

Part IV

Other Systems: I

Java Threads

C is quirky, flawed, and an enormous success.

1

Dennis M. Ritchie

Java Threads: 1/6

- **Java has two ways to create threads:**
 - ❖ **Create a new class derived from the `Thread` class and overrides its `run ()` method. This is similar to that of ***ThreadMentor***.**
 - ❖ **Define a class that implements the `Runnable` interface.**

Java Threads: 2/6

- **Method #1:** Use the Thread class

```
public class HelloThread extends Thread
{
    public void run()
    {
        System.out.println("Hello World");
    }
    public static void main(String[] args)
    {
        HelloThread t = new HelloThread();
        t.start();
    }
}
```

Java Threads: 3/6

- ***Method #2***: Use the `Runnable` interface defined as follows:

```
public interface Runnable
{
    public abstract void run();
}
```

Java Threads: 4/6

```
class Foo {
    String name;
    public Foo(String s) { name = s; }
    public void setName(String s) { name = s; }
    public String getName() { return name; }
}
class FooBar extends Foo implements Runnable {
    public FooBar(String s) { super(s); }
    public void run() {
        for (int i = 0; i < 10; i++)
            System.out.println(getName()+" : Hello World");
    }
    public static void main(String[] args) {
        FooBar f1 = new FooBar("Romeo");
        Thread t1 = new Thread(f1);    t1.start();
        FooBar f2 = new FooBar("Juliet");
        Thread t2 = new Thread(f2);    t2.start();
    }
}
```

Java Threads: 5/6

```
public class Fibonacci extends Thread {
    int n, result;
    public Fibonacci(int n) { this.n = n; }
    public void run()
    {
        if ((n == 0) || (n == 1))
            result = 1;
        else {
            Fibonacci f1 = new Fibonacci(n-1);
            Fibonacci f2 = new Fibonacci(n-2);
            f1.start(); f2.start();
            try {
                f1.join(); f2.join();
            } catch (InterruptedException e) {};
            result = f1.getResult()+f2.getResult();
        }
    }
    public int getResult() { return result; }
```

Java Threads: 6/6

```
public static void main(String [] args) {  
    Fibonacci f1 =  
        new Fibonacci(Integer.parseInt(args[0]));  
    f1.start();  
    try {  
        f1.join();  
    } catch (InterruptedException e) {};  
    System.out.println("Ans = "+f1.getResult());  
}  
}
```

Part 2/2

The synchronized **Keyword**

- The `synchronized` keyword of a block implements mutual exclusion.

```
public class Counter{  
    private int count = 0;  
    public int inc()  
    {  
        synchronized(this)  
        { return ++count; }  
    }  
}
```

this is a critical section



Java ReentrantLock: 1/2

- A lock provides exclusive access to a shared resource: only one thread at a time can acquire the lock and all access to the shared resource requires that the lock be acquired first.
- A `ReentrantLock` is similar to the `synchronized` keyword.
- You may use `lock()` to acquire a lock and `unlock()` to release a lock.
- There are other methods (e.g., `tryLock()`).

Java ReentrantLock: **2/2**

- The following is a typical use of locks in Java.

```
Lock myLock = new ReentrantLock();

myLock.lock(); // acquire a lock
try {
    // in critical section now
    // catch exceptions and
    // restore invariants if needed
} finally {
    myLock.unlock();
}
```

Java `wait()` **and** `notify()` : **1/7**

- Method `wait()` causes a thread to release the lock it is holding on an object, allowing another thread to run.
- `wait()` should always be wrapped in a `try` block because it throws `IOException`.
- `wait()` can only be invoked by the thread that owns the lock on the object.
- The thread that calls `wait()` becomes inactive until it is notified. Note that actual situation can be more complex than this.

Java `wait()` **and** `notify()` : **2/7**

- A thread uses the `notify()` method of an object to release a waiting thread or the `notifyAll()` method to release all waiting threads.
- After `notify()` or `notifyAll()`, a thread may be picked by the thread scheduler and resumes its execution.
- Then, this thread regains its lock automatically.
- Using `notify()` and `notifyAll()` as the last statement can avoid many potential problems.

Java wait() **and** notify() : 3/7

```
public class Counter implements BoundedCounter {
    protected long count = MIN;
    public synchronized long value() { return count; }
    public synchronized long inc()
        { awaitINC(); setCount(count+1); }
    public synchronized long dec()
        { awaitDEC(); setCount(count-1); }
    protected synchronized void setCount(long newVal)
        { count = newVal; notifyAll(); }
    protected synchronized void awaitINC() {
        while (count >= MAX)
            try { wait(); } catch (InterruptedException e) {};
    }
    protected synchronized void awaitDEC() {
        while (count <= MIN)
            try { wait(); } catch (InterruptedException e) {};
    }
}
```

Java wait() and notify(): 4/7

```
public final class CountingSemaphore {
    private int count = 0;
    public CountingSemaphore(int initVal)
        { count = initVal; }

    public synchronized void P() // semaphore wait
    {
        count--;
        while (count < 0)
            try { wait(); } catch (InterruptedException e) {}
    }

    public synchronized void V() // semaphore signal
    {
        count++;
        notify();
    }
}
```

they are different from our definition
can you see they are equivalent?

why is testing for count <= 0 unnecessary?

Java wait() **and** notify() : 5/7

```
public class Buffer implements BoundedBuffer {
    protected Object[] buffer;
    protected int in;
    protected int out;
    protected int count;
    public Buffer(int size)
        throws IllegalArgumentException {
        if (size <= 0)
            throw new IllegalArgumentException();
        buffer = new Object[size];
    }
    public int GetCount() { return count; }
    public int capacity() { return Buffer.length; }
    // methods put() and get()
}
```

Java wait() **and** notify() : **6/7**

```
public synchronized void put(Object x)
{
    while (count == Buffer.length)
        try { wait(); }
        catch (InterruptedException e) {};
    Buffer[in] = x;
    in = (in + 1) % Buffer.length;
    if (count++ == 0)
        notifyAll();
}
```

Part 2/3

Java wait() **and** notify(): 7/7

```
public synchronized void get(Object x)
{
    while (count == 0)
        try { wait(); }
            catch (InterruptedException e) {};
    Object x = Buffer[out];
    Buffer[out] = null;
    out = (out + 1) % Buffer.length;
    if (count-- == Buffer.length)
        notifyAll();
    return x;
}
```

Part 3/3

The End