Multi-Threaded Programming

- Thread: sequential execution of a series of instructions.
- Most conventional programming languages are single-threaded.
- Multi-threaded programming are more difficult:
  - shared access to objects
  - race hazard

A Bank Account

class Account {
  // ...
  public boolean withdraw(long amount) {
    if (amount <= balance) {
      long newbalance = balance - amount;
      balance = newbalance;
      return true;
    } else
      return false;
  }
  private long balance;
}

A "Perfect" Crime

Assume the initial balance is $1,000,000. Two withdraw requests are made almost simultaneously.

<table>
<thead>
<tr>
<th>balance</th>
<th>withdraw 1</th>
<th>withdraw 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000,000</td>
<td>amount&lt;=balance</td>
<td>amount&lt;=balance</td>
</tr>
<tr>
<td>1,000,000</td>
<td>newbalance=...;</td>
<td>newbalance=...;</td>
</tr>
<tr>
<td>1,000,000</td>
<td>balance=...;</td>
<td>balance=...;</td>
</tr>
<tr>
<td>0</td>
<td>balance=...;</td>
<td>balance=...;</td>
</tr>
<tr>
<td>0</td>
<td>return true;</td>
<td>return true;</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Creating Threads

Method A:
- Subclass the Thread class.
- Override the run() method.
- Create a thread with new MyThread(...).
- Start the thread by calling the start() method.

Method B:
- Implement the Runnable interface.
- Override the run() method.
- Create a thread with new Thread(runnable).
- Start the thread by calling the start() method.
A Simple Counter

```java
public class Counter1 extends Thread {
    protected int count, inc, delay;

    public Counter1(int init, int inc, int delay) {
        this.count = init;
        this.inc = inc;
        this.delay = delay;
    }

    public void run() {
        try {
            for (;;) {
                System.out.print(count + " ");
                count += inc;
                sleep(delay);
            }
        } catch (InterruptedException e) {}
    }
}
```

A Simple Counter II

```java
public class Counter2 implements Runnable {
    protected int count, inc, delay;

    public Counter2(int init, int inc, int delay) {
        this.count = init;
        this.inc = inc;
        this.delay = delay;
    }

    public void run() {
        try {
            for (;;) {
                System.out.print(count + " ");
                count += inc;
                Thread.sleep(delay);
            }
        } catch (InterruptedException e) {}}
```
Synchronization

Mutual exclusion of threads.
- Each synchronized method or statement is guarded by an object.
- When entering a synchronized method or statement, the object will be locked until the method is finished.
- When the object is locked by another thread, the current thread must wait.

Synchronized Method and Statement

- Synchronized method:
  ```java
class MyClass{
    synchronized void aMethod(){
      statements
    }
}
```

- Synchronized statement:
  ```java
  synchronized(exp){
    statements
  }
  ```