Threading in Java

Introduction

- Motivation for concurrency
- Package: java.util.concurrent
- The Multis
  - ➢ Multiprogramming
  - ➢ Multiprocessing
  - ➢ Multithreading

Processes and Threads

- Process (program)
  - ➢ Private resources
  - ➢ To communicate between processes, InterProcess (IPC) Communication is used(*)
  - ➢ JVM running a java application is a single process
  - ➢ Has at least 1 thread (main thread)
  - ➢ A feature of the OS
- Thread (lightweight process)
  - ➢ Shared resources (memory and open files)
  - ➢ A feature of the Java platform

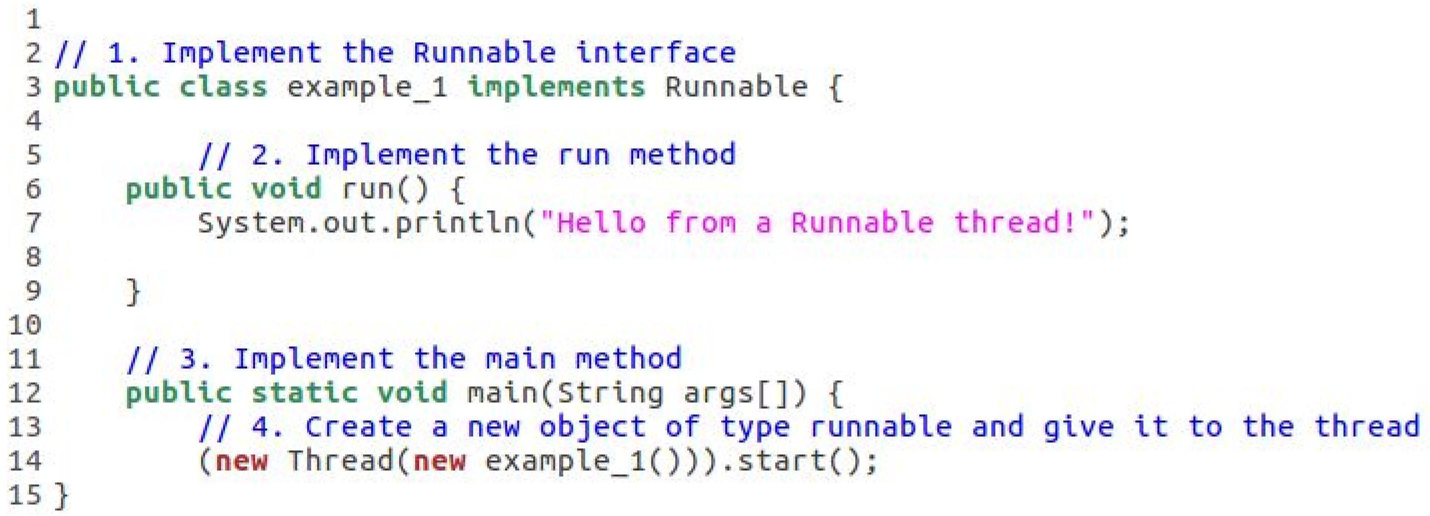
(*) Will learn about this in an OS course

Thread objects (Creation)

- Each thread is an object of class Thread
- To create a thread, you need:
  - ➢ A thread object (or one of its subclasses)
  - ➢ Implement the core of the thread (implement/override the run() method)
  - ➢ Call start() on the thread object which invokes the run
- Define and start thread:
  - ➢ Implement the Runnable interface, and provide a runnable object (example_1.java)
  - ➢ Extend a Thread class (example_2.java)

Thread objects (Creation)

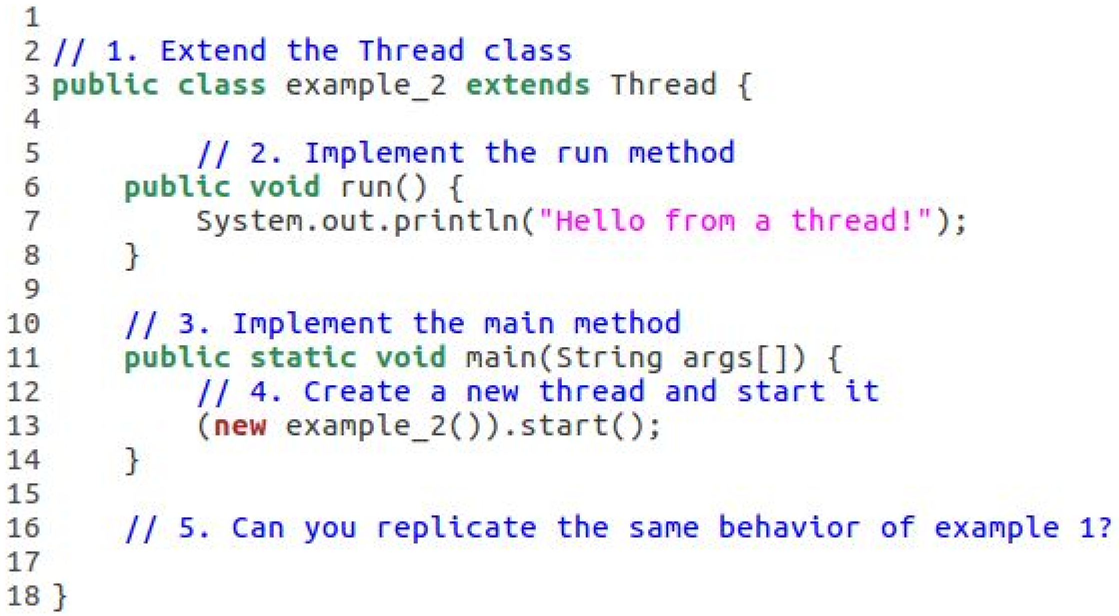
- Implement the Runnable interface, and provide a runnable object



```java

// 1. Implement the Runnable interface
public class example_1 implements Runnable {

        // 2. Implement the run method
    public void run() {
        System.out.println("Hello from a Runnable thread!");

    }

    // 3. Implement the main method
    public static void main(String args[]) {
        // 4. Create a new object of type runnable and give it to the thread
        (new Thread(new example_1())).start();
}
```

Thread objects (Creation)

- Implementation by Extending Thread class



```java

// 1. Extend the Thread class
public class example_2 extends Thread {

        // 2. Implement the run method
    public void run() {
        System.out.println("Hello from a thread!");
    }

    // 3. Implement the main method
    public static void main(String args[]) {
        // 4. Create a new thread and start it
        (new example_2()).start();
    }

    // 5. Can you replicate the same behavior of example 1?

}
```

Thread objects (Creation)

Think:

