```
0 new #2 <Class java.lang.Object>
                        //Create an object of type java.lang.Object
                        //and push a reference to it(lock).
3 dup                   //Duplicate the top stack value and push it.
4 invokespecial #3 <Method java.lang.Object()>
                        //Pop the object reference(lock), and invoke
                        //its constructor.
7 astore_1              //Pop the object reference(lock) and store
                        //it at index 1 of the local variable table.
```

Creating a zero element array does not require a constructor call like the creation of `Object` does. It therefore executes faster. In addition, `byte` arrays that contain elements are often represented more compactly in the JVM than `int` arrays.

Either option results in code that is thread safe. Remember that synchronizing on an instance method or object reference obtains a completely different lock than code that synchronizes on a static method or class literal. Simply because two methods are declared `synchronized` does not necessarily mean they are thread safe. You must be careful to recognize and distinguish between the different locks obtained with synchronization.

PRAXIS 48: Use `private` data with an accessor method instead of `public` or `protected` data

The purpose of writing code with `synchronized` methods is to protect data from corruption. To properly protect data, you must ensure that it is declared and accessed correctly. Failure to correctly protect data allows users of your class to bypass whatever synchronization mechanisms you have in place.

For example, consider the following class that contains two methods that operate on an array. The array is declared as instance data of the class. Both methods are declared `synchronized` to ensure that data is not added and subtracted from the array concurrently. Is this class thread safe?

```
class Test implements Runnable
{
  public int[] intArray = new int[10];

  public synchronized void addToArray(int[] ar)
  {
    int len = intArray.length;
    if (len == ar.length)
    {
```