CS2833: MODULAR SOFTWARE DEVELOPMENT

MULTITHREADING

Department of Computer Science and Engineering

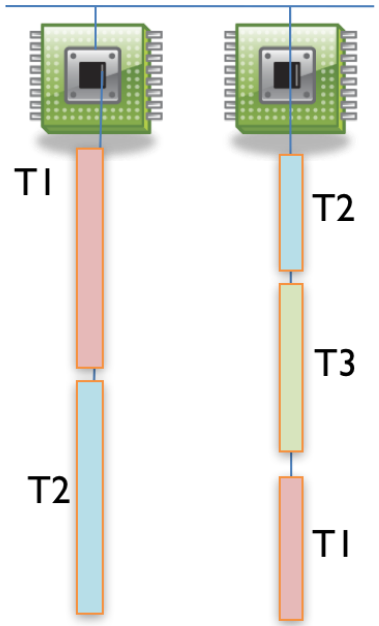
University of Moratuwa

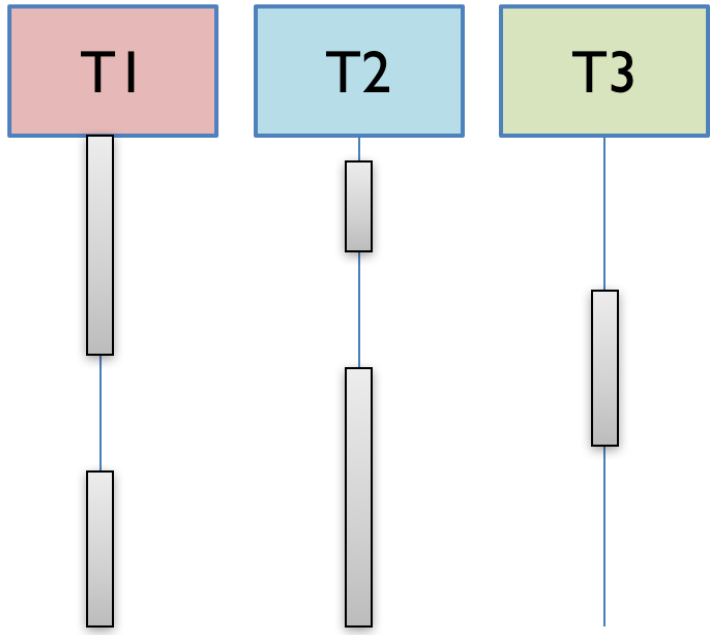
Adapted from slides by Dr Chinthana, Dr. Indika and Ms. Sanduni

WHY?

Concurrency in Java

- Working on more than one job at a time is known as concurrent programming.
- In Java this can be achieved by “Threads”.
- Threads provide execution environments. -> Thus, they require resources.





“Why?” but more complex examples

- Processor time can be maximized if the programs goes into I/O operations by getting other non-I/O programs to run on the idling processor.
- Very advantageous when it comes to network programing where the network operations are usually slow. With concurrent programming, the processor can go do other work while it waits for the network to respond.
- Threads in Java are subprograms of a main application program and share the same memory space. This enables the memory management easier too.

