Overview of Concurrent Programming in Java

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
• Recognize how these concepts are supported in Java

Each Java thread has its own stack, registers, etc.

Java Class:
- yield():void
- currentThread():Thread
- sleep(long):void
- sleep(long,int):void
- Thread()
- Thread(Runnable)
- Thread(String)
- start():void
- run():void
- exit():void
- interrupt():void
- interrupted():boolean
- isInterrupted():boolean
- isAlive():boolean
- setPriority(int):void
- getPriority():int
- join(long):void
- join(long,int):void
- join():void
- setDaemon(boolean):void
- isDaemon():boolean
An Overview of Concurrent Programming in Java
A Java thread is an object

Class Thread

```java
public class Thread
    extends Object
    implements Runnable

A thread is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.

Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority. Each thread may or may not also be marked as a daemon. When code running in some thread creates a new Thread object, the new thread has its priority initially set equal to the priority of the creating thread, and is a daemon thread if and only if the creating thread is a daemon.
```

See [docs.oracle.com/javase/8/docs/api/java/lang/Thread.html](https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html)
An Overview of Concurrent Programming in Java

- A Java thread is an object, e.g.
  - It contains methods & fields

Each Java thread has its own stack, registers, etc.

See blog.jamesdbloom.com/JVMInternals.html
An Overview of Concurrent Programming in Java

- A Java thread is an object, e.g.
  - It contains methods & fields
  - It can also be in one of various “states”

See [docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.html](http://docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.html)
An Overview of Concurrent Programming in Java

• Concurrent Java threads interact via shared objects and/or message passing

See docs.oracle.com/javase/8/docs/api/?java/util/concurrent/package-summary.html
An Overview of Concurrent Programming in Java

- Concurrent Java threads interact via shared objects and/or message passing
- **Shared objects**
  - Synchronize concurrent operations on objects to ensure certain properties

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

- Concurrent Java threads interact via shared objects and/or message passing

- **Shared objects**
  - Synchronize concurrent operations on objects to ensure certain properties, e.g.
    - **Mutual exclusion**
      - Interactions between threads won’t corrupt shared mutable data

See [en.wikipedia.org/wiki/Monitor_(synchronization)#Mutual_exclusion](en.wikipedia.org/wiki/Monitor_(synchronization)#Mutual_exclusion)
An Overview of Concurrent Programming in Java

- Concurrent Java threads interact via shared objects and/or message passing

  - **Shared objects**
    - Synchronize concurrent operations on objects to ensure certain properties, e.g.
      - **Mutual exclusion**
      - **Coordination**
        - Operations occur in the right order, at the right time, & under the right conditions

See [en.wikipedia.org/wiki/Monitor_(synchronization)#Condition_variables](en.wikipedia.org/wiki/Monitor_(synchronization)#Condition_variables)
An Overview of Concurrent Programming in Java

- Concurrent Java threads interact via shared objects and/or message passing

**Shared objects**
- Synchronize concurrent operations on objects to ensure certain properties

Examples of Java synchronizers:
- Synchronized statements/methods
- Reentrant locks & intrinsic locks
- Atomic operations
- Semaphores
- Condition objects
- “Compare-and-swap” (CAS) operations in sun.misc.unsafe

See [dzone.com/articles/the-java-synchronizers](dzone.com/articles/the-java-synchronizers)
An Overview of Concurrent Programming in Java

- Concurrent Java threads interact via shared objects and/or message passing
  - Shared objects
  - Message passing
    - Send message(s) from producer thread(s) to consumer thread(s) via a thread-safe queue

See en.wikipedia.org/wiki/Message_passing
An Overview of Concurrent Programming in Java

• Concurrent Java threads interact via shared objects and/or message passing

- **Shared objects**

- **Message passing**
  - Send message(s) from producer thread(s) to consumer thread(s) via a thread-safe queue

- Examples of Java thread-safe queues
  - Array & linked blocking queues
  - Priority blocking queue
  - Synchronous queue
  - Concurrent linked queue

See [docs.oracle.com/javase/tutorial/collections/implementations/queue.html](https://docs.oracle.com/javase/tutorial/collections/implementations/queue.html)
An Overview of Concurrent Programming Hazards
An Overview of Concurrent Programming Hazards

- Java shared objects & message passing are designed to share resources safely & avoid concurrency hazards

See en.wikipedia.org/wiki/Thread_safety
An Overview of Concurrent Programming Hazards

- Java shared objects & message passing are designed to share resources safely & avoid concurrency hazards, e.g.
- Race conditions
  - Race conditions occur when a program depends upon the sequence or timing of threads for it to operate properly

See en.wikipedia.org/wiki/Race_condition#Software
An Overview of Concurrent Programming Hazards

- Java shared objects & message passing are designed to share resources safely & avoid concurrency hazards, e.g.
  - Race conditions
    - Race conditions occur when a program depends upon the sequence or timing of threads for it to operate properly

**This test program induces race conditions due to lack of synchronization between producer & consumer threads accessing a bounded queue**

See [github.com/douglascraigschmidt/LiveLessons/tree/master/BuggyQueue](https://github.com/douglascraigschmidt/LiveLessons/tree/master/BuggyQueue)
An Overview of Concurrent Programming Hazards

- Java shared objects & message passing are designed to share resources safely & avoid concurrency hazards, e.g.
  - Race conditions
  - Memory inconsistencies
    - These errors occur when different threads have inconsistent views of what should be the same data

An Overview of Concurrent Programming Hazards

• Java shared objects & message passing are designed to share resources safely & avoid concurrency hazards, e.g.
  • Race conditions
  • Memory inconsistencies
  • Deadlocks
    • Occur when 2+ competing threads are waiting for the other(s) to finish, & thus none ever do

See en.wikipedia.org/wiki/Deadlock
End of Overview of Concurrent Programming in Java