Overview of Concurrent Programming Concepts

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

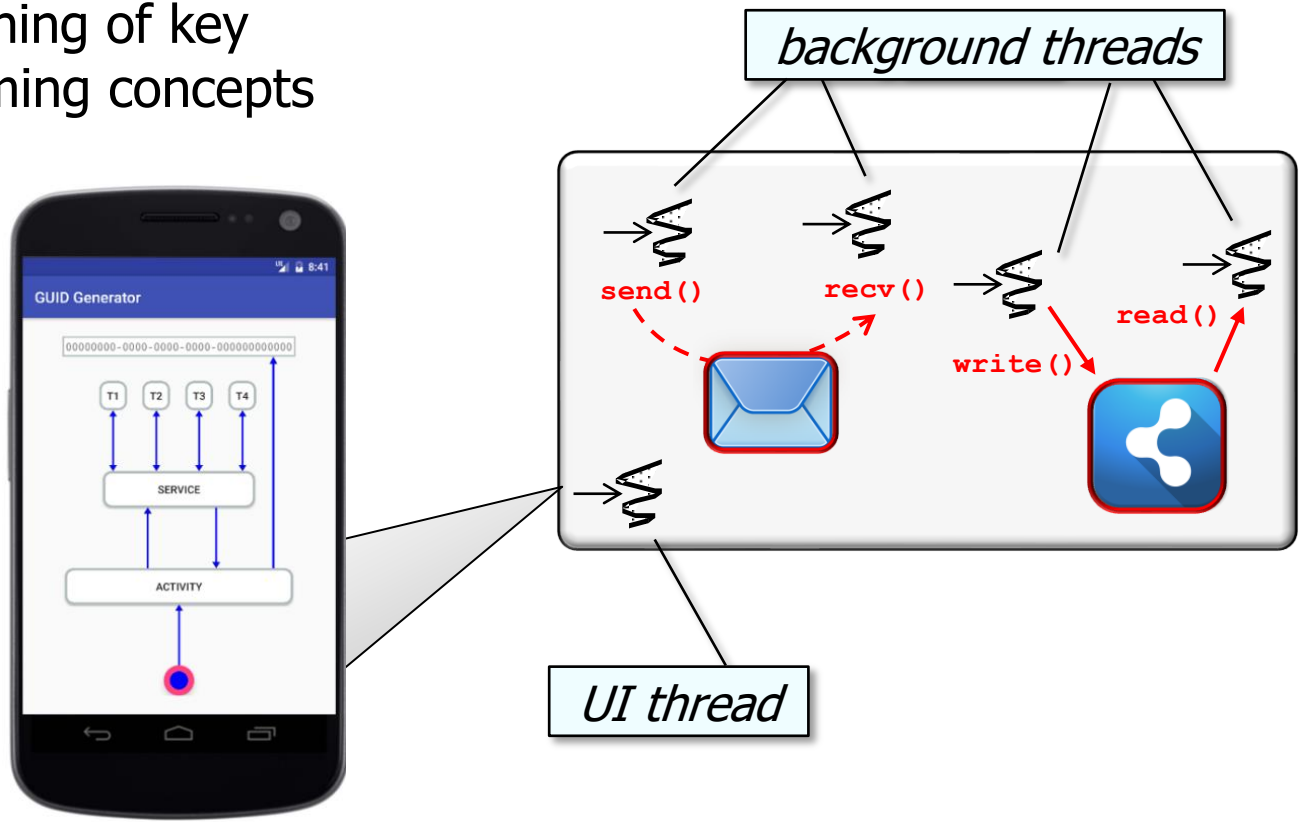
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand the meaning of key concurrent programming concepts