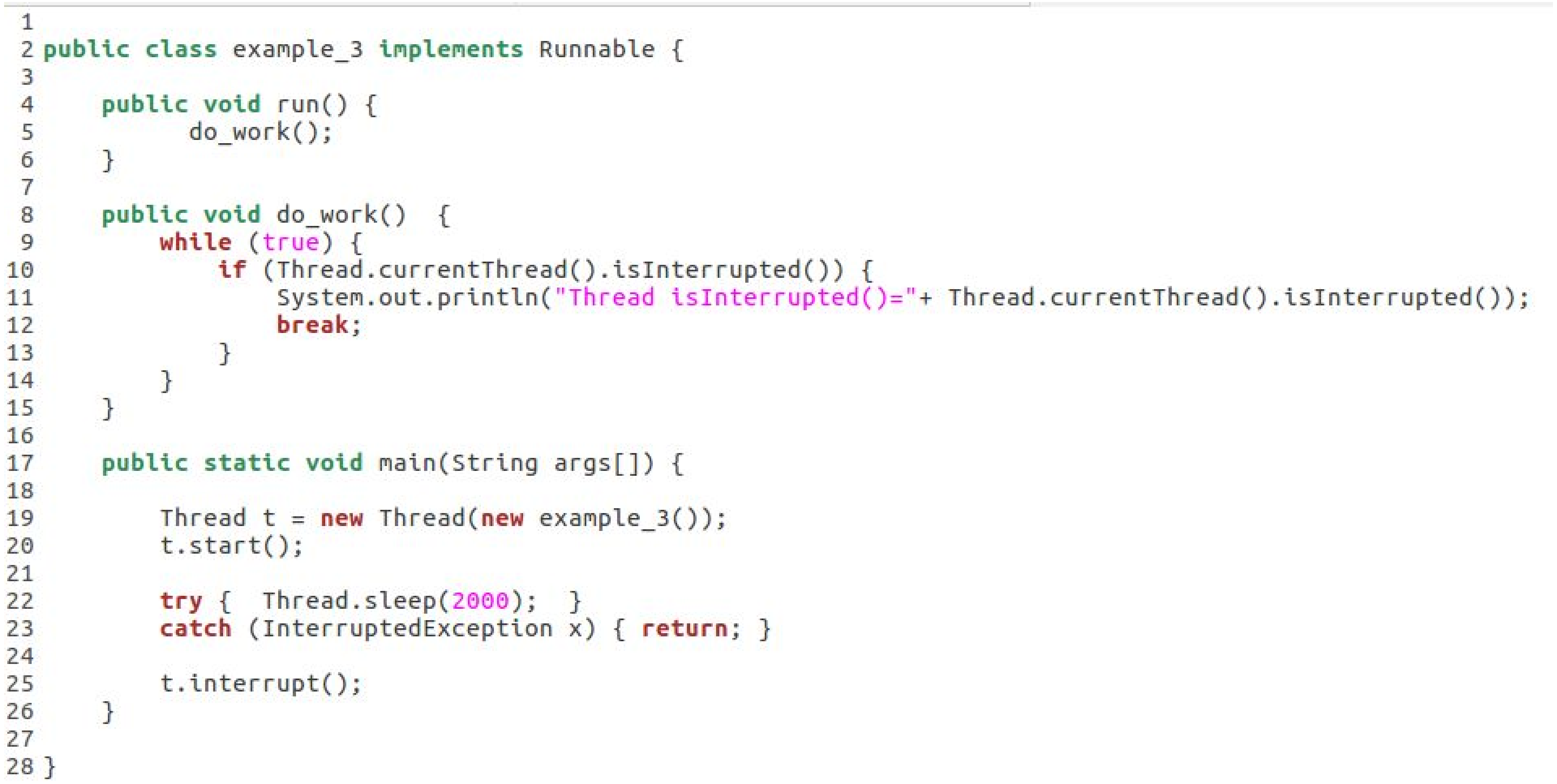
a. Replicate the behavior of example_1 on example_2, does your program run in parallel?

b. Use run instead of start, what happens? Why?

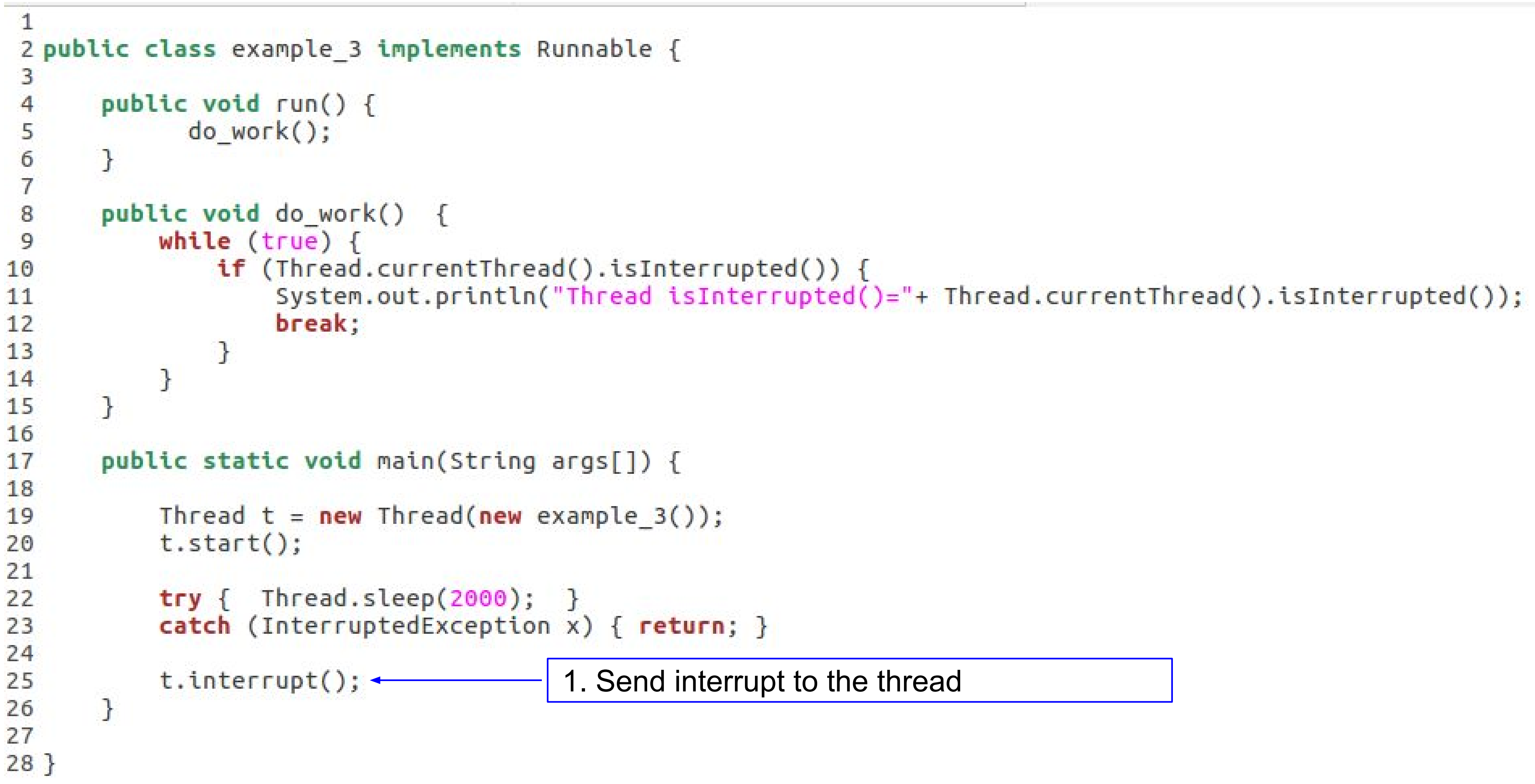
c. What will happen when the main thread terminates?

Thread Objects (Interrupts)

- What’s an thread interrupt?
- To interrupt a thread call interrupt on the object
- To check if the current thread is interrupted use: Thread.currentThread().isInterrupted()
- To handle the interrupt, let the receiving method throw a InterruptedException
- Catch the exception and do the required handling
- Exercise: example_3.java

