synchronized String take () throws InterruptedException {
        return buffer.take();
    }
}
```
Give a situation in which nested monitor lockout can occur with this implementation of `StringBufferNonEmpty`.

(c) (8 points) How can the implementation of `StringBufferNonEmpty` be fixed so that nested monitor lockout cannot occur? (You may also alter the implementation of `StringBuffer` if you wish.)
3. (20 points)
(a) (6 points) Suppose field list is initialized as follows.

```java
List<Object> list = Collections.synchronizedList(new ArrayList<Object>());
```

Now consider the following method implemented in a separate class

```java
public static void putIfAbsent (List<Object> l, Object o) {
    if (!l.contains(o)) l.add(o);
}
```

If list is initially empty, and two different threads invoke `putIfAbsent(list, o)` with the same object `o`, will list be guaranteed to contain only one copy of `o` after both terminate? Explain.

(b) (7 points) Consider the variable list defined above, and assume that several threads are accessing it. Suppose one thread executes the following.

```java
System.out.println(list);
```

Explain why the exception `ConcurrentModificationException` may be thrown by this statement.
(c) (7 points) Give a fix to the statement in the previous part that eliminates the possibility of the given exception being thrown.
4. (20 points)

(a) (5 points) Explain how bounded blocking queues may be used in Producer-Consumer applications to “slow down” producers to match the rate at which consumers process elements from the queue.

(b) (5 points) What mechanism do CopyOnWriteArrayList objects use to ensure that ConcurrentModificationExceptions can never be thrown?

(c) (5 points) Explain the difference between a CountdownLatch object and a Semaphore object.

(d) (5 points) What is “thread-starvation deadlock?”
5. (20 points) Linear search is an algorithm for locating the first position in an array at which a given element occurs. Linear search algorithms return this position, if the element occurs in the array, or -1 if the element is not in the array.

Implement a parallel linear-search algorithm for locating an integer in an array of integers. Your method should have the following header.

    public static int find (int[] elts, int elt);

Your solution must use an Executor of some sort, although the type you use is up to you. You should also make an attempt to “tune” your solution so that it efficiently exploits parallelism. For this purpose, you may find the method call Runtime.getRuntime().availableProcessors() useful.