An Overview of Sequential Programming

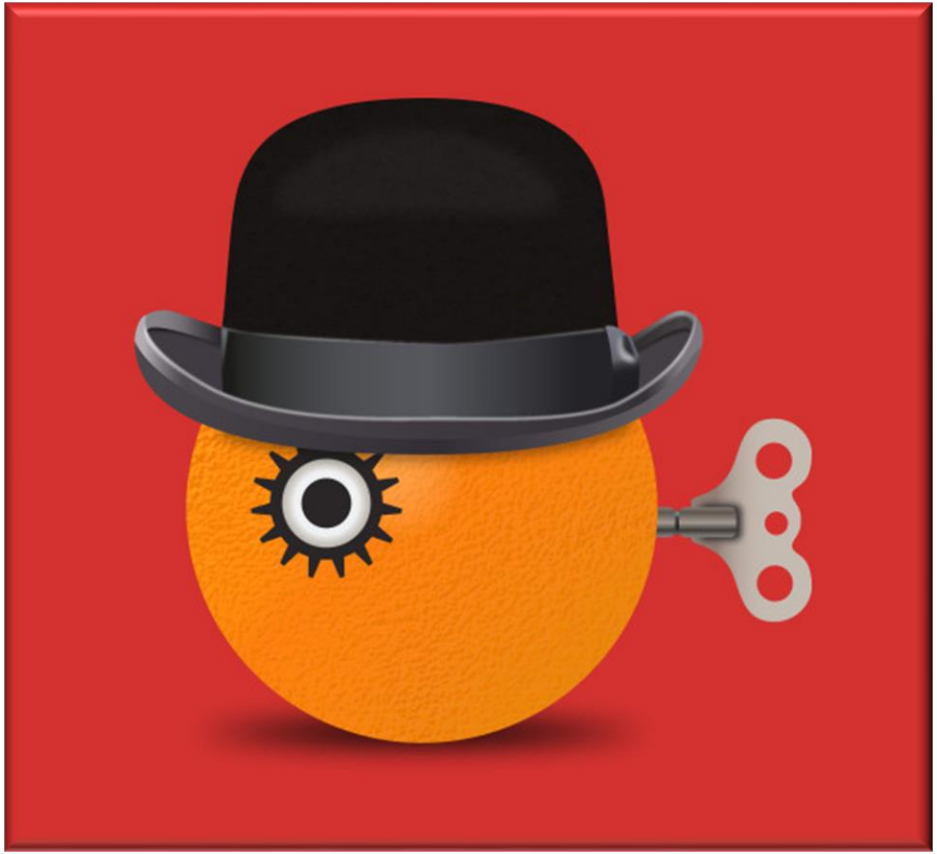
An Overview of Sequential Programming

- Sequential programming is a form of computing that executes the same sequence of instructions & always produces the same results

See en.wikipedia.org/wiki/Sequential_algorithm

An Overview of Sequential Programming

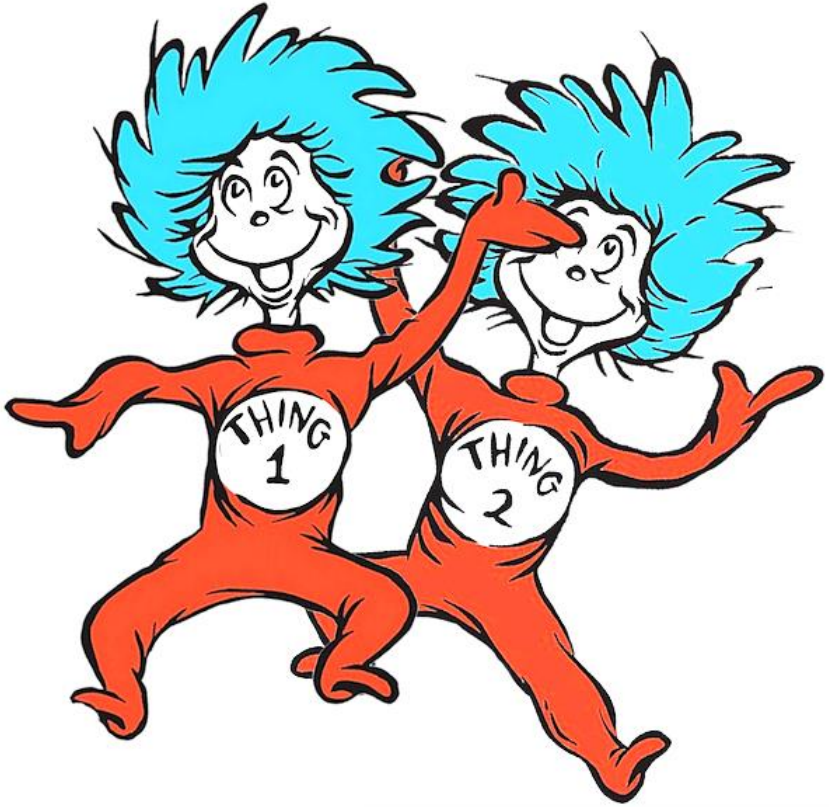
- Sequential programming is a form of computing that executes the same sequence of instructions & always produces the same results
  - i.e., execution is deterministic (assuming no deliberate use of randomness)



