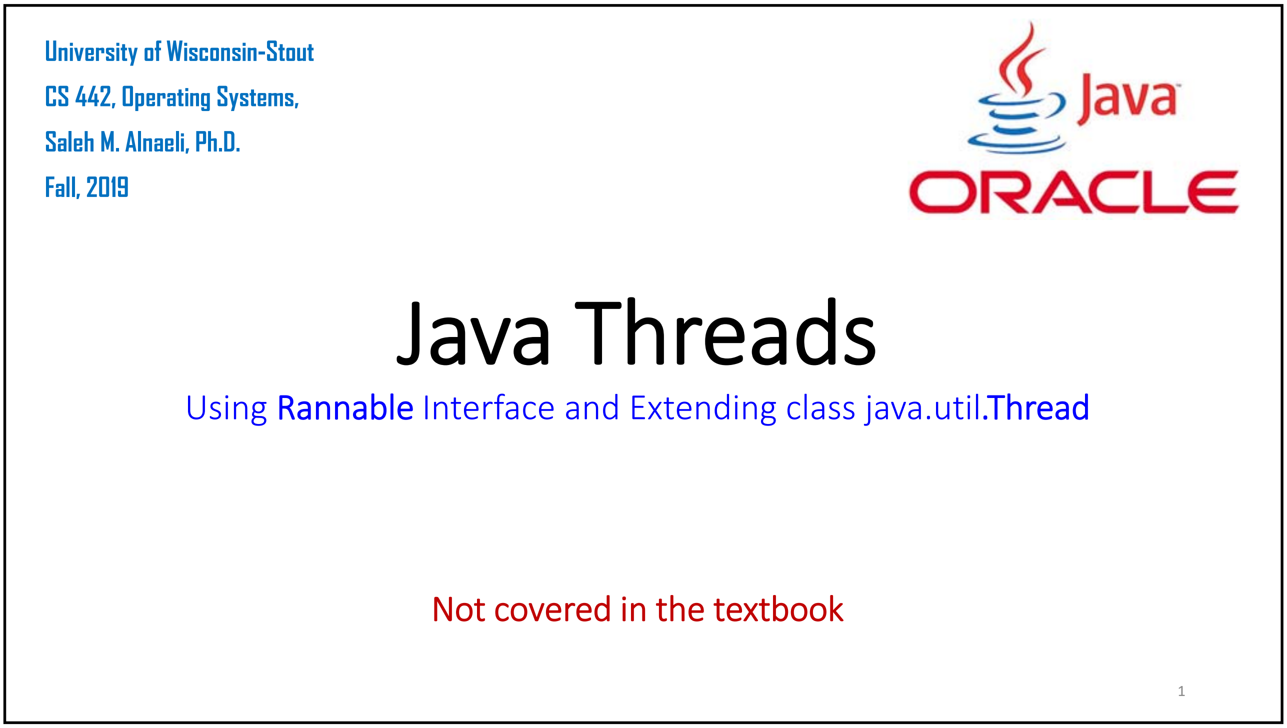


University of Wisconsin-Stout

CS 442, Operating Systems,

Saleh M. Alnaeli, Ph.D.