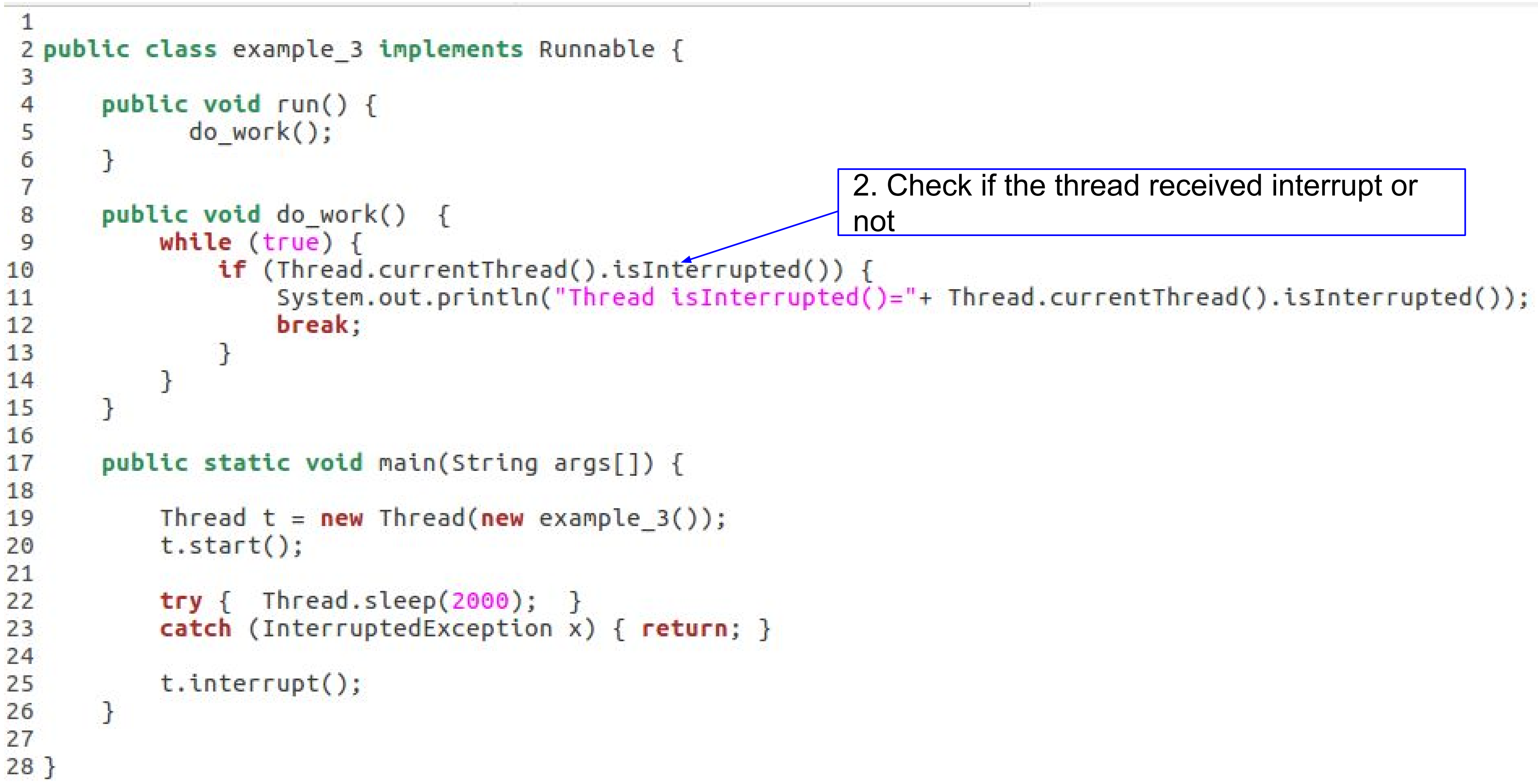


```java

public class example_3 implements Runnable {

    public void run() {
          do_work();
    }

    public void do_work()  {
        while (true) {
            if (Thread.currentThread().isInterrupted()) {
                System.out.println("Thread isInterrupted()="+ Thread.currentThread().isInterrupted());
                break;
            }
        }
    }

    public static void main(String args[]) {

        Thread t = new Thread(new example_3());
        t.start();

        try {  Thread.sleep(2000);  }
        catch (InterruptedException x) { return; }

        t.interrupt();
    }

}
```



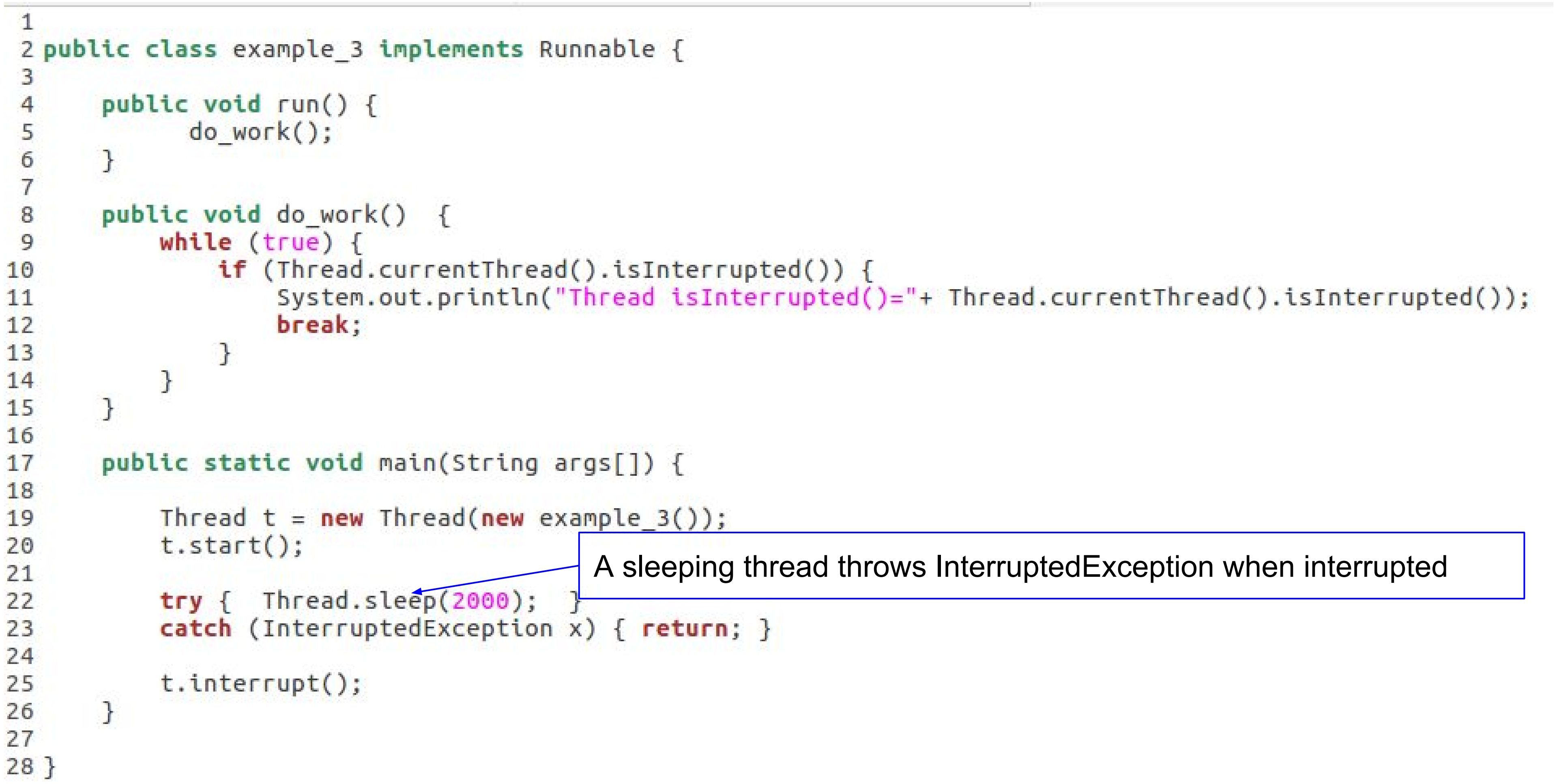
```java

public class example_3 implements Runnable {

    public void run() {
        do_work();
    }

    public void do_work()  {
        while (true) {
            if (Thread.currentThread().isInterrupted()) {
                System.out.println("Thread isInterrupted()="+ Thread.currentThread().isInterrupted());
                break;
            }
        }
    }

    public static void main(String args[]) {

        Thread t = new Thread(new example_3());
        t.start();

        try {  Thread.sleep(2000);  }
        catch (InterruptedException x) { return; }

        t.interrupt();
    }

}
```

1. Send interrupt to the thread



```java

public class example_3 implements Runnable {

    public void run() {
        do_work();
    }

    public void do_work()  {
        while (true) {
            if (Thread.currentThread().isInterrupted()) {
                System.out.println("Thread isInterrupted()="+ Thread.currentThread().isInterrupted());
                break;
            }
        }
    }

    public static void main(String args[]) {

        Thread t = new Thread(new example_3());
        t.start();

        try {  Thread.sleep(2000);  }
        catch (InterruptedException x) { return; }

        t.interrupt();
    }

}
```

2. Check if the thread received interrupt or not



```java

public class example_3 implements Runnable {

    public void run() {
          do_work();
    }

    public void do_work()  {
        while (true) {
            if (Thread.currentThread().isInterrupted()) {
                System.out.println("Thread isInterrupted()="+ Thread.currentThread().isInterrupted());
                break;
            }
        }
    }

    public static void main(String args[]) {

        Thread t = new Thread(new example_3());
        t.start();

        try {  Thread.sleep(2000);  }
        catch (InterruptedException x) { return; }

        t.interrupt();
    }

}
```

A sleeping thread throws InterruptedException when interrupted

Thread Objects (Join)

- In example_1.java, both main thread and new thread were printing together, can we postpone the execution of the main until all threads finish?
- The join method allows one thread to wait for the completion of another.