See screenprism.com/insights/article/what-is-the-ludovico-technique-and-how-does-it-work

An Overview of Sequential Programming

- Sequential programs have two characteristics



An Overview of Sequential Programming

- Sequential programs have two characteristics:
  - The textual order of statements specifies their order of execution

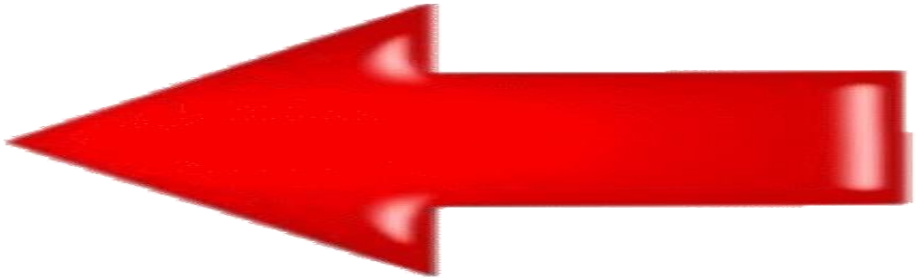
```
public E get(int index) {
  rangeCheck(index);

  return elementData
                  (index);
}
```

*e.g., the rangeCheck() method **must** be called before the elementData() method*

See src/share/classes/java/util/ArrayList.java

An Overview of Sequential Programming

- Sequential programs have two characteristics:
  - The textual order of statements specifies their order of execution
  - Successive statements must execute without any temporal overlap visible to programmers or programs



An Overview of Sequential Programming

- Sequential programs have two characteristics:
  - The textual order of statements specifies their order of execution
  - Successive statements must execute without any temporal overlap visible to programmers or programs
    - However, instructions can be reordered transparently to avoid pipeline stalls

- **For the code sequence:**

$$a = b + c$$

$$d = e - f$$

a, b, c, d ,e, and f are in memory

- **Assuming loads have a latency of one clock cycle, the following code or pipeline compiler schedule eliminates stalls:**

Original code with stalls:

```
        LD    Rb,b
        LD    Rc,c
Stall→
        DADD  Ra,Rb,Rc
        SD    Ra,a
        LD    Re,e
        LD    Rf,f
Stall→
        DSUB  Rd,Re,Rf
        SD    Rd,d
```

Scheduled code with no stalls:

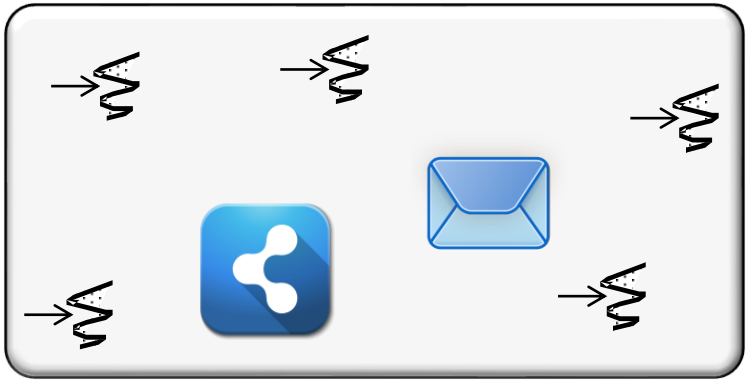
```
LD    Rb,b
LD    Rc,c
LD    Re,e
DADD  Ra,Rb,Rc
LD    Rf,f
SD    Ra,a
DSUB  Rd,Re,Rf
SD    Rd,d
```