Creating Threads

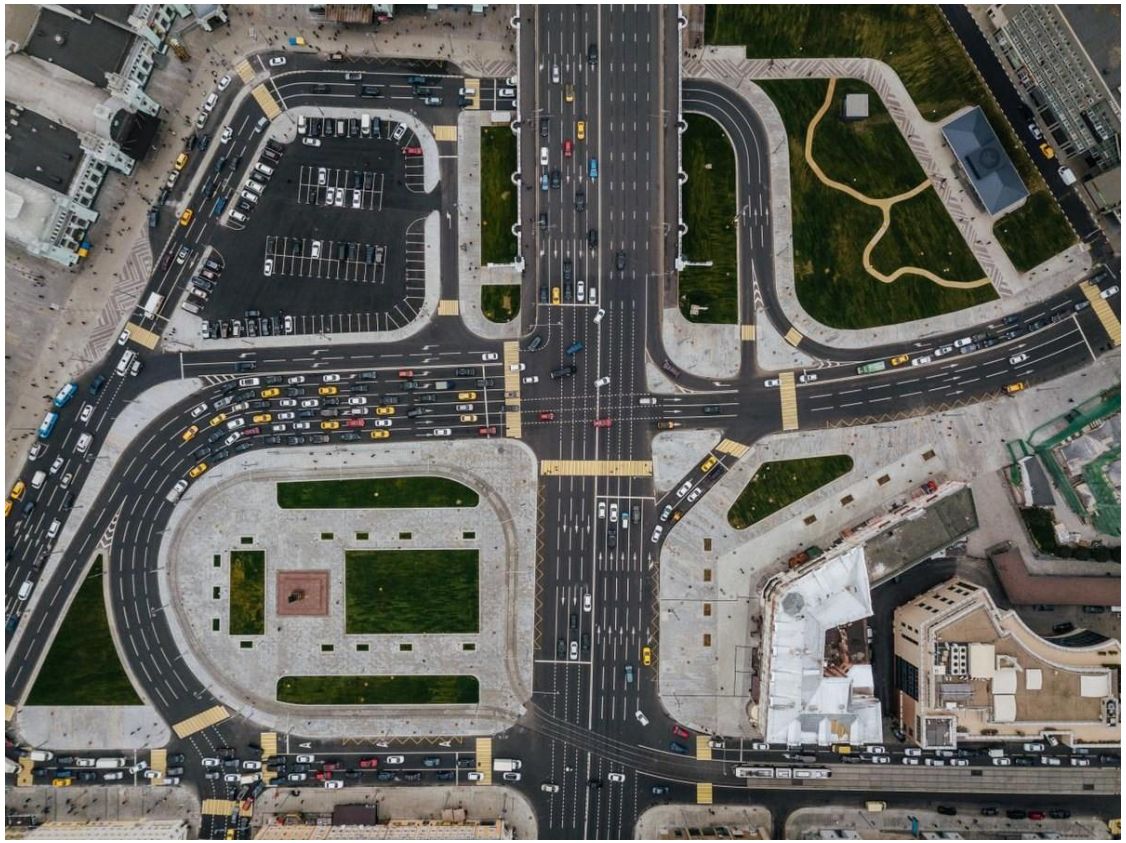
- You need to give the code that will run in a thread.
- In Java this can be done in two main ways,

1. Extending the thread class – This is easier
2. Implementing the runnable interface – This is more general as it still allows for whatever class to be extended

