Concurrency
November 27, 2007

Thread Classes

<<interface>>
Runnable
void run()

Thread
void interrupt()
void sleep (long millis)
void start()
(Simplified) Thread State Machine

Terminated

Runnable

New

Sleeping

start

time out, interrupt

run method ends

sleep

Terminated

Runnable

HelloWithCancel Classes

<<interface>>
Runnable
void run()

SlowCounter
void run()

Thread
void interrupt()
void sleep (long millis)
void start()

Hello
void actionPerformed

Wednesday, November 28, 2007
Creating and Starting the thread

2nd thread running

Interrupting the 2nd thread

Shared Data

- An object can be referenced by more than one thread -> shared data
- Changing the value in one thread can affect the behavior of the other thread!

<table>
<thead>
<tr>
<th>BankAccount</th>
<th>Consider a joint account where each account holder simultaneously uses a different ATM</th>
</tr>
</thead>
<tbody>
<tr>
<td>int balance</td>
<td></td>
</tr>
<tr>
<td>void withdraw (int amount)</td>
<td></td>
</tr>
<tr>
<td>void deposit (int amount)</td>
<td></td>
</tr>
<tr>
<td>int getBalance()</td>
<td></td>
</tr>
</tbody>
</table>
Race Condition

<table>
<thead>
<tr>
<th>Account</th>
<th>Withdrawer</th>
<th>Depositer</th>
</tr>
</thead>
<tbody>
<tr>
<td>balance = 1000</td>
<td>withdraw(10)</td>
<td></td>
</tr>
<tr>
<td>newBalance = 990</td>
<td></td>
<td></td>
</tr>
<tr>
<td>balance = 990</td>
<td></td>
<td></td>
</tr>
<tr>
<td>deposit(10)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>newBalance = 1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>balance = 1000</td>
<td>withdraw(10)</td>
<td></td>
</tr>
<tr>
<td>in withdraw newBalance = 990</td>
<td></td>
<td></td>
</tr>
<tr>
<td>deposit(10)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>in deposit newBalance = 1010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>balance = 1010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>balance = 990</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Locking Objects

- Need to assure that deposit and withdraw do not interleave
- Java solution: Declare methods to be synchronized
- Entering a synchronized method locks the object
- Exiting unlocks it
- If object is already locked, Java waits at the beginning of the synchronized method

Wednesday, November 28, 2007
Synchronized Methods

```java
public synchronized void withdraw (int amount) {
    //balance = balance - amount;
    int newBalance = balance - amount;
    Thread.yield();
    balance = newBalance;
    notifyListeners();
}

public synchronized void deposit (int amount) {
    //balance = balance + amount;
    int newBalance = balance + amount;
    Thread.yield();
    balance = newBalance;
    notifyListeners();
}
```

Threads

- New -> Runnable
  - start
  - yield

- Runnable
  - sleep
  - time out, interrupt
  - waiting for a lock
  - lock acquired

- Sleeping
  - yield
  - lock acquired

- Blocked
  - yield

- Terminated
  - ()

Wednesday, November 28, 2007
Account | Withdrawer | Depositer
--- | --- | ---
balance = 1000 | withdraw(10) | 
lock | 
newBalance = 990 | balance = 990 | unlock
lock | deposit(10) | 
lock | 
newBalance = 1000 | balance = 1000 | unlock
lock | withdraw(10) | 
lock | 
in withdraw newBalance = 990 | deposit(10) | wait for lock
balance = 990 | | unlock
lock | 
in deposit newBalance = 1000 | balance = 1000 | unlock

**Wait/Notify**

че What if we don’t want to allow a withdrawal if it would bring the balance below $0?

че Suppose we know a deposit will happen soon?

```java
public synchronized void withdraw (int amount) {
    //balance = balance - amount;
    if (balance < 10) {
        ??? throw an exception ???
    }
    int newBalance = balance - amount;
    balance = newBalance;
    notifyListeners();
}
```
**Wait/Notify**

- `wait()` temporarily gives up the lock
- When should we get it back?

```java
public synchronized void withdraw (int amount) {
    //balance = balance - amount;
    while (balance < 10) {
        wait();
    }
    int newBalance = balance - amount;
    balance = newBalance;
    notifyListeners();
}
```

**Wait/Notify**

- `notify()` puts waiting threads in queue for the lock

```java
public synchronized void deposit (int amount) {
    //balance = balance + amount;
    int newBalance = balance + amount;
    balance = newBalance;
    notifyAll();
    notifyListeners();
}
```
### Threads

- **New**
  - start
  - wait

- **Runnable**
  - run method
  - ends
  - sleep
  - wait
  - sleeping
  - waiting for a lock

- **Waiting**
  - notified

- **Blocked**
  - locked acquired

- **Terminated**

### Account Management

<table>
<thead>
<tr>
<th>Account</th>
<th>Withdrawer</th>
<th>Depositor</th>
</tr>
</thead>
<tbody>
<tr>
<td>balance = 90</td>
<td>withdraw(100)</td>
<td></td>
</tr>
<tr>
<td>lock</td>
<td></td>
<td>deposit(100)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>wait for lock</td>
</tr>
<tr>
<td>newBalance = -10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unlock and wait</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lock</td>
<td></td>
<td></td>
</tr>
<tr>
<td>balance = 190</td>
<td></td>
<td></td>
</tr>
<tr>
<td>notifyAll</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unlock</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lock</td>
<td></td>
<td></td>
</tr>
<tr>
<td>in withdraw newBalance = 90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>balance = 90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unlock</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Thread Safety

- A class is thread safe if it is correct even if accessed by multiple threads

- 2 simple cases:
  - A class with no instance or static variables
  - An immutable class

- Otherwise:
  - Find all methods that read and write the same instance variable
  - Only need to worry about non-final instance variables
  - Make those methods synchronized

This will avoid race conditions but might introduce...deadlock! Aye, matie!