See en.wikipedia.org/wiki/Instruction_scheduling

An Overview of Concurrent Programming

An Overview of Concurrent Programming

- Concurrent programming is a form of computing where threads can run simultaneously



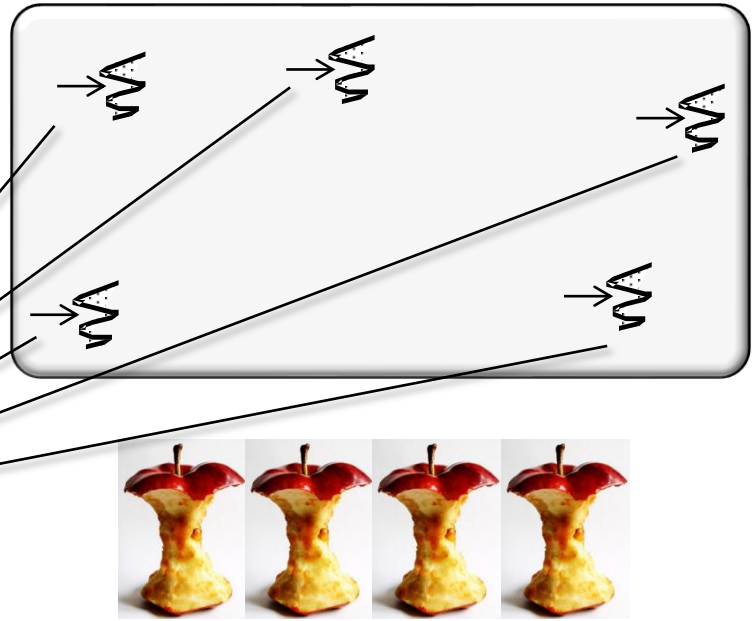
See en.wikipedia.org/wiki/Concurrency_(computer_science)

An Overview of Concurrent Programming

- Concurrent programming is a form of computing where threads can run simultaneously

```
for (int i = 0; i < 5; i++)
  new Thread(() ->
              someComputation()).
              start();
```

A thread is a unit of execution for instruction streams that can run concurrently on 1+ processor cores

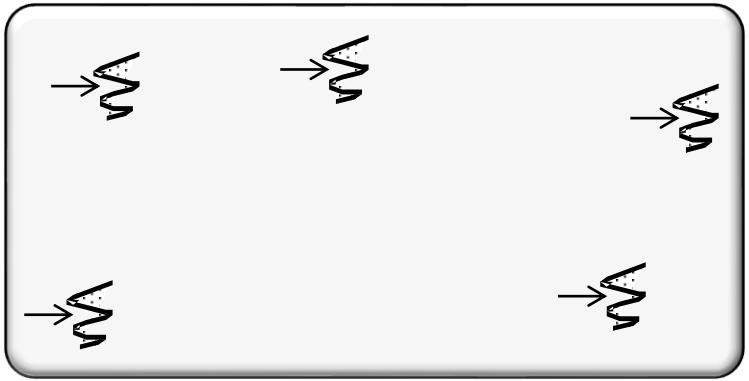


See docs.oracle.com/javase/tutorial/essential/concurrency/threads.html

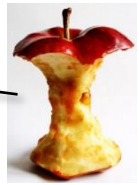
An Overview of Concurrent Programming

- Concurrent programming is a form of computing where threads can run simultaneously

```
for (int i = 0; i < 5; i++)
  new Thread(() ->
              someComputation()).
              start();
```



Threads may be multiplexed over one core, though this is increasingly rare..



