JAC444 / BTP400 Course Object-Oriented Software Development II - Java

Threads

Segment 3

Threads

In this section you will be learning about:

- Solutions to Liveness Problems
- Guarded Blocks
- Immutable Objects
- High Level Concurrency

Guarded Blocks

- Threads have to coordinate their actions (they must work together).
- The *guarded block* is the most common coordination idiom for threads coordination.
- The guarded block uses three methods form `Object` class:

`1. wait()`
Causes the current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object.

`2. notify()`
Wakes up a single thread that is waiting on this object's monitor

`3. notifyAll()`
Wakes up all threads that are waiting on this object's monitor.

wait()notify()notifyAll()

wait()

Makes a thread to wait until some conditions are satisfied.
Places the invoking thread on the monitor's waiting list.

notify()/notifyAll()

Tells waiting thread/s that something has occurred that might satisfy that condition.
Reactivates one/all threads in monitor's waiting list.

Important note:

`wait()`, `notify()`, `notifyAll()` are not methods of `Thread` class, but rather of the `Object` class.

`wait()` Method Idiom

When `wait` is invoked, the thread releases the lock and suspends execution

```
public synchronized void guardedExamResult() {

    // This guard only loops once for each special event, which
    // may not be the event we're waiting for.

    while(!examResult) {
        try {
            wait();
        } catch (InterruptedException e) {}
    }

    System.out.println("Exam Result have been received!");
}
```

Important note:

Always invoke `wait` inside a loop that tests for the condition being waited for.

`notifyAll()` Method Idiom

When `notifyAll` is invoked, it informs all threads waiting on a lock that something important has happened

```
public synchronized notifyExamResult() {
    examResult = true;
    notifyAll();
}
```

Important note:

There is a second notification method, `notify`, which wakes up a single thread. The `notify` method doesn't allow you to specify the thread that is woken up

Look in the lab for Producer/Consumer example.

Immutable Objects

- If the state of object is not changed after its creation, the object is called an *immutable* object (e.g., String object)
- Since immutable object cannot change its state, it cannot be corrupted by thread interference
- Strategies to make an object immutable:

1. Make the constructor private and construct instances in factory methods
2. Make all fields final and private.
3. Don't allow subclasses to override methods
4. Don't implement methods that modify fields or objects referred to by fields.

High Level Concurrency Objects

Most are implemented in the new **`java.util.concurrent`** packages:

- *Lock objects*

They support locking idioms that simplify many concurrent applications.

- *Executors*

They define a high-level API for launching and managing threads

- *Concurrent collections*

It make it easier to manage large collections of data

- *Atomic variables*

They have properties that help avoid memory consistency errors

Lock Objects

Lock objects are similar to the implicit locks used by synchronized code, but they have the ability to back out of an attempt to acquire a lock

- Lock objects also support a wait/notify mechanism, through their associated `Condition` objects

- The `tryLock` method backs out if the lock is not available immediately or before a timeout expires

- The `lockInterruptibly` method backs out if another thread sends an interrupt before the lock is acquired

Note: *See samples in the lab*

Executors

Objects that separate:
1. the thread object itself as defined by **`Thread`** class
2. the task being done by thread, as defined by its **`Runnable`** interface

are called *executors*

- Executor Interfaces

They define the three executor object types.

- Thread Pools

They are the executor implementation

- Fork/Join

New in JDK 7 this is a framework for multiple processors.

Note: *See samples in the lab*

Threads

Do not rely on thread priority for algorithm correctness.

Use concurrency as a normal programming style in Java

Further Readings:

Concurrent Programming in Java: Design Principles and Pattern (2nd Edition) by Doug Lea.

Effective Java Programming Language Guide (2nd Edition) by Joshua Bloch.

Concurrency: State Models & Java Programs (2nd Edition), by Jeff Magee and Jeff Kramer.