Fall, 2019

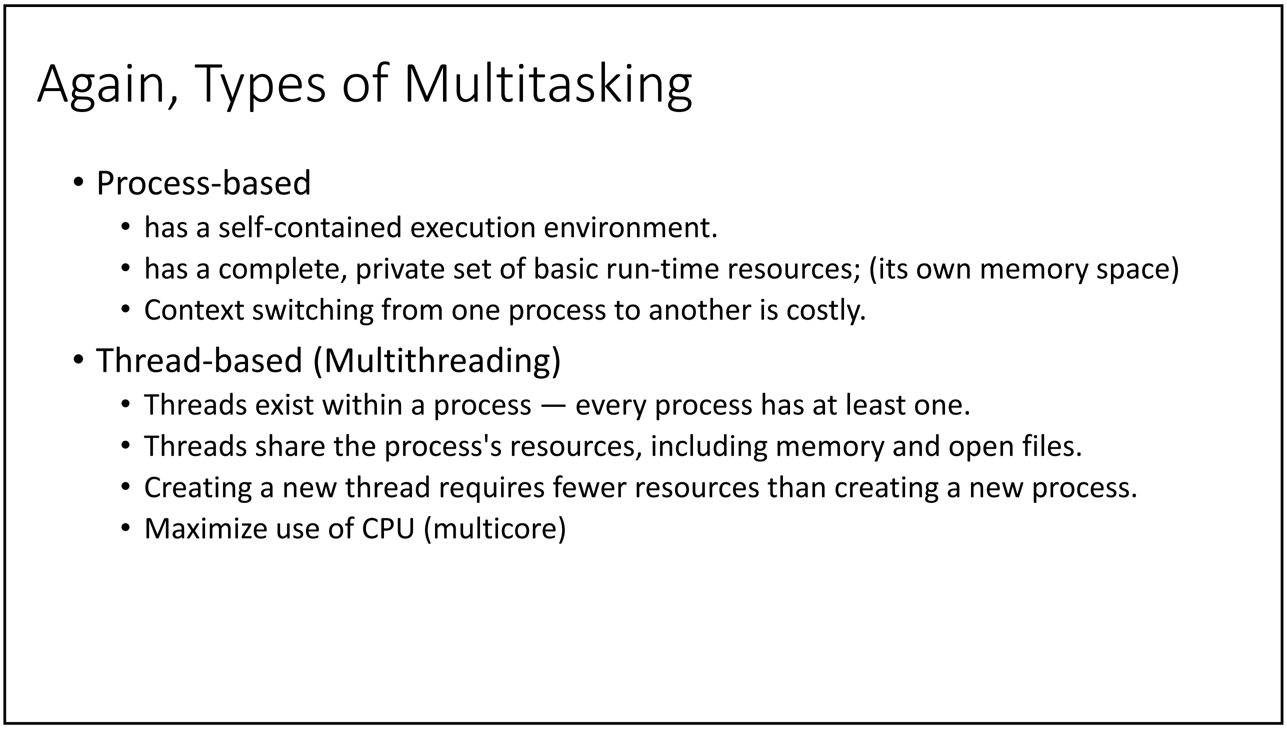
Java Threads

Using Rannable Interface and Extending class java.util.Thread

Not covered in the textbook

1

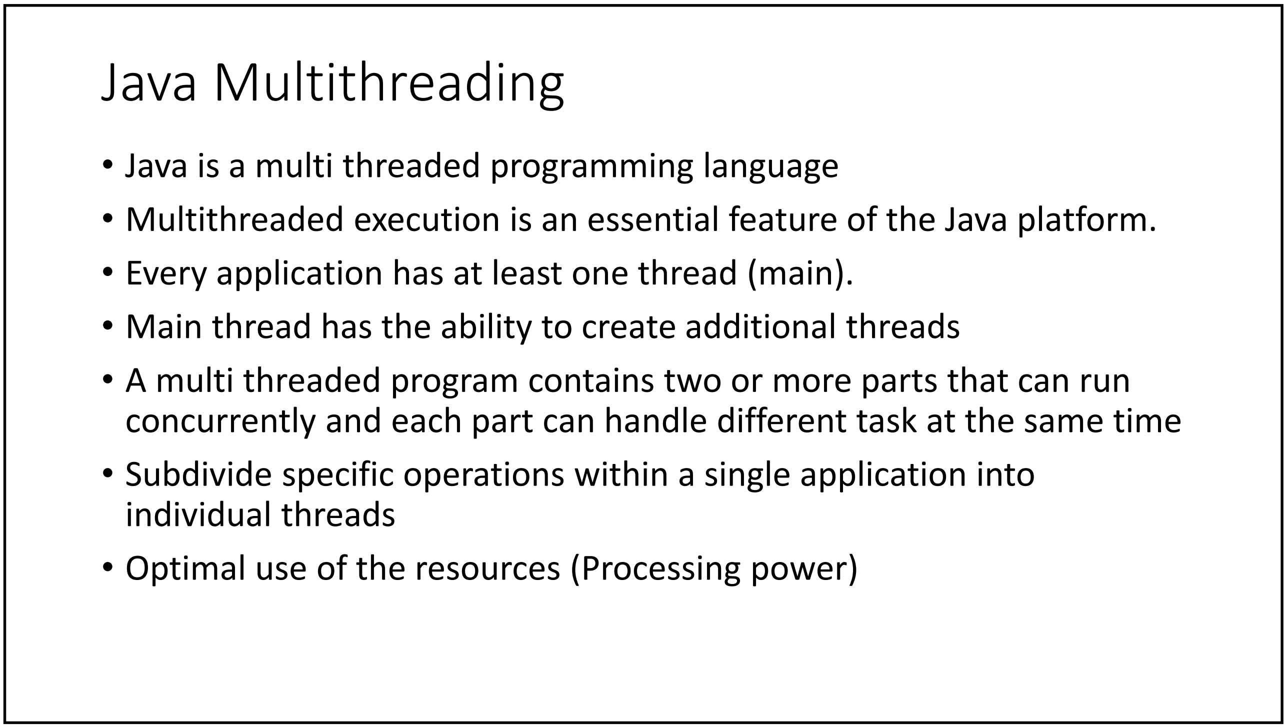
1



Again, Types of Multitasking

- Process-based
  - has a self-contained execution environment.
  - has a complete, private set of basic run-time resources; (its own memory space)
  - Context switching from one process to another is costly.
- Thread-based (Multithreading)
  - Threads exist within a process — every process has at least one.
  - Threads share the process's resources, including memory and open files.
  - Creating a new thread requires fewer resources than creating a new process.