Exercise: example_4.java

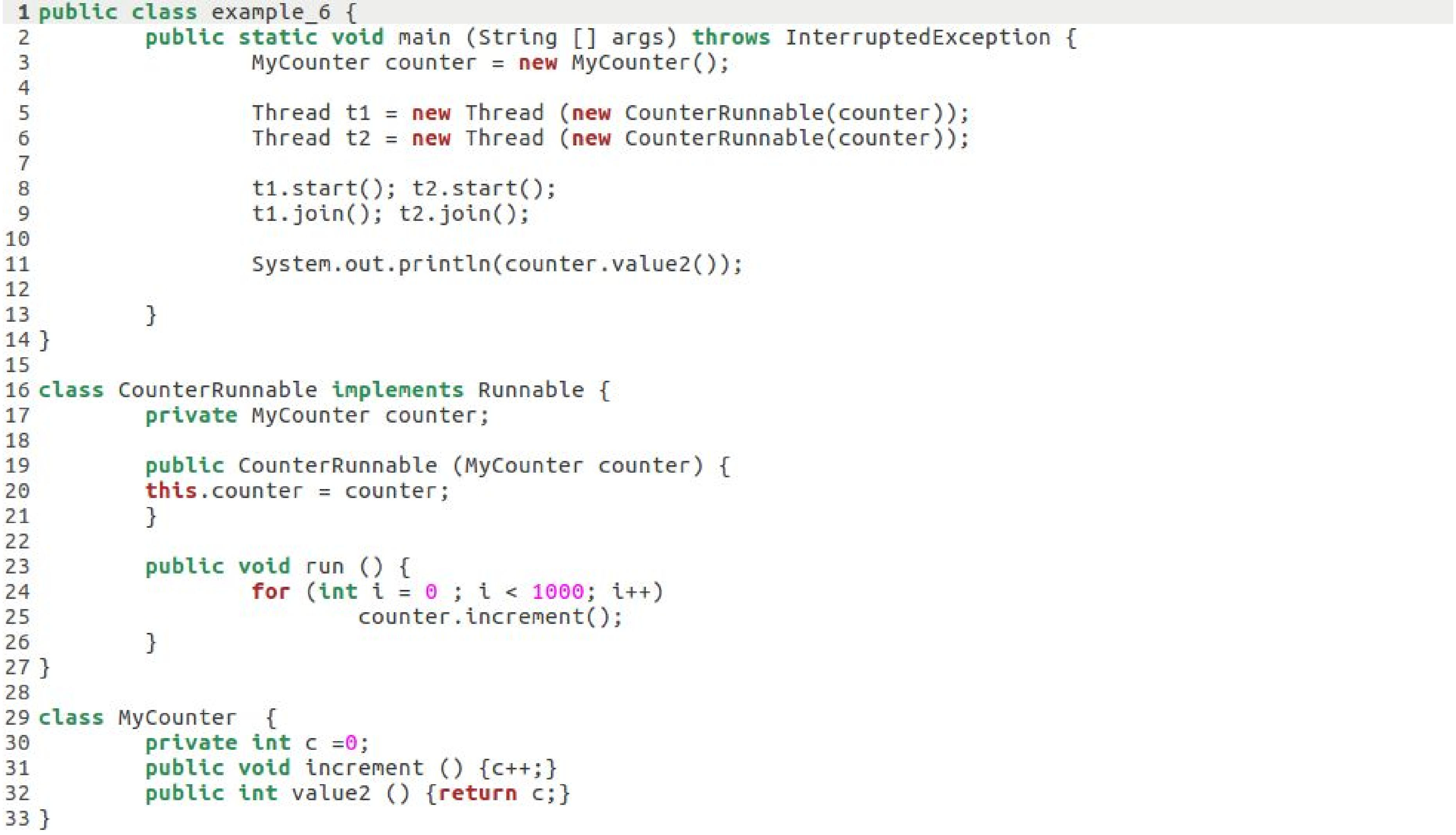
- Create 2 threads, set a name for each, and set a sleeping period for each based on the thread id
- Start threads
- Wait for them to join the main thread

Check example_5.java, it sets the priority of a thread to either normal(5) or max(10), and changes the core of the run() method to do different things based on the threads priority. Anything interesting?

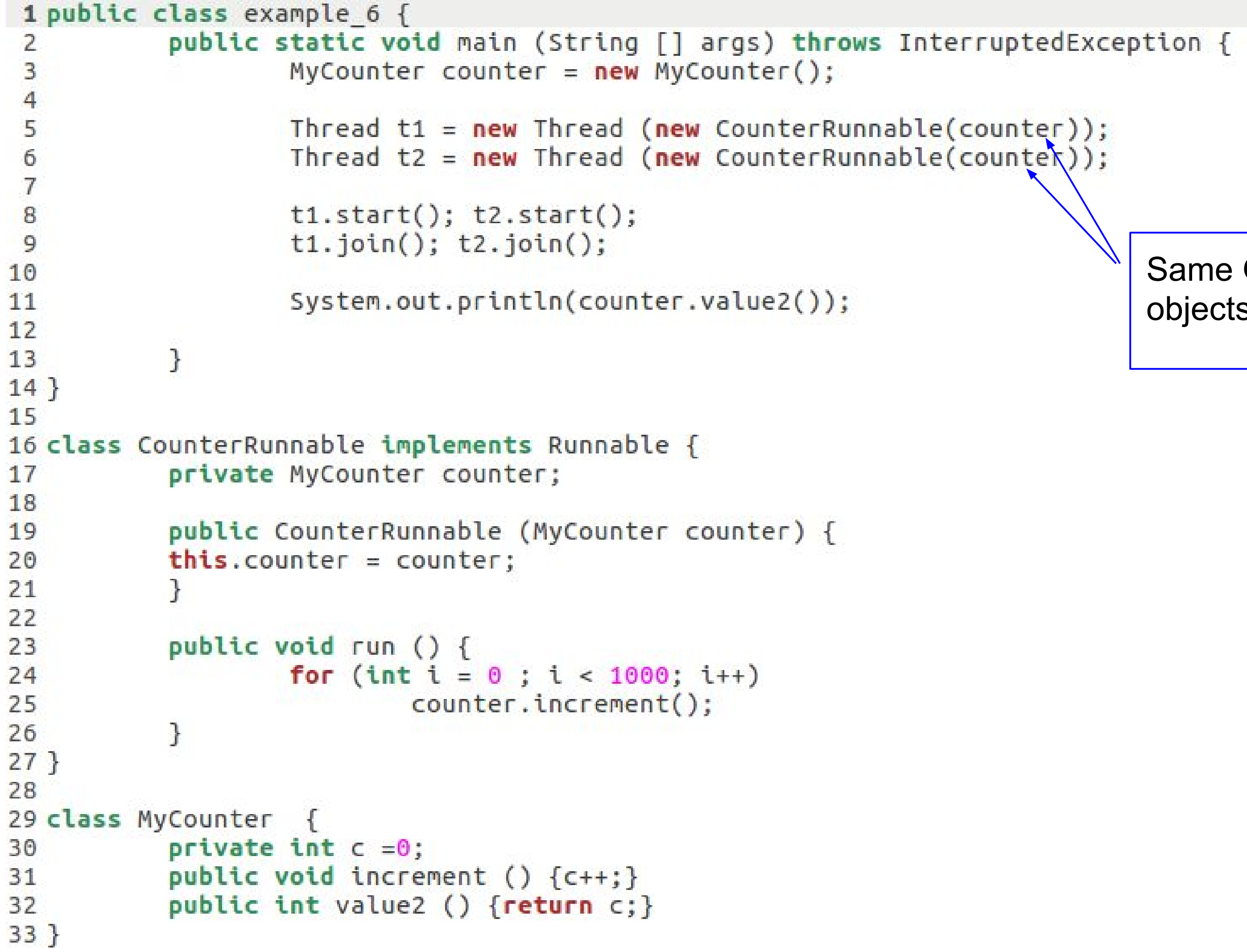
```
    public void run () {
            int prio = Thread.currentThread().getPriority();
            System.out.println("Hello from "+ Thread.currentThread().getName());
            System.out.println("Priority " + prio);


            // if prio = 0 Print 0 ->4 elements else print 5->10
            for (int i = prio ; i< prio+5; i++ )
                    System.out.println("Thread "+ Thread.currentThread().getName()+": "+i);
    }

     // 3. Implement the main method
public static void main(String args[]) throws Exception {

    // Get a handler on the current main thread
    Thread t0 = Thread.currentThread();

    // 4. Create 2 threads
    Thread t1 = new Thread(new example_5());
    Thread t2 = new Thread(new example_5());

    // 5. Set the name of each thread, and optional their priorities
    t0.setName("Main Thread");

    // Thread.MAX_PRIORITY = 10
    // Thread.NORM_PRIORITY = 5
    // Thread.MIN_PRIORITY = 1
    t1.setName("Thread 1"); t1.setPriority(Thread.MIN_PRIORITY);
    t2.setName("Thread 2"); t1.setPriority(Thread.MAX_PRIORITY);

    // 6. Start threads (t0 already running)
    t1.start();
    t2.start();

    // 7. Wait for them to join the current thread
    t1.join();
    t2.join();

    System.out.println("All printed");
}
```