See en.wikipedia.org/wiki/Single-core

An Overview of Concurrent Programming

- Different executions of a concurrent program may produce different instruction orderings

See en.wikipedia.org/wiki/Nondeterministic_algorithm

An Overview of Concurrent Programming

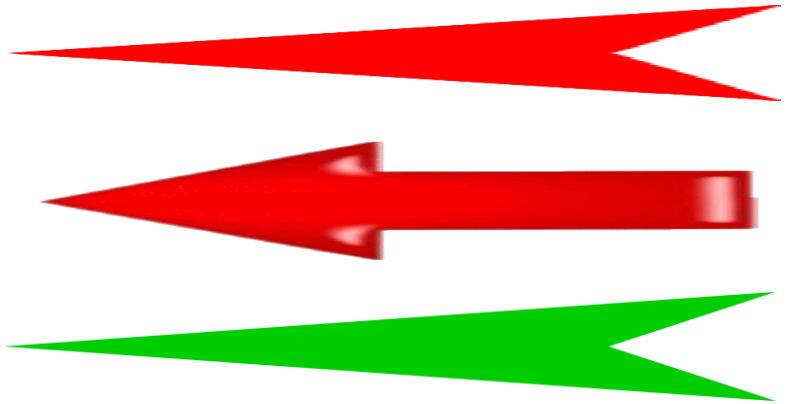
- Different executions of a concurrent program may produce different instruction orderings:
  - The textual order of the source code doesn't define the order of execution

computationA(), computationB(), & computationC() can run in any order after their threads start executing

```
new Thread(() ->
           computationA()).
           start();

new Thread(() ->
           computationB()).
           start();

new Thread(() ->
           computationC()).
           start();
```

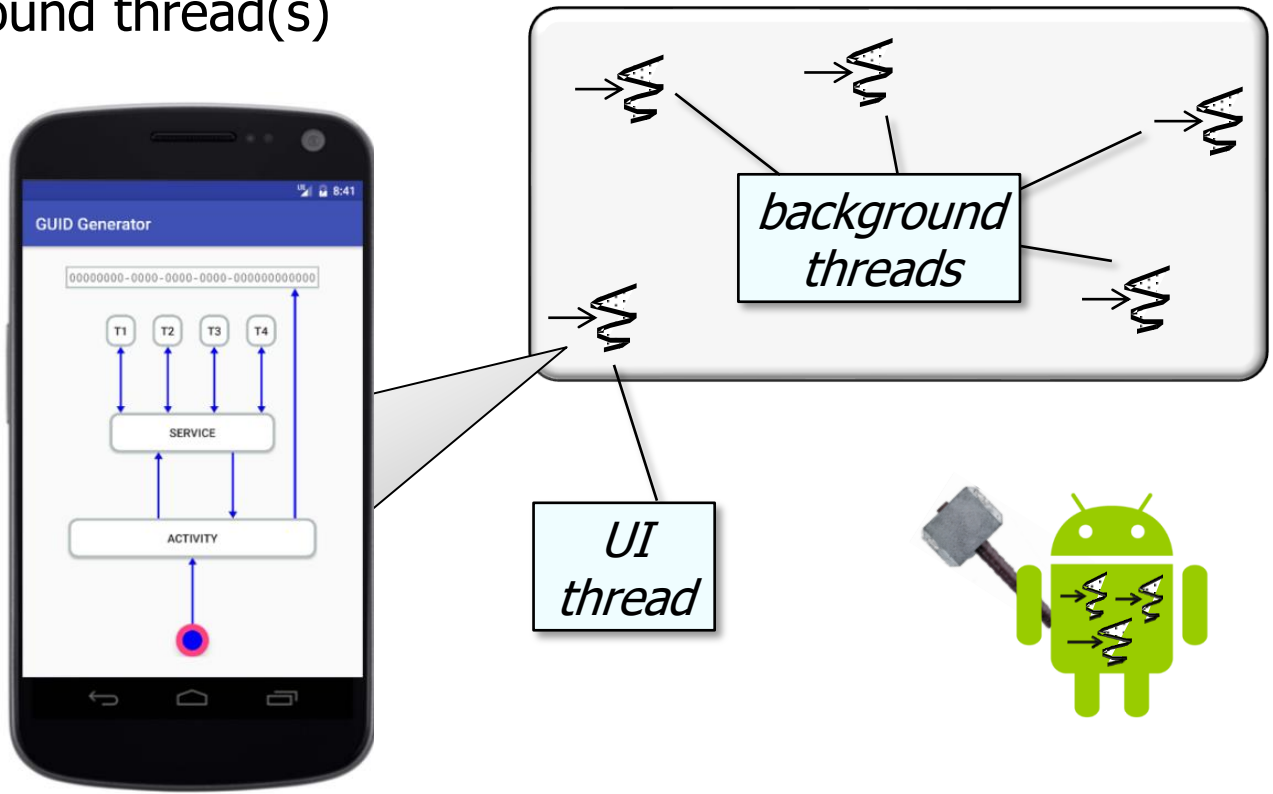
An Overview of Concurrent Programming

- Different executions of a concurrent program may produce different instruction orderings:
  - The textual order of the source code doesn't define the order of execution
  - Operations are permitted to overlap in time across multiple cores



