Multithreading.fm Page 184 Sunday, July 16, 2000 8:37 PM


```
class SomeClass implements Runnable
{
  public int sumArrays(ArrayWithLockOrder a1,
                       ArrayWithLockOrder a2)
  {
    int value = 0;
    ArrayWithLockOrder first = a1;  //Keep a local copy of array
    ArrayWithLockOrder last = a2;   //references.
    int size = a1.array().length;
    if (size == a2.array().length)
    {
      if (a1.lockOrder() > a2.lockOrder()) //Determine and set the
      {                                    //lock order of the
        first = a2;                        //objects.
        last = a1;
      }
      synchronized(first) {  //Lock the objects in correct order.
        synchronized(last) {
          int[] arr1 = a1.array();
          int[] arr2 = a2.array();
          for (int i=0; i<size; i++)
            value += arr1[i] + arr2[i];
        }
      }
    }
    return value;
  }
  public void run() {
    //...
  }
}
```

The `ArrayWithLockOrder` class is provided as a wrapper to the arrays used in the first example. This class increments the `static` `num_locks` variable each time a new object of the class is created. A separate `lock_order` instance variable is set to the current value of the `num_locks` `static` variable. This ensures that each object of this class has a unique value for the `lock_order` variable. The `lock_order` instance variable serves as the indicator for the order that this object should be locked in relation to other objects of this class.

Note that the manipulation of the `static` `num_locks` variable is done from within a `synchronized` statement. This is required because each instance of an object shares its `static` variables. Therefore, if two threads create an object of the `ArrayWithLockOrder` class concurrently, the `static` `num_locks` variable could be corrupted if the code manipulating it is not synchronized. Synchronizing this code ensures that each object of the `ArrayWithLockOrder` class has a unique value for its `lock_order` variable.