# Volatile Keyword

CS 272 Software Development

**Professor Sophie Engle**
Department of Computer Science

# Synchronized Keyword

- Protects blocks of code, not objects

- Provides **mutual exclusion**, which causes **blocking**, which slows down code

- Can be used to prevent **thread interference** (atomicity) and prevent **memory consistency** errors (visibility)

http://docs.oracle.com/javase/tutorial/essential/concurrency/sync.html

**CS 272 Software Development**
Professor Sophie Engle

**Department of Computer Science**
https://www.cs.usfca.edu/

UNIVERSITY OF
SAN FRANCISCO

# Volatile Keyword

- Indicates a variable is **unstable** (i.e. volatile) and may be accessed concurrently

- Provides *lightweight* synchronization
  - Changes are always visible to other threads
  - Does not causes blocking

- Does <u>not</u> eliminate need for other synchronization!

http://docs.oracle.com/javase/tutorial/essential/concurrency/atomic.html

**CS 272 Software Development**
Professor Sophie Engle

**Department of Computer Science**
https://www.cs.usfca.edu/

UNIVERSITY OF
SAN FRANCISCO

# Volatile Keyword

- Threads always read latest value (not cached value)

- Write operations cannot depend on current value
  - e.g. `shutdown = true;`

- Read operations cannot be used with other variables
  - ~~e.g. `if (volatileVar < otherVar)`~~
  - e.g. `if (volatileVar == true)`

Source: **Java Theory and Practice: Managing Volatility** *by Brian Goetz*

**CS 272 Software Development**
Professor Sophie Engle

**Department of Computer Science**
https://www.cs.usfca.edu/

UNIVERSITY OF
SAN FRANCISCO

# Proper Use Patterns

## Java Theory and Practice: Managing Volatility

*Brian Goetz for IBM Developer*

https://www.ibm.com/developerworks/java/library/j-jtp06197/

Archived:

https://web.archive.org/web/20210228140403/https://www.ibm.com/developerworks/java/library/j-jtp06197/index.html

**CS 272 Software Development**
Professor Sophie Engle

**Department of Computer Science**
https://www.cs.usfca.edu/

UNIVERSITY OF
SAN FRANCISCO

# Proper Use Patterns

- Pattern #1: Status flags
    - Write of flag does not depend on current value
    - Read of flag does not depend on other variables

- Pattern #2: One-Time Safe Publication
    - Object must be thread-safe or effectively immutable
    - Object must be initialized only once

Source: **Java Theory and Practice: Managing Volatility** *by Brian Goetz*

**CS 272 Software Development**
Professor Sophie Engle

**Department of Computer Science**
https://www.cs.usfca.edu/

UNIVERSITY OF
SAN FRANCISCO

```
1.  private volatile boolean active;
2.
3.  public void shutdown() {
4.      active = false;
5.  }
6.
7.  public void run() {
8.      while (active) {
9.          // do stuff…
10.     }
11. }
```

Source: **Java Theory and Practice: Managing Volatility** *by Brian Goetz*

**CS 272 Software Development**
Professor Sophie Engle

**Department of Computer Science**
https://www.cs.usfca.edu/

UNIVERSITY OF
SAN FRANCISCO

```
1. public class WidgetLoader extends Thread {
2.     public volatile Widget widget;
3.     public void run() {
4.         widget = loadWidget();
5.     }
6. }
7.
8. public class MainThread extends Thread {
9.     public void run() {
10.         while (true) {
11.             if (widgetLoader.widget ≠ null) {
12.                 // do stuff ...
```

Source: **Java Theory and Practice: Managing Volatility** *by Brian Goetz*

**CS 272 Software Development**
Professor Sophie Engle

**Department of Computer Science**
https://www.cs.usfca.edu/

UNIVERSITY OF
SAN FRANCISCO

# Proper Use Patterns

- Pattern #3: Independent Observations
  - Similar to one-time safe publication, except multiple independent writes of effectively immutable object

- Pattern #5: Cheap Read-Write Lock
  - Use **volatile** for non-blocking reads
  - Use **synchronized** for blocking writes

Source: **Java Theory and Practice: Managing Volatility** *by Brian Goetz*

**CS 272 Software Development**
Professor Sophie Engle

**Department of Computer Science**
https://www.cs.usfca.edu/

UNIVERSITY OF
SAN FRANCISCO

```java
 1. private volatile String lastUser;
 2.
 3. public void auth(String user, String pass) {
 4.     boolean valid = checkPass(user, pass);
 5.     if (valid) {
 6.         activeUsers.add(user);
 7.         lastUser = user;
 8.     }
 9.     return valid;
10.   }
11. }
```

Source: **Java Theory and Practice: Managing Volatility** *by Brian Goetz*

**CS 272 Software Development**
Professor Sophie Engle

**Department of Computer Science**
https://www.cs.usfca.edu/

UNIVERSITY OF
SAN FRANCISCO

```
1. private volatile int counter;
2.
3. public int getCount() {
4.     return counter;
5. }
6.
7. public synchronized void increment() {
8.     counter++;
9. }
```

Source: **Java Theory and Practice: Managing Volatility** *by Brian Goetz*

**CS 272 Software Development**
Professor Sophie Engle

**Department of Computer Science**
https://www.cs.usfca.edu/

UNIVERSITY OF
SAN FRANCISCO

# Proper Use Patterns

- Use **carefully**, or not at all
  - This class does not require the use of this keyword

- Use for **simplicity** when full synchronization is not necessary

- Use for **scalability** when reads outnumber writes
  - Or, use an actual read/write lock

Source: **Java Theory and Practice: Managing Volatility** *by Brian Goetz*

**CS 272 Software Development**
Professor Sophie Engle

**Department of Computer Science**
https://www.cs.usfca.edu/

UNIVERSITY OF
SAN FRANCISCO