

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 "Perfact" Crime

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

balance	withdraw 1	withdraw 2
1,000,000	amount<=balance	
1,000,000		amount<=balance
1,000,000	newbalance=...;	
1,000,000		newbalance=...;
0	balance=...;	
0		balance=...;
0	return true;	
0		return true;

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

```
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 (cont'd)

(class Counter1 continued.)

```
    public static void main(String[] args) {  
        new Counter1(0, 1, 33).start();  
        new Counter1(0, -1, 100).start();  
    }  
}
```

Output:

```
0 0 1 2 -1 3 4 5 -2 6 7 8 -3 9 10 -4 11 12 13  
-5 14 15 16 -6 17 18 -7 19 20 21 -8 22 23 24 -9  
25 26 -10 27 28 -11 29 30 31 -12 32 33 34 -13  
35 36 37 -14 38 39 -15 40 41 42 -16 43 44 45
```

A Simple Counter II

```
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) {}  
    }  
}
```

A Simple Counter II (cont'd)

(class Counter2 continued.)

```
    public static void main(String[] args) {  
        new Thread(new Counter2(0, 1, 33)).start();  
        new Thread(new Counter2(0, -1, 100)).start();  
    }  
}
```

Output:

```
0 0 1 2 -1 3 4 5 -2 6 7 8 -3 9 10 -4 11 12 13 -5  
14 15 16 -6 17 18 -7 19 20 21 -8 22 23 24 -9 25  
26 -10 27 28 -11 29 30 31 -12 32 33 34 -13 35 36  
-14 37 38 39 -15 40 41 42 -16 43 44 45 -17 46 47
```

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:

```
class MyClass{  
    synchronized void aMethod(){  
        statements  
    }  
}
```

- Synchronized statement:

```
synchronized(exp) {  
    statements  
}
```