§66 Синхронизированный доступ к разделяемым mutable данным

- JLS гарантирует, что чтение-запись переменных примитивных типов кроме long, double – атомарно
- Синхронизация необходима для надежной коммуникации между потоками, и для взаимного исключения
- Атомарно – не значит синхронизированно

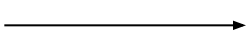
```
// Broken! - How long would you expect this program to run?
public class StopThread {
    private static boolean stopRequested;

    public static void main(String[] args)
            throws InterruptedException {
        Thread backgroundThread = new Thread(new Runnable() {
            public void run() {
                int i = 0;
                while (!stopRequested)
                    i++;
            }
        });
        backgroundThread.start();

        TimeUnit.SECONDS.sleep(1);
        stopRequested = true;
    }
}
```

- Hoisting optimization (для не синхронизированных переменных):

```
while (!done)
    i++;
```



```
if (!done)
    while (true)
        i++;
```

```
// Properly synchronized cooperative thread termination
public class StopThread {
    private static boolean stopRequested;
    private static synchronized void requestStop() {
        stopRequested = true;
    }
    private static synchronized boolean stopRequested() {
        return stopRequested;
    }

    public static void main(String[] args)
            throws InterruptedException {
        Thread backgroundThread = new Thread(new Runnable() {
            public void run() {
                int i = 0;
                while (!stopRequested())
                    i++;
            }
        });
        backgroundThread.start();

        TimeUnit.SECONDS.sleep(1);
        requestStop();
    }
}
```

- Недостаточно синхронизировать только чтение или только запись

```
// Cooperative thread termination with a volatile field
public class StopThread {
    private static volatile boolean stopRequested;

    public static void main(String[] args)
            throws InterruptedException {
        Thread backgroundThread = new Thread(new Runnable() {
            public void run() {
                int i = 0;
                while (!stopRequested)
                    i++;
            }
        });
        backgroundThread.start();

        TimeUnit.SECONDS.sleep(1);
        stopRequested = true;
    }
}
```

- Менее громоздкое исправление

```
// Broken - requires synchronization!
private static volatile int nextSerialNumber = 0;

public static int generateSerialNumber() {
    return nextSerialNumber++;
}
```

- ++ - не атомарный оператор. На самом деле это ДВЕ операции: прочитать значение, записать значение + 1
- Можно исправить это добавив syncronized и убрав volatile
- Но лучше так:

```
private static final AtomicLong nextSerialNum = new AtomicLong();

public static long generateSerialNumber() {
    return nextSerialNum.getAndIncrement();
}
```

- Лучший способ избежать подобных проблем: не использовать общие не mutable данные
- Ограничивайте использование mutable данных внутри одного потока
- Если все же вы используете общие mutable данные каждый поток должен заботиться о синхронизации чтения/записи.