From last time

Explain briefly how a deadlock may occur (2 marks)

A deadlock may occur in a set of processes, each holding some resources and waiting for resources held by another process in the set.

Explain the purpose of the semaphores in:

- Lines 2 & 3 of both threads (1 mark)
- Line 5 of A & line 4 of B. (1 mark)
- Lines 6 & 7 of A & lines 7 & 8 of B. (1 mark)

What is output by the print statement in line 8 of A? (3 marks)

1,1 2,2 3,3 4,5 5,8 6,13 7,21

Will A ever terminate? Justify your answer. (1 mark)

```java
class T1 extends Thread {
    public void run() {
        ...
    }
}
new T1().start();
```

Overview & Learning Outcomes

Creating and running threads
- java.lang.Thread (class)
- java.lang.Runnable (interface)
- run(), start()

Synchronized accesses to shared data
- synchronize methods & blocks
- wait(), notify(), notifyAll()

java.util.concurrent
Example 1

class MyThread extends Thread {
    public void run() {
        try {
            for (int i = 5; i > 0; i--)
                System.out.println( i );
            Thread.sleep(1000);
        }
        catch (InterruptedException e)
            System.out.println("child interrupted");
        System.out.println("exiting child thread");
    }
}

Example 1 ctd

public static void main(String[] args) {
    MyThread myt = new MyThread();
    myt.start();
    try {
        Thread.sleep(2000);
    }
    catch (InterruptedException e)
        System.out.println("interrupted");
    System.out.println("exiting main thread");
}

Question

What output would you expect from Example 1?

5
4
exiting main thread
3
2
1
exiting child thread

2) Runnable interface

class T2 implements Runnable {
    public void run() {...}
}
...
new Thread(new T2()).start();

Example 2

class ThreadDemo implements Runnable {
    ThreadDemo() {
        Thread ct = Thread.currentThread();
        Thread t = new Thread(this, "Demo Thread");
        System.out.println("currentThread: "+ ct);
        System.out.println("Thread created: "+ t);
        t.start();
        try {Thread.sleep(3000);} catch (InterruptedException e)
            System.out.println("interrupted");
        System.out.println("exiting main thread");
    }
    // public void run() as previous example
    public static void main(String args[]) {
        new ThreadDemo();
    }
}

Example 2 – output

currentThread: Thread[main,5,main]
Thread created: Thread[Demo Thread,5,main]
5
4
3
exiting main thread
2
1
exiting child thread
Synchronized Method

Every object/class has an associated mutually exclusive “lock”
use to synchronize access to object/class contents
Only one thread may hold the lock at any one time
Methods can be declared synchronized
i.e. the lock must be obtained before the method can start

Synchronized Block

synchronized (expression) { ... }
“expression” gives object (e.g. this)
whose lock will be obtained and held
while the following block { ... } is executed

Example – Bounded Buffer

We want a data buffer of fixed maximum size
to carry values between processes (or threads).
Able to read/write from buffer asynchronously (at any time),
but prevent buffer overflow/underflow
overflow: write a value when the buffer is already full
underflow: to read from an empty buffer

Example code – class BoundedBuffer

class BoundedBuffer {
    private int [] buffer;
    private int inPtr, outPtr, count, numEls;
    public BoundedBuffer (int size) {
        buffer= new int[size]; numEls= size;
        inPtr= 0; outPtr= 0; count= 0;
    }
    public synchronized void deposit(int message) throws InterruptedException {
        while (count == numEls) wait();
        buffer[inPtr]= message;
        inPtr= (inPtr + 1) % numEls;
        if (count++ == 0) notifyAll();
    }
}
Example code – class BoundedBuffer ctd.

```java
public synchronized int extract ()
  throws InterruptedException {
while (count == 0)
  wait();
int message= buffer[outPtr];
outPtr= (outPtr + 1) % numEls;
if (count-- == numEls)
  notifyAll();
return message;
}
} // end of class BoundedBuffer
```

Notes

use while ... wait(); not if ... wait();
because the notified thread does not necessarily succeed when it tries
to acquire the lock
and if some other thread changes the state it may not be appropriate
to continue.

Similarly, use notifyAll() not notify()
because otherwise danger of “lost wakeup”
e.g. two consumers waiting, two items deposited, but only one notify
occurs

stackoverflow.com/questions/37026/#3186336

Example code – class User

```java
class User implements Runnable {
private BoundedBuffer buffer ;
User(BoundedBuffer newBuffer) {
  buffer= newBuffer;
  Thread t= new Thread(this, "Consumer");
t.start();
  try {
    for (int i= 0; i < 30; i++)
      System.out.println("Got " + buffer.extract());
  } catch (InterruptedException e) {
    System.out.println("Producer interrupted");
  }
  }
public static void main(String [] args) {
  BoundedBuffer myBuffer=new BoundedBuffer(8);
  new User(myBuffer);
  }
} // end of class User
```

Question: Possible Output?

Typical (just last digit of each number output):

```
Sent 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4
Got 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4
Sent 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
Got 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
```

Possible?

```
Sent 012345678 9 01 23 4
Got 012345678 90 12 34
Sent 56 7 8 9 01 2 3 4 5 67 8 9
Got 5 6 7 8 90 1 2 3 4 56 7 8 9
```

I saw this once, I don’t know why:

```
Sent 012345678 9 01 23 4
Got 012345678 90 12 34
Sent 56 7 8 9 01 2 3 4 5 67 8 9
Got 5 6 7 8 90 1 2 3 4 56 7 8 9
```

java.util.concurrent

Executors
Queues
TimeUnit
Synchronizers
Concurrent Collections
Memory Consistency

http://download.oracle.com/javase/7/docs/api/java/util/concurrent/package-summary.html
Summary of key points

Creating and running threads
– java.lang.Thread (class)
– java.lang.Runnable (interface)
– run(), start()

Synchronized accesses to shared data
– synchronize methods & blocks
– wait(), notify(), notifyAll()

java.util.concurrent

Next time . . .

Exam Questions

What happens when a synchronised static method is called in Java? (2 marks)

In Java, it is possible to use a synchronised statement as the body of an instance method instead of making the method itself synchronised. Illustrate this with some simple code. (2 marks)

Briefly explain why Java code which waits in a synchronized method for a condition to hold will commonly retest the condition when it has been released from its wait. (2 marks)

Glossary

Thread
start()
run()
sleep()
InterruptedException
Runnable
currentThread()
Lock
synchronized
wait()
notify()
notifyAll()

Reading

Java books/docs: Thread, Runnable, Object.wait(), Object.notify()
http://download.oracle.com/javase/tutorial/essential/concurrency/

OSC/J (6th ed.): Sections 5.7, 7.8 good; 6.8 OK

OSC (7 ed.): Section 4.3.3 (rather condensed) & box on p218