Synchronization

- Why we need synchronization (run example_6.java)?



```java
public class example_6 {
        public static void main (String [] args) throws InterruptedException {
                MyCounter counter = new MyCounter();

                Thread t1 = new Thread (new CounterRunnable(counter));
                Thread t2 = new Thread (new CounterRunnable(counter));

                t1.start(); t2.start();
                t1.join(); t2.join();

                System.out.println(counter.value2());

        }
}

class CounterRunnable implements Runnable {
        private MyCounter counter;

        public CounterRunnable (MyCounter counter) {
        this.counter = counter;
        }

        public void run () {
                for (int i = 0 ; i < 1000; i++)
                        counter.increment();
        }
}

class MyCounter  {
        private int c =0;
        public void increment () {c++;}
        public int value2 () {return c;}
}
```



```java
public class example_6 {
    public static void main (String [] args) throws InterruptedException {
        MyCounter counter = new MyCounter();

        Thread t1 = new Thread (new CounterRunnable(counter));
        Thread t2 = new Thread (new CounterRunnable(counter));

        t1.start(); t2.start();
        t1.join(); t2.join();

        System.out.println(counter.value2());

    }
}

class CounterRunnable implements Runnable {
    private MyCounter counter;

    public CounterRunnable (MyCounter counter) {
    this.counter = counter;
    }

    public void run () {
        for (int i = 0 ; i < 1000; i++)
            counter.increment();
    }
}

class MyCounter  {
    private int c =0;
    public void increment () {c++;}
    public int value2 () {return c;}
}
```

Same Object, remember objects are sent by reference

Synchronization

- Why we need synchronization (example_6.java)?
- The “happens before” relationship
  - ➢ Pros: Maintaining memory consistency
  - ➢ Cons: Thread contention
- Synchronization using:
  - ➢ Locks
    - Intrinsic
    - Extrinsic
  - ➢ Atomic operations

Add synchronized to example_6.java (Is that what we want?)

Intrinsic Locks

- Implements the monitor construct to enforce mutual exclusion (mutex+condition)
- An object has an associated intrinsic lock
- A thread needs to acquire the lock before accessing this object’s fields, and releases it when done (happens before relation is established)
- All other threads accessing this object block when the lock is not available (held by another thread)
- Lock is released on return from the synchronized block, even if it was caused by an exception
- What about static synchronized methods?

Synchronized keyword

- Synchronized Methods:

  Adding the keyword synchronized to a method synchronizes access to the object containing this method:

  ➢ Prevents threads from interleaving execution on this portion.

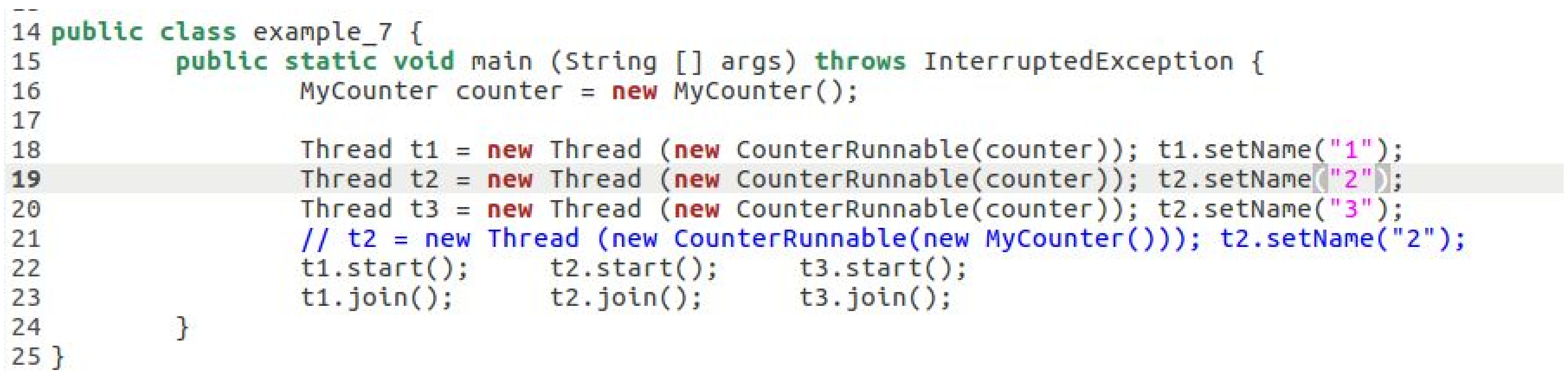
  ➢ Establishes a happens before relation with any subsequent invocation of the same method.

  ➢ Constructors cannot be synchronized (syntax error)

  If an object is shared between threads, all reads/writes to this object should be synchronized (What about final?)