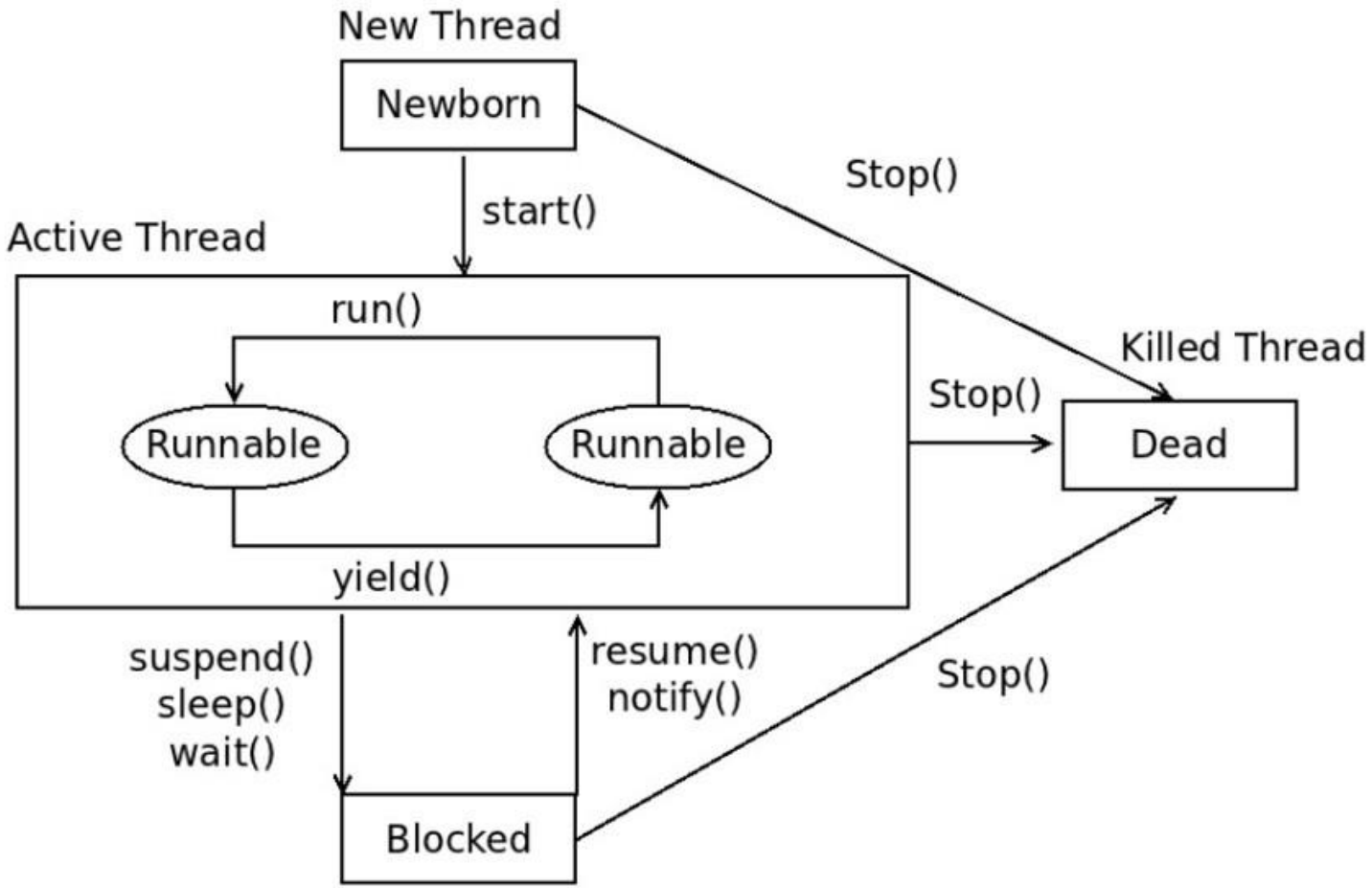
```
package extendthreadclass;

class FirstThreadClass extends Thread {
    public void run() {
        for (int i=0; i<10; i++) {
            System.out.println("FirstThreadClass"+i);
        }
    }
}

class SecondThreadClass extends Thread {
    public void run() {
        for (int i=0; i<10; i++) {
            System.out.println("SecondThreadClass"+i);
        }
    }
}

public class ExtendThreadClass {
    public static void main (String argv[]) {
        FirstThreadClass c1 = new FirstThreadClass();
        SecondThreadClass c2 = new SecondThreadClass();
        c1.start();
        c2.start();
    }
}
```

```
package implementrunnable;

class RunnableThread implements Runnable {
    public void run() {
        for (int i=0; i<10; i++) {
            System.out.println(Thread.currentThread().getName());

        }
    }
}

public class ImplementRunnable {
    public static void main (String argv[]) {
        RunnableThread t = new RunnableThread();
        Thread t1 = new Thread(t,"Thread 1");
        Thread t2 = new Thread(t,"Thread 2");
        t1.start();
        t2.start();
    }
}
```

```
public class ThreadPriorities {
    public static void main (String args[]) {
        B b1 = new B();
        A a1 = new A(b1);
        A a2 = new A(b1);
        A a3 = new A(b1);

        a1.start();
        a2.start();
        a3.start();

        try{
            a1.join();
            a2.join();
            a3.join();
        } catch(InterruptedException e) {
            System.out.println(e);
        } finally {
            System.out.println("Final value of totalNum in b1 is:"
                    + b1.getTotalNum());
        }
    }
}
```



Is that it? Nope!