An Overview of Concurrent Programming

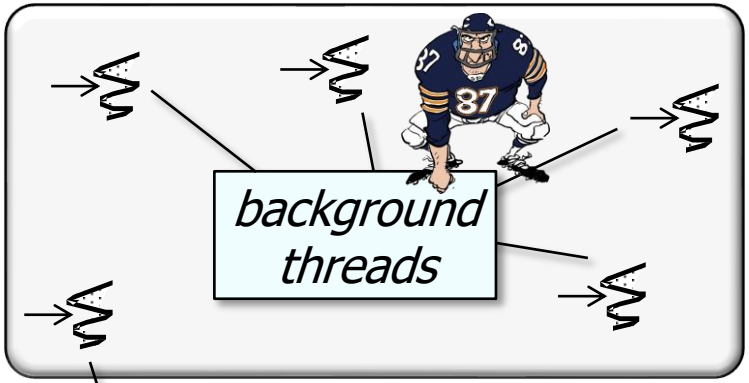
- Concurrent programming is often used to offload work from the user interface (UI) thread to background thread(s)

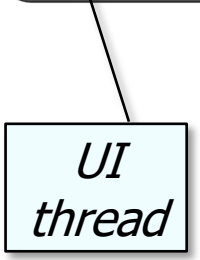


See developer.android.com/topic/performance/threads.html

An Overview of Concurrent Programming

- Concurrent programming is often used to offload work from the user interface (UI) thread to background thread(s), e.g.
  - Background thread(s) can block