  - Maximize use of CPU (multicore)

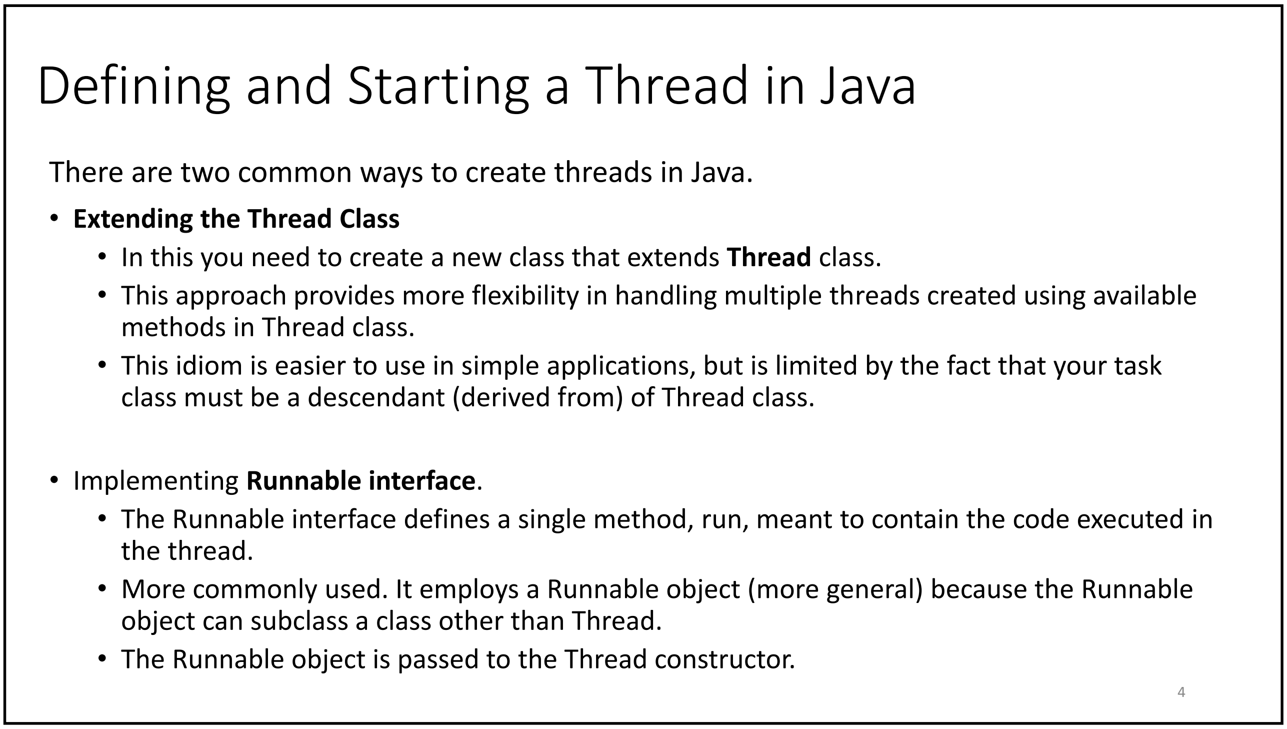
2



Java Multithreading

- Java is a multi threaded programming language
- Multithreaded execution is an essential feature of the Java platform.
- Every application has at least one thread (main).
- Main thread has the ability to create additional threads
- A multi threaded program contains two or more parts that can run concurrently and each part can handle different task at the same time
- Subdivide specific operations within a single application into individual threads
- Optimal use of the resources (Processing power)

3



Defining and Starting a Thread in Java

There are two common ways to create threads in Java.