LIFE CYCLE OF A THREAD

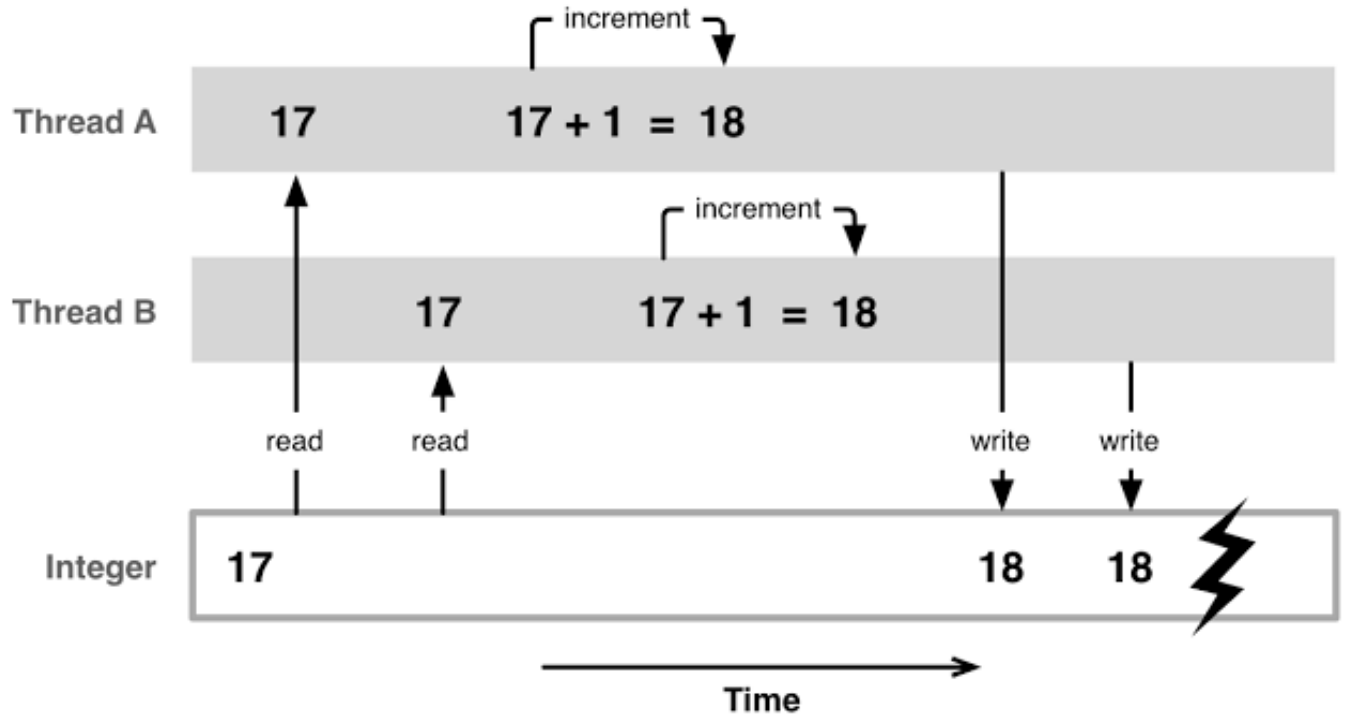
```
public class ThreadPriorities {
    public static void main (String args[]) {
        A threadA = new A();
        B threadB = new B();
        C threadC = new C();
        threadC.setPriority (10);
        threadB.setPriority(1);
        threadA.setPriority(5);
        System.out.println("Start Thread A");
        threadA.start();
        System.out.println("Start Thread B");
        threadB.start();
        System.out.println("Start Thread C");
        threadC.start();
        System.out.println("End of main thread");
    }
}
```

