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 concepts associated with concurrent programming
  - e.g., where two or more threads can run simultaneously & interact via shared objects & message passing

Concurrent programming helps address ‘cons’ of sequential programming
An Overview of Concurrent Programming
An Overview of Concurrent Programming

• Concurrent programming is a form of computing where two or more threads can run simultaneously

See [en.wikipedia.org/wiki/Concurrency_(computer_science)](en.wikipedia.org/wiki/Concurrency_(computer_science))
An Overview of Concurrent Programming

- Concurrent programming is a form of computing where two or more threads can run simultaneously.

A thread is a unit of execution for a stream of instructions that can run concurrently on one or more processor cores over its lifetime.

See docs.oracle.com/javase/tutorial/essential/concurrency/threads.html
An Overview of Concurrent Programming

- Concurrent programming is a form of computing where two or more threads can run simultaneously.

A thread typically runs in a process, which allocates & manages resources (e.g., files, memory, & network connections) & prevents corruption from threads in other processes.

See en.wikipedia.org/wiki/Process_(computing)
An Overview of Concurrent Programming

Concurrent programming is a form of computing where two or more threads can run simultaneously.

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

This code snippet creates/starts 5 Java Thread objects that run someComputation concurrently across 4 processor cores.
An Overview of Concurrent Programming

• Concurrent programming is a form of computing where two or more threads can run simultaneously

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

A Java Thread object needn’t run on the same core throughout its lifetime, but instead it can be “multiplexed” across multiple cores via “time-slicing”

See scalibq.wordpress.com/2012/06/01/multi-core-and-multi-threading
An Overview of Concurrent Programming

- Concurrent programming is a form of computing where two or more threads can run simultaneously.

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

Multiple threads can also be multiplexed over a single-core processor.
An Overview of Concurrent Programming

- Concurrent programming is a form of computing where two or more threads can run simultaneously.

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

However, single-core processors are becoming rare for general-purpose computing devices.

An Overview of Concurrent Programming

- Threads can interact via shared objects (synchronizers) & message passing

See upcoming lesson on “Overview of How Concurrent Programs are Developed in Java”
An Overview of Concurrent Programming

• Threads can interact via shared objects (synchronizers) & message passing

Shared objects (synchronizers) can be used to ensure mutual exclusion between—and coordination amongst—multiple threads

See upcoming lesson on “Overview of How Concurrent Programs are Developed in Java”
An Overview of Concurrent Programming

- Threads can interact via shared objects (synchronizers) & message passing

Multiple threads can pass messages via queues that are properly synchronized

See upcoming lesson on “Overview of How Concurrent Programs are Developed in Java”
An Overview of Concurrent Programming

• Unlike sequential programming, different executions of a concurrent program may produce different orderings of instructions:

See earlier lesson on “Overview of Sequential Programming Concepts”
An Overview of Concurrent Programming

- Unlike sequential programming, different executions of a concurrent program may produce different orderings of instructions:
- The textual order of the source code doesn’t define the order of execution

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

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

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

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

See [en.wikipedia.org/wiki/Indeterminacy_in_concurrent_computation](en.wikipedia.org/wiki/Indeterminacy_in_concurrent_computation)
Unlike sequential programming, different executions of a concurrent program may produce different orderings of instructions:

- The textual order of the source code doesn’t define the order of execution
- Operations are permitted to overlap in time across multiple cores

Multiple computations can execute concurrently (during overlapping time periods) instead of sequentially (with one completing before the next starts)

See [en.wikipedia.org/wiki/Concurrent_computing](en.wikipedia.org/wiki/Concurrent_computing)
An Overview of Concurrent Programming

- Concurrent programming can 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 can 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
Concurrent programming can 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 can 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

- e.g., a “race condition” can occur when a program depends upon the sequence or timing of threads for it to operate properly

See upcoming lesson on “Overview of Concurrency in Java’"
An Overview of Concurrent Programming

• Concurrent programming can 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 Java synchronizers

See docs.oracle.com/javase/tutorial/essential/concurrency(sync.html)
Concurrent programming can 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
- Message passing mechanisms can be used to avoid sharing state across multiple threads
End of Overview of Concurrent Programming Concepts