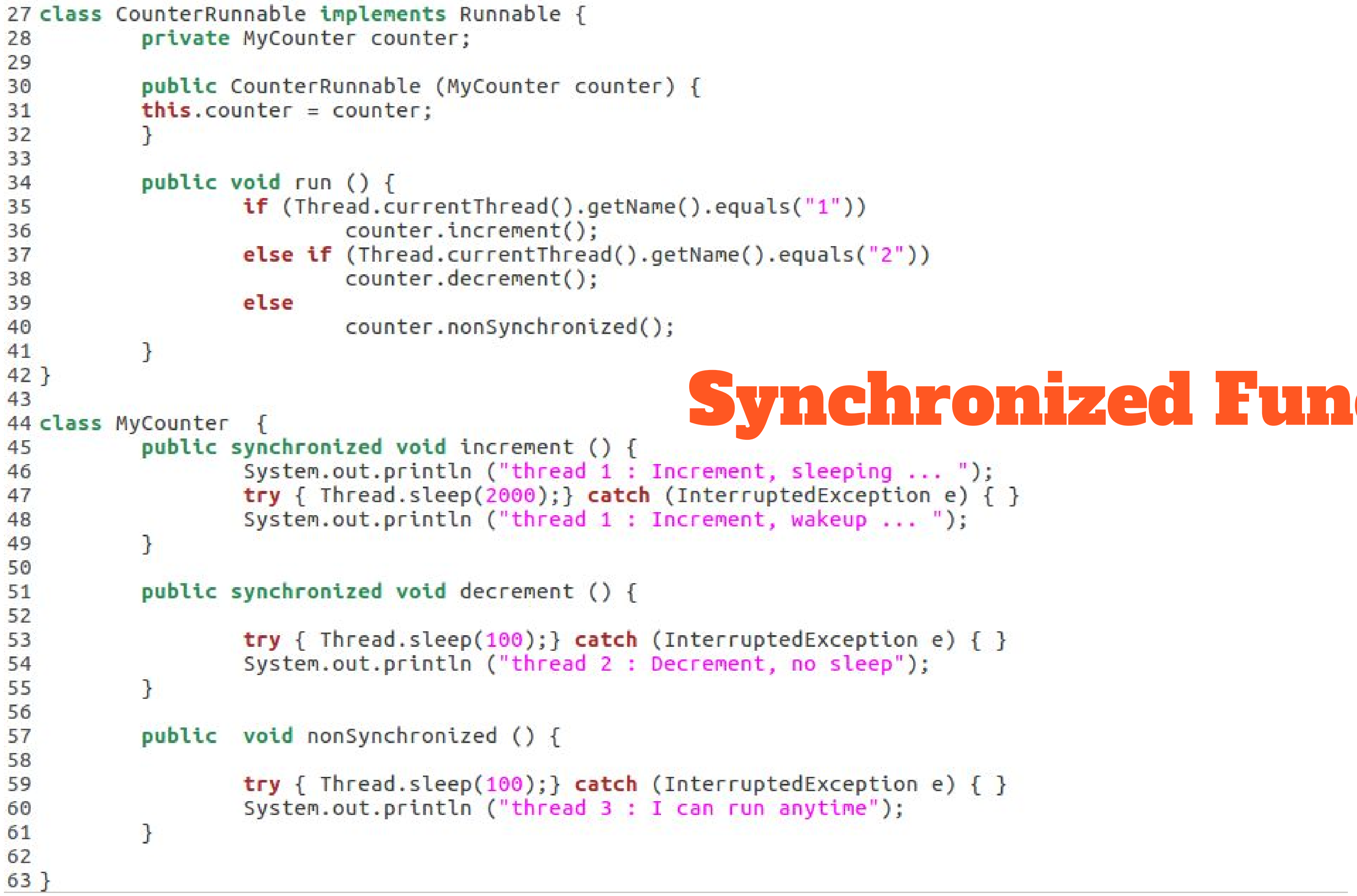
- Example_7.java, synchronizing 2 methods.

Synchronized keyword



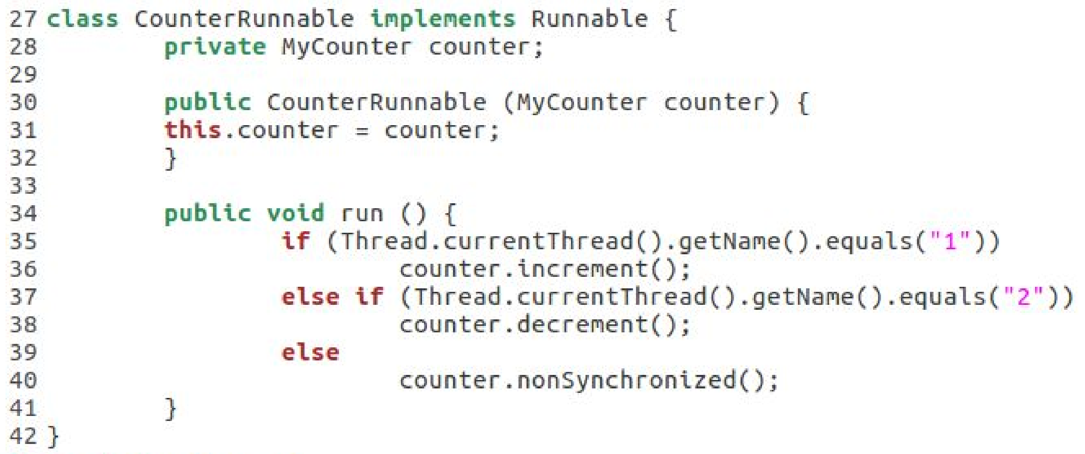
```
public class example_7 {
        public static void main (String [] args) throws InterruptedException {
                MyCounter counter = new MyCounter();

                Thread t1 = new Thread (new CounterRunnable(counter)); t1.setName("1");
                Thread t2 = new Thread (new CounterRunnable(counter)); t2.setName("2");
                Thread t3 = new Thread (new CounterRunnable(counter)); t2.setName("3");
                // t2 = new Thread (new CounterRunnable(new MyCounter())); t2.setName("2");
                t1.start();     t2.start();     t3.start();
                t1.join();      t2.join();      t3.join();
        }
}
```

Synchronized Function

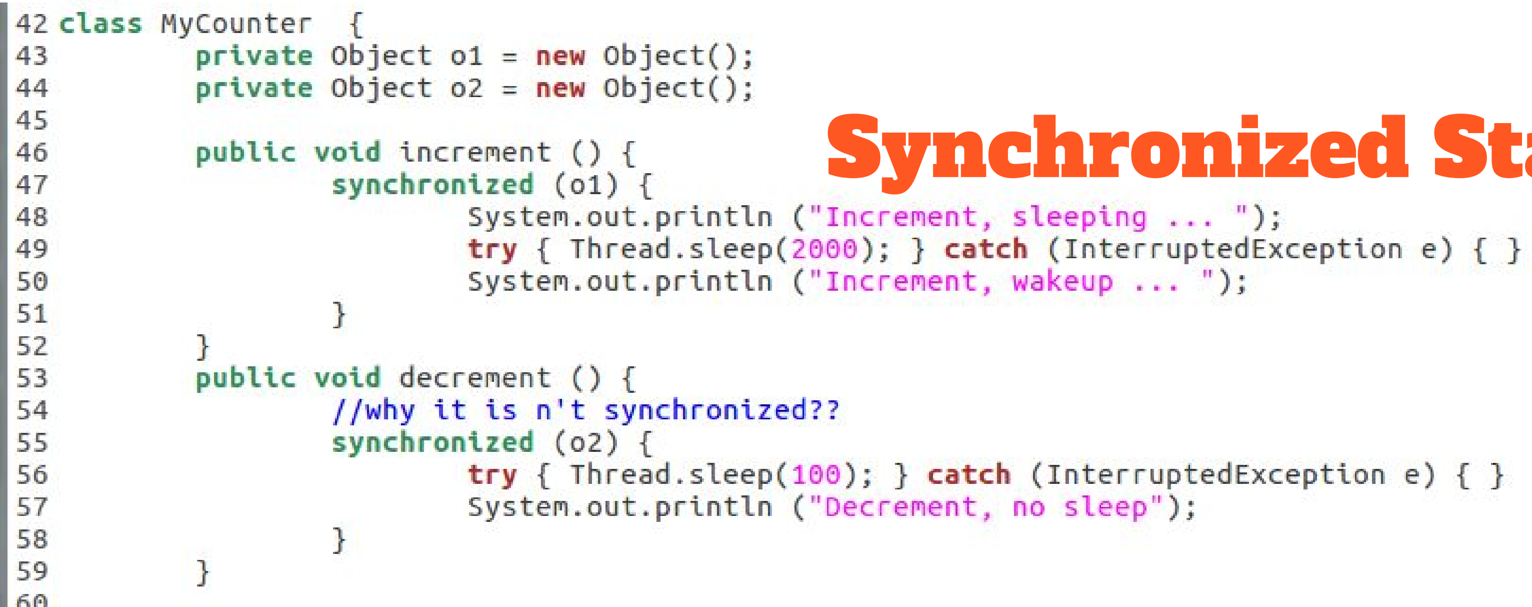


```java
class CounterRunnable implements Runnable {
        private MyCounter counter;

        public CounterRunnable (MyCounter counter) {
        this.counter = counter;
        }

        public void run () {
                if (Thread.currentThread().getName().equals("1"))
                        counter.increment();
                else if (Thread.currentThread().getName().equals("2"))
                        counter.decrement();
                else
                        counter.nonSynchronized();
        }
}

class MyCounter  {
        public synchronized void increment () {
                System.out.println ("thread 1 : Increment, sleeping ... ");
                try { Thread.sleep(2000);} catch (InterruptedException e) { }
                System.out.println ("thread 1 : Increment, wakeup ... ");
        }

        public synchronized void decrement () {

                try { Thread.sleep(100);} catch (InterruptedException e) { }
                System.out.println ("thread 2 : Decrement, no sleep");
        }

        public  void nonSynchronized () {

                try { Thread.sleep(100);} catch (InterruptedException e) { }
                System.out.println ("thread 3 : I can run anytime");
        }

}
```