- **Extending the Thread Class**
  - In this you need to create a new class that extends **Thread** class.
  - This approach provides more flexibility in handling multiple threads created using available methods in Thread class.
  - This idiom is easier to use in simple applications, but is limited by the fact that your task class must be a descendant (derived from) of Thread class.

- Implementing **Runnable interface**.
  - The Runnable interface defines a single method, run, meant to contain the code executed in the thread.
  - More commonly used. It employs a Runnable object (more general) because the Runnable object can subclass a class other than Thread.
  - The Runnable object is passed to the Thread constructor.

4

4

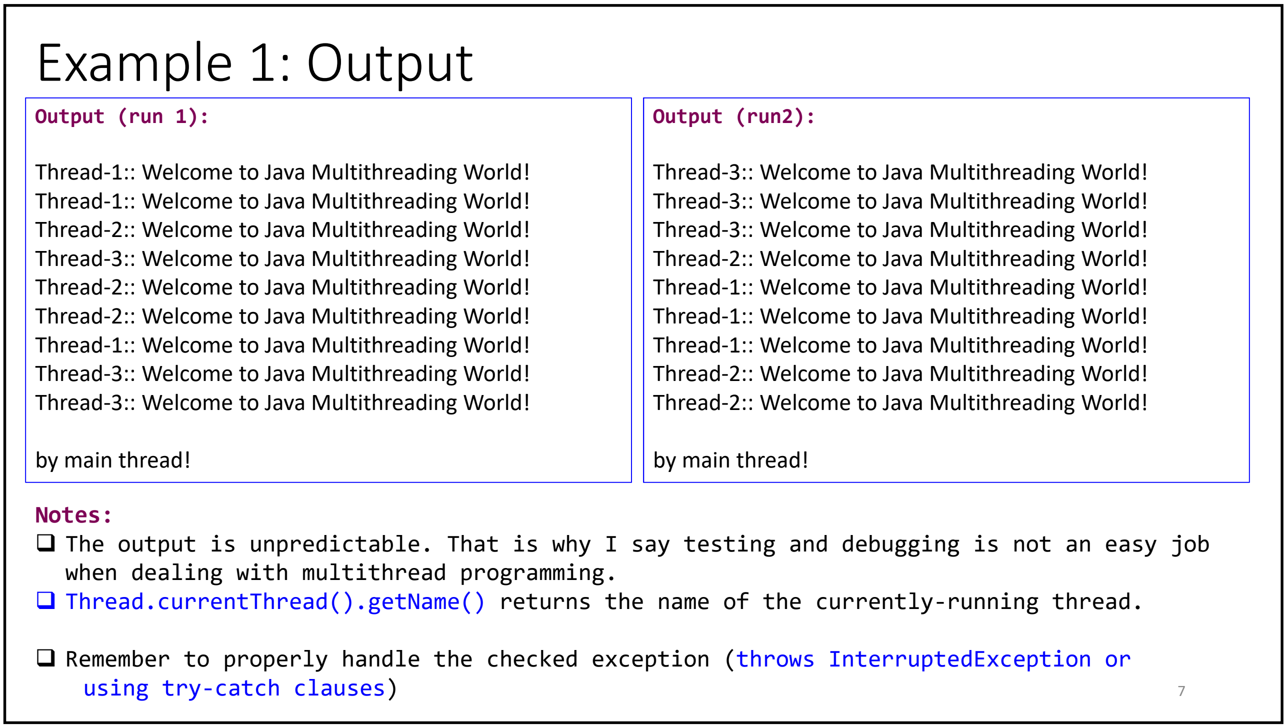
Example 0: Extending the Thread Class

```
public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new HelloThread()).start();
    }
}

//oracle.com
```

