Implementing a “Fair” Semaphore
Via the Specific Notification Pattern

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

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

• Understand the *Specific Notification* pattern

• Be aware of the semantics of “fair” semaphores

---

**Class Semaphore**

```java
java.lang.Object
    java.util.concurrent.Semaphore

All Implemented Interfaces:
Serializable

public class Semaphore
extends Object
implements Serializable

A counting semaphore. Conceptually, a semaphore maintains a set of permits. Each acquire() blocks if necessary until a permit is available, and then takes it. Each release() adds a permit, potentially releasing a blocking acquirer. However, no actual permit objects are used; the Semaphore just keeps a count of the number available and acts accordingly.

Semaphores are often used to restrict the number of threads than can access some (physical or logical) resource. For example, here is a class that uses a semaphore to control access to a pool of items:
```
An Overview of Fair Semaphore Semantics
Threads calling acquire() on a "fair" semaphore obtain permits in "first-in, first-out" (FIFO) order. 

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Semaphore.html

Overview of Fair Semaphore Semantics

Class Semaphore

```
java.lang.Object
    java.util.concurrent.Semaphore

All Implemented Interfaces:
    Serializable

public class Semaphore
    extends Object
    implements Serializable

A counting semaphore. Conceptually, a semaphore maintains a set of permits. Each acquire() blocks if necessary until a permit is available, and then takes it. Each release() adds a permit, potentially releasing a blocking acquirer. However, no actual permit objects are used; the Semaphore just keeps a count of the number available and acts accordingly.

Semaphores are often used to restrict the number of threads than can access some (physical or logical) resource. For example, here is a class that uses a semaphore to control access to a pool of items:
```
Overview of Fair Semaphore Semantics

- Threads calling acquire() on a “fair” semaphore obtain permits in “first-in, first-out” (FIFO) order.
- FIFO ordering applies to internal points of execution within semaphore methods.

```java
public class Semaphore
    extends Object
    implements Serializable

A counting semaphore. Conceptually, a semaphore maintains a set of permits. Each acquire() blocks if necessary until a permit is available, and then takes it. Each release() adds a permit, potentially releasing a blocking acquirer. However, no actual permit objects are used; the Semaphore just keeps a count of the number available and acts accordingly.

Semaphores are often used to restrict the number of threads than can access some (physical or logical) resource. For example, here is a class that uses a semaphore to control access to a pool of items:
```
Overview of Fair Semaphore Semantics

- Threads calling acquire() on a “fair” semaphore obtain permits in “first-in, first-out” (FIFO) order
  - FIFO ordering applies to internal points of execution within semaphore methods
    - e.g., one thread can invoke acquire() before another, but reach the ordering point after the other

```java
public class Semaphore extends Object implements Serializable
```

A counting semaphore. Conceptually, a semaphore maintains a set of permits. Each acquire() blocks if necessary until a permit is available, and then takes it. Each release() adds a permit, potentially releasing a blocking acquirer. However, no actual permit objects are used; the Semaphore just keeps a count of the number available and acts accordingly.

Semaphores are often used to restrict the number of threads than can access some (physical or logical) resource. For example, here is a class that uses a semaphore to control access to a pool of items:
Overview of Fair Semaphore Semantics

- Threads calling acquire() on a "fair" semaphore obtain permits in "first-in, first-out" (FIFO) order.
- FIFO ordering applies to internal points of execution within semaphore methods.
- The Specific Notification pattern provides an effective model for implementing fair semaphore semantics.

See www.dre.vanderbilt.edu/~schmidt/PDF/specific-notification.pdf (especially Listing 3)
End of Implementing a “Fair” Semaphore Via the Specific Notification Pattern