**Esercizio 1**

Modifica a classe Reader:

while (S5Esercizio1.*isRunning*.get()) {  
 // MANY Reader  
 int lastValue = S5Esercizio1.*sharedValue*;

Modifica a classe S5Esercizio1:

//final static ReentrantLock lock = new ReentrantLock();  
static volatile int *sharedValue* = 0;

for (int i = 0; i < 1000; i++) {  
 // ONE Writer  
 S5Esercizio1.*sharedValue* = random.nextInt(10);

In questo esempio i lock sono superflui visto che ho un unico writer e tanti reader. Dichiarando la variabile sharedValue come volatile ho inoltre risolto i possibili problemi di visibilita’.

**Esercizio 2**

Modifica a classe EventListener. Il costruttore diventa privato e aggiungo un metodo factory:

public static EventListener build(final int id, final EventSource eventSource) {  
 EventListener listener = new EventListener(id);  
 // Aggiunge listener alla eventSource per ricevere le notifiche  
 eventSource.registerListener(id, listener);  
 return listener;  
}  
  
private EventListener(final int id) {  
 try {  
 Thread.*sleep*(4);  
 } catch (final InterruptedException e) { }  
 this.id = id;  
}

Modifica a classe S5Esercizio2

for (int i = 1; i <= 20; i++) {  
 allListeners.add(EventListener.*build*(i, eventSource));  
}

**Esercizio 3**

Nuova classe ImmutableHolderSharedState e ImmutableValue

final class ImmutableValue {  
 private final int value;  
  
 ImmutableValue(int value) {  
 this.value = value;  
 }  
  
 int getValue() {  
 return value;  
 }  
  
 ImmutableValue increment() {  
 return new ImmutableValue(value+1);  
 }  
}  
  
final class ImmutableHolderSharedState implements IState {  
 private final AtomicReference<ImmutableValue> holderRef = new AtomicReference<>();  
  
 ImmutableHolderSharedState() {  
 holderRef.set(new ImmutableValue(0));  
 }  
  
 @Override  
 public void increment() {  
 while(true) {  
 ImmutableValue oldVal = holderRef.get();  
 if(holderRef.compareAndSet(oldVal, oldVal.increment()))  
 break;  
 }  
 }  
  
 @Override  
 public int getValue() {  
 return holderRef.get().getValue();  
 }  
}

Modica a classe Helper:

for (int i = 0; i < 5000; i++) {  
 S5Esercizio3.*incrementSharedValue*();

Modifica a classe Starter

for (int i = 0; i < 5000; i++) {  
 S5Esercizio3.*incrementSharedValue*();

Modifica a classe S5Esercizio3:

static final Object *lockObject* = new Object();  
volatile static IState *sharedState* = null;

static void incrementSharedValue() {  
 if (*THREADSAFE\_SHARE*)  
 *sharedState*.increment();  
 else {  
 synchronized (*lockObject*) {  
 *sharedState*.increment();  
 }  
 }  
}  
  
static int getSharedValue() {  
 int curValue;  
 if (S5Esercizio3.*THREADSAFE\_SHARE*)  
 curValue = S5Esercizio3.*sharedState*.getValue();  
 else {  
 synchronized (S5Esercizio3.*lockObject*) {  
 curValue = S5Esercizio3.*sharedState*.getValue();  
 }  
 }  
 return curValue;  
}