Example 1: Extending the Thread Class

```
class Extended extends Thread {
    int limit = 3; // default value
    Extended() {}
    public void run() {
       for (int i = 0; i < limit; ++i) {
         System.out.println(Thread.currentThread().getName() + ": Welcome to Java Multithreading
        World!");
        }
    }
}  //end of class
```

```
public class TestingMutliThreading {
    public static void main(String[] args) throws InterruptedException {
        Extended R1 = new Extended();
        Thread t1 = new Thread(R1);
        Thread t2 = new Thread(R1);
        Thread t3 = new Thread(R1);
        t1.start(); t2.start(); t3.start();
        t1.join(); t2.join(); t3.join();
        System.out.println(“\nby main thread!");
    }
} //end of class
```



Example 1: Output

Output (run 1):

```
Thread-1:: Welcome to Java Multithreading World!
Thread-1:: Welcome to Java Multithreading World!
Thread-2:: Welcome to Java Multithreading World!
Thread-3:: Welcome to Java Multithreading World!
Thread-2:: Welcome to Java Multithreading World!
Thread-2:: Welcome to Java Multithreading World!
Thread-1:: Welcome to Java Multithreading World!
Thread-3:: Welcome to Java Multithreading World!
Thread-3:: Welcome to Java Multithreading World!

by main thread!
```

Output (run2):

```
Thread-3:: Welcome to Java Multithreading World!
Thread-3:: Welcome to Java Multithreading World!
Thread-3:: Welcome to Java Multithreading World!
Thread-2:: Welcome to Java Multithreading World!
Thread-1:: Welcome to Java Multithreading World!
Thread-1:: Welcome to Java Multithreading World!
Thread-1:: Welcome to Java Multithreading World!
Thread-2:: Welcome to Java Multithreading World!
Thread-2:: Welcome to Java Multithreading World!

by main thread!
```

Notes:

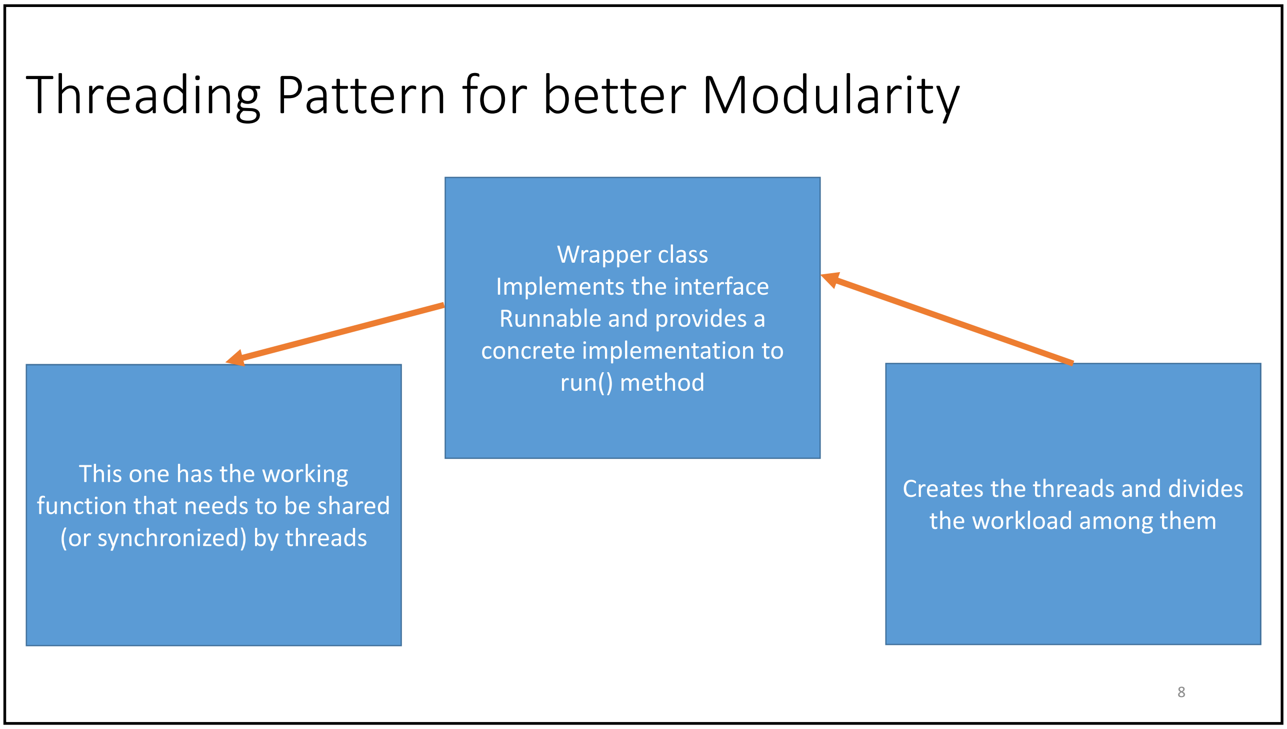
❑ The output is unpredictable. That is why I say testing and debugging is not an easy job when dealing with multithread programming.