Synchronized Statement



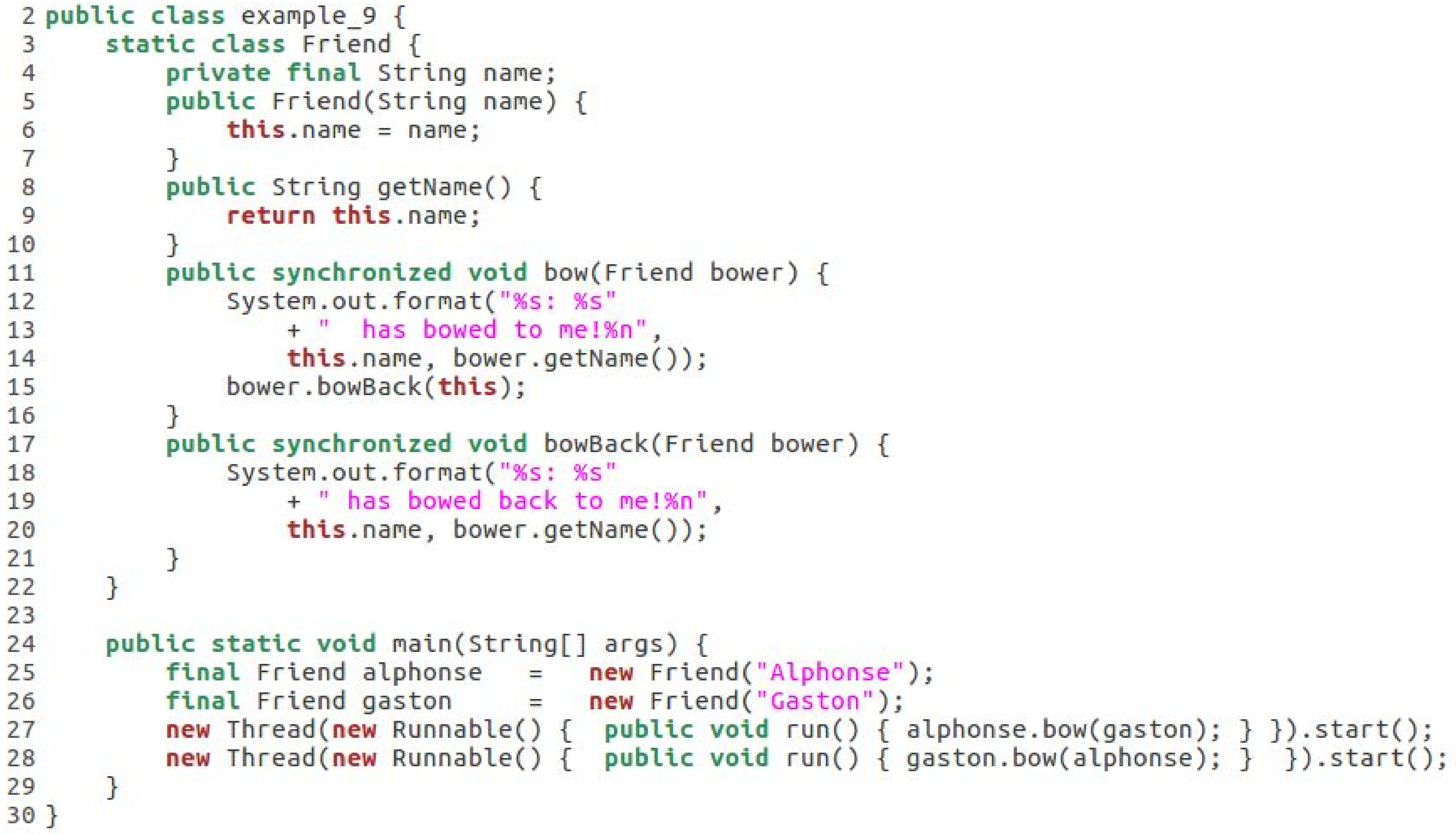
```java
class CounterRunnable implements Runnable {
        private MyCounter counter;

        public CounterRunnable (MyCounter counter) {
        this.counter = counter;
        }

        public void run () {
                if (Thread.currentThread().getName().equals("1"))
                        counter.increment();
                else if (Thread.currentThread().getName().equals("2"))
                        counter.decrement();
                else
                        counter.nonSynchronized();
        }
}
```



```java
class MyCounter  {
        private Object o1 = new Object();
        private Object o2 = new Object();

        public void increment () {
                synchronized (o1) {
                        System.out.println ("Increment, sleeping ... ");
                        try { Thread.sleep(2000); } catch (InterruptedException e) { }
                        System.out.println ("Increment, wakeup ... ");
                }
        }
        public void decrement () {
                //why it is n't synchronized??
                synchronized (o2) {
                        try { Thread.sleep(100); } catch (InterruptedException e) { }
                        System.out.println ("Decrement, no sleep");
                }
        }
```