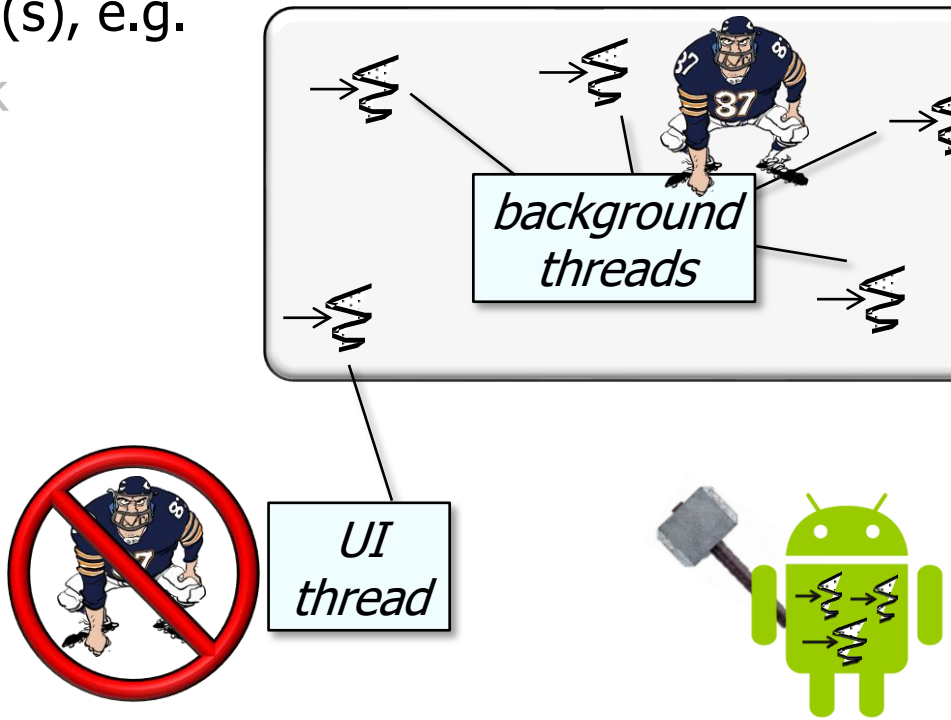




See developer.android.com/training/multiple-threads/communicate-ui.html

An Overview of Concurrent Programming

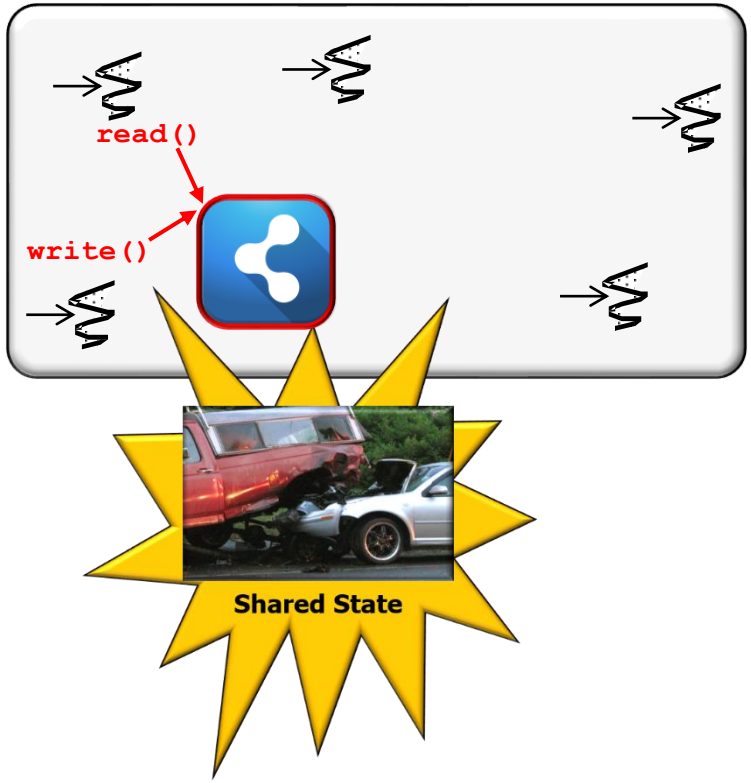
- Concurrent programming is often used to offload work from the user interface (UI) thread to background thread(s), e.g.
  - Background thread(s) can block
  - The UI thread does not block



See developer.android.com/training/multiple-threads/communicate-ui.html

An Overview of Concurrent Programming

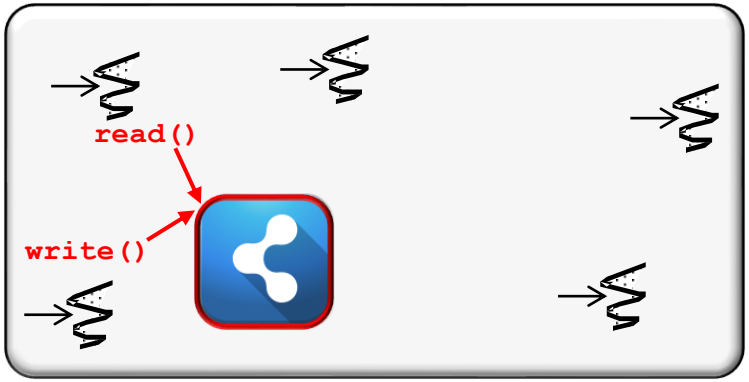
- Concurrent programming is often used to offload work from the user interface (UI) thread to background thread(s), e.g.
  - Background thread(s) can block
  - The UI thread does not block
  - Any mutable state shared between these threads must be protected to avoid concurrency hazards



See upcoming lesson on "*Overview of Concurrency in Java*"

An Overview of Concurrent Programming

- Concurrent programming is often used to offload work from the user interface (UI) thread to background thread(s), e.g.
  - Background thread(s) can block
  - The UI thread does not block
  - Any mutable state shared between these threads must be protected to avoid concurrency hazards
    - Motivates the need for various types of synchronizers



See upcoming lesson on "*Overview of Java Synchronizers*"

End of Overview of Concurrent Programming Concepts