❑ `Thread.currentThread().getName()` returns the name of the currently-running thread.

❑ Remember to properly handle the checked exception (`throws InterruptedException` or using `try-catch` clauses)

7

7



Threading Pattern for better Modularity

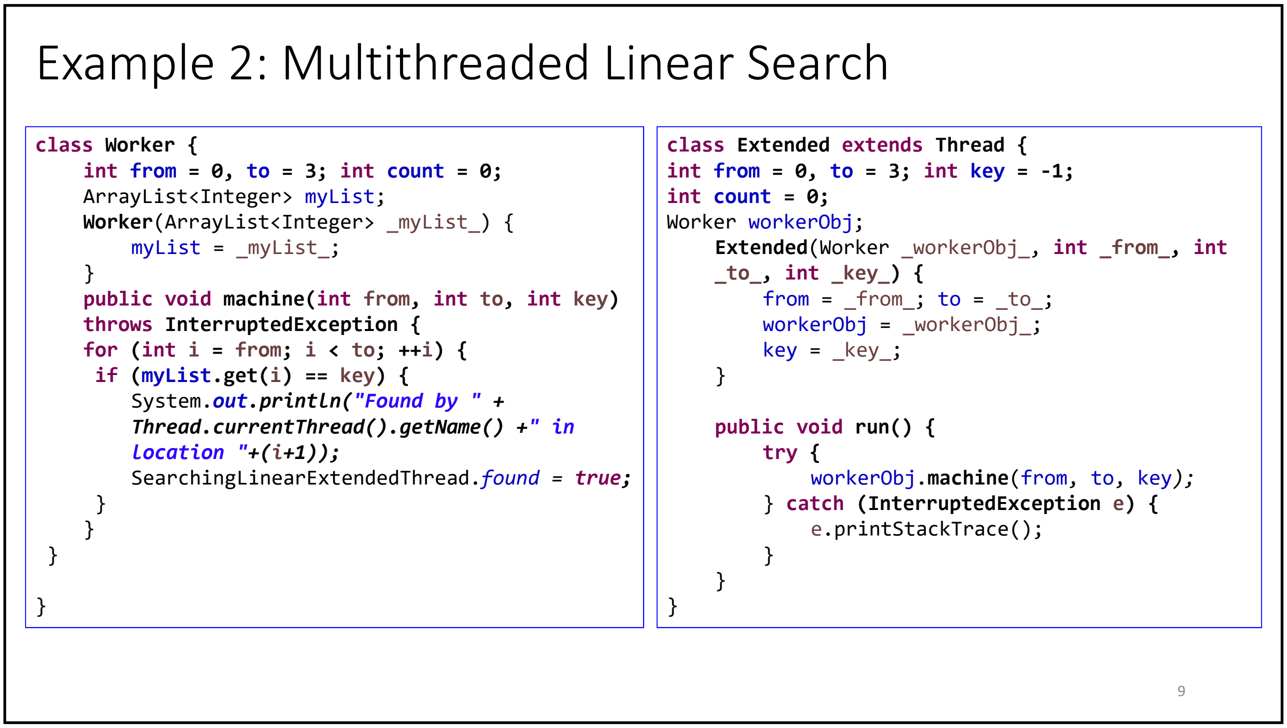
Wrapper class Implements the interface Runnable and provides a concrete implementation to run() method

This one has the working function that needs to be shared (or synchronized) by threads

Creates the threads and divides the workload among them

8