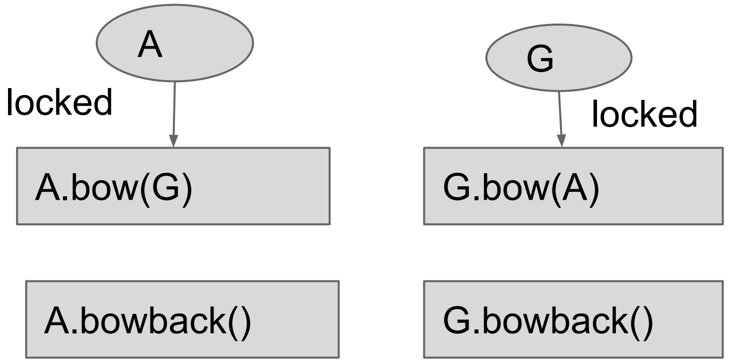
Synchronized keyword

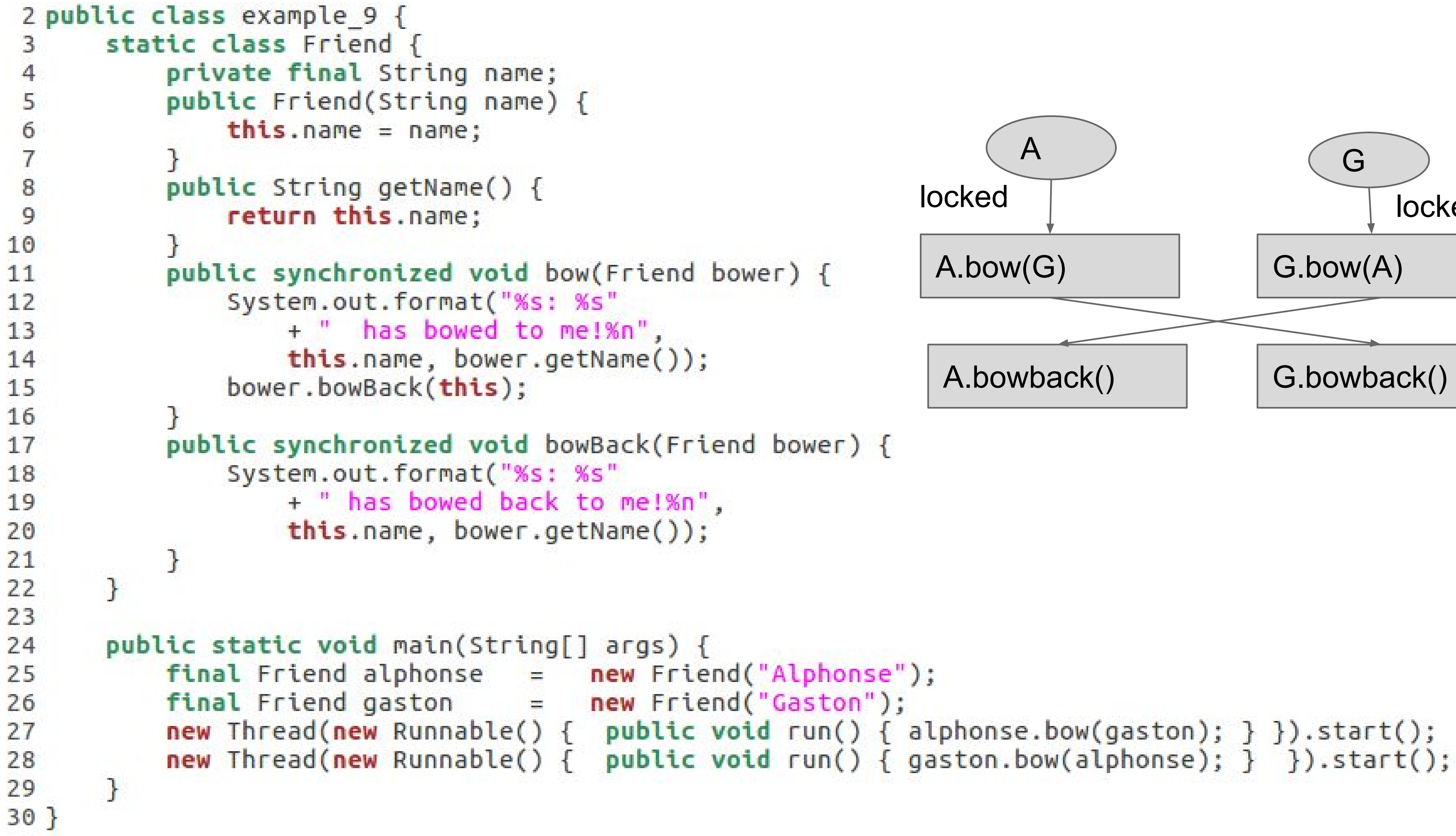
- Synchronization may lead to a deadlock :
  - ➢ Example_9.java



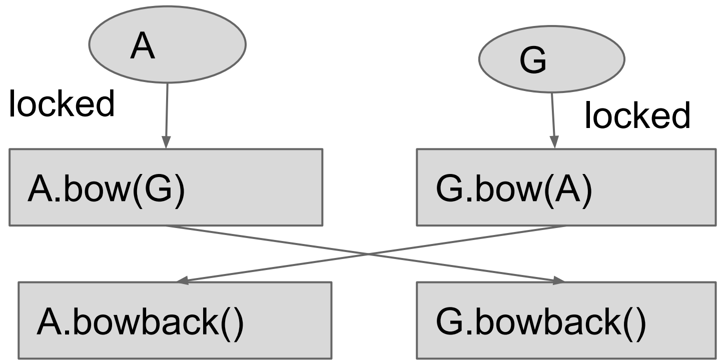
```java
public class example_9 {
    static class Friend {
        private final String name;
        public Friend(String name) {
            this.name = name;
        }
        public String getName() {
            return this.name;
        }
        public synchronized void bow(Friend bower) {
            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();
    }
}
```

```java
public class example_9 {
    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();
    }
}
```

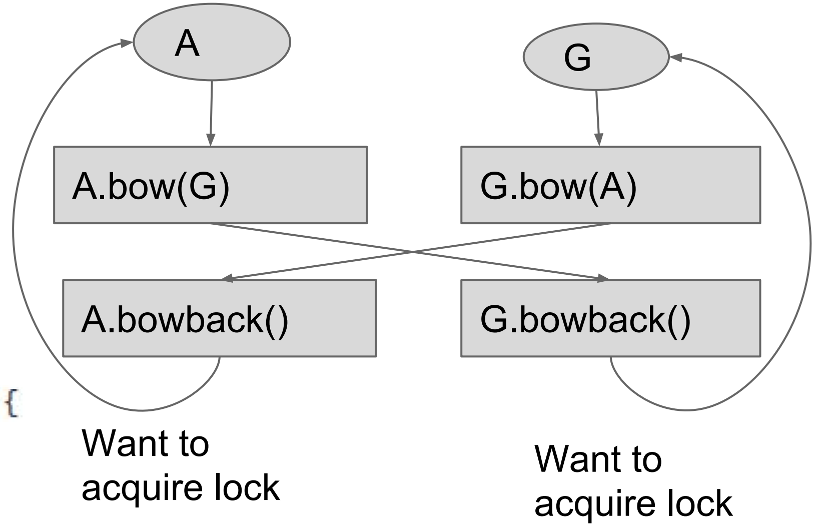




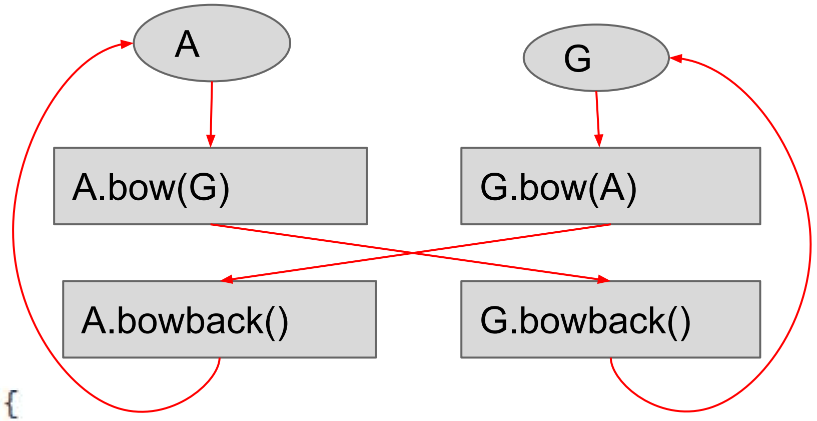
```
public class example_9 {
    static class Friend {
        private final String name;
        public Friend(String name) {
            this.name = name;
        }
        public String getName() {
            return this.name;
        }
        public synchronized void bow(Friend bower) {
            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();
    }
}
```



```java
public class example_9 {
    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();
    }
}
```



```java
public class example_9 {
    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();
    }
}
```



Deadlock

Atomic Operations

- All or none, why would that be needed?
- Compare atomic to synchronized operations:
  - ➢ example_10.java

Guarded Blocks

- Some threads may depend on others conditionally

- To make a thread wait for a condition to be true use wait()

➔ wait() only used with synchronized statement/functions

- To let the other threads know that notifyAll()

- Example_11.java

Summary

- Thread is implemented by Extending Thread Class or implementing Runnable Interface
- run() vs start()
- sleep() / interrupt()
- wait() / notify()
- join()
- Synchronized Function vs Synchronized Object