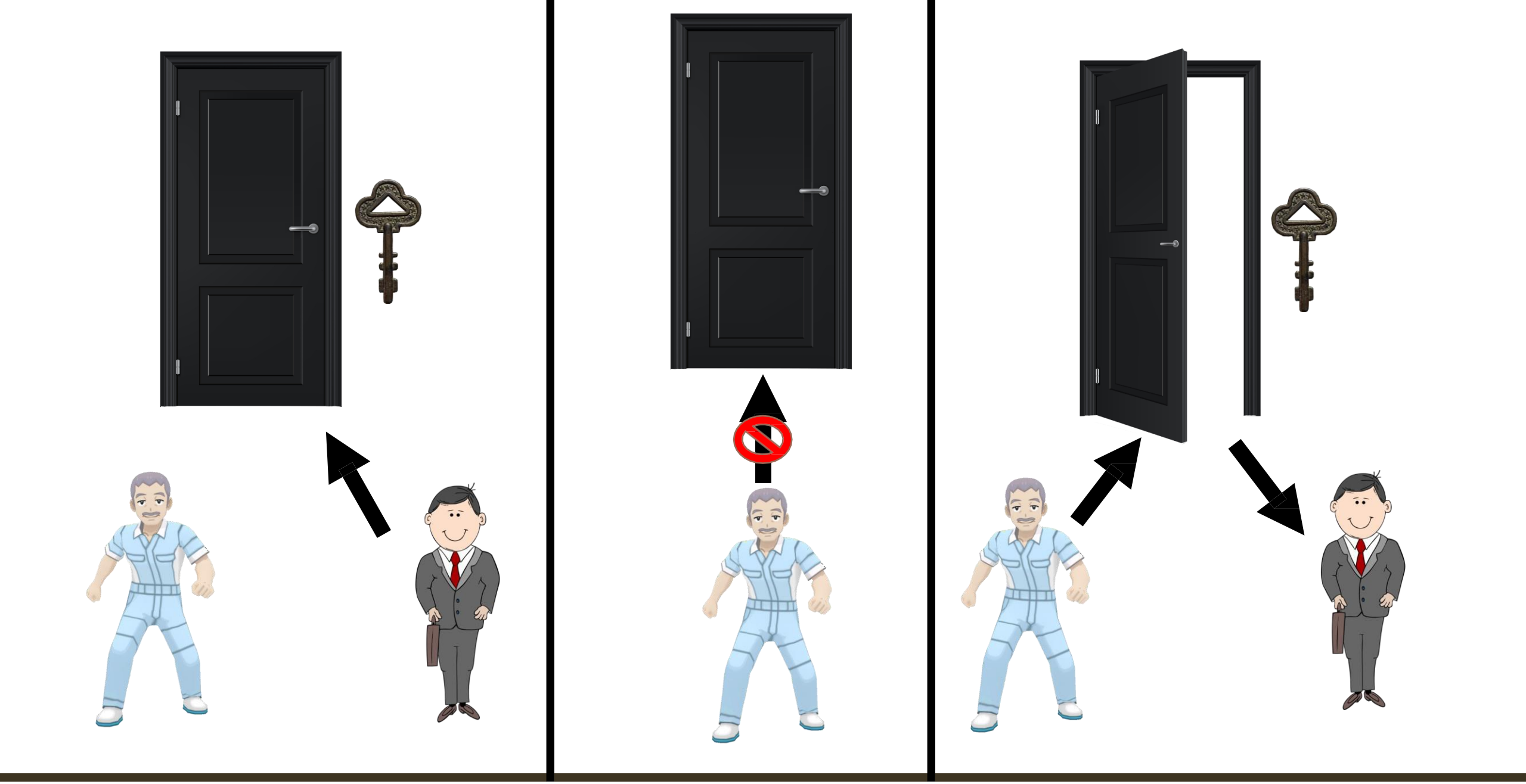
Thread Scheduling

- These will determine the execution order when two or more threads are in runnable state waiting to access the CPU.
- You can set priorities to threads.(But this is not a guarantee that the threads with higher priorities would execute first)

Why Synchronization?

- Race conditions are issues that comes up when there are shared resources (like a variable that is accessed by multiple threads, because even a=a+1 is not atomic)
- You may also want to have a strict order when executing threads.





Synchronization in Java

1. synchronized keyword (method or block)
2. Locks
   - ReentrantLock and may others (ReadWriteLock, StampedLock …)

And may others (Semaphores, AtomicInteger …)

```
class B{
    private int totalNum = 0;

    synchronized void incrementTotal () {
        totalNum+=1;
    }
}
```

```
import java.util.concurrent.locks.ReentrantLock;

class B{
    private int totalNum = 0;
    ReentrantLock lock = new ReentrantLock();

    void incrementTotalWithLock () {
        lock.lock();
        totalNum+=1;
        lock.unlock();
    }
}
```

```
    public void run () {
        synchronized(localB){
            localB.incrementTotal();
            System.out.println("Local B value is: " + localB.getTotalNum()
                    + " from: " + this.getName());
        }
    }
```

Deadlock describes a situation where two or more threads are blocked forever, waiting for each other.

Are we done yet? Nope!

```
package deadlockexample;

public class DeadlockExample {
    static class Friend {
        private final String name;
        public Friend(String name) {
            this.name = name;
        }
        public String getName() {
            return this.name;
        }
        public synchronized void bow(Friend bower) {
            System.out.format("%s: %s has bowed to me!%n",
                    this.name, bower.getName());
            bower.bowBack(this);
        }
        public synchronized void bowBack(Friend bower) {
            System.out.format("%s: %s has bowed back to me!%n",
                    this.name, bower.getName());
        }
    }

    public static void main(String[] args) {
        final Friend alphonse = new Friend("Alphonse");
        final Friend gaston = new Friend("Gaston");
        new Thread(new Runnable() {
            public void run() { alphonse.bow(gaston); }
        }).start();
        new Thread(new Runnable() {
            public void run() { gaston.bow(alphonse); }
        }).start();
    }
}
```

THANK YOU!

IF YOU HAVE ANY QUESTIONS, PLEASE BRING THEM UP IN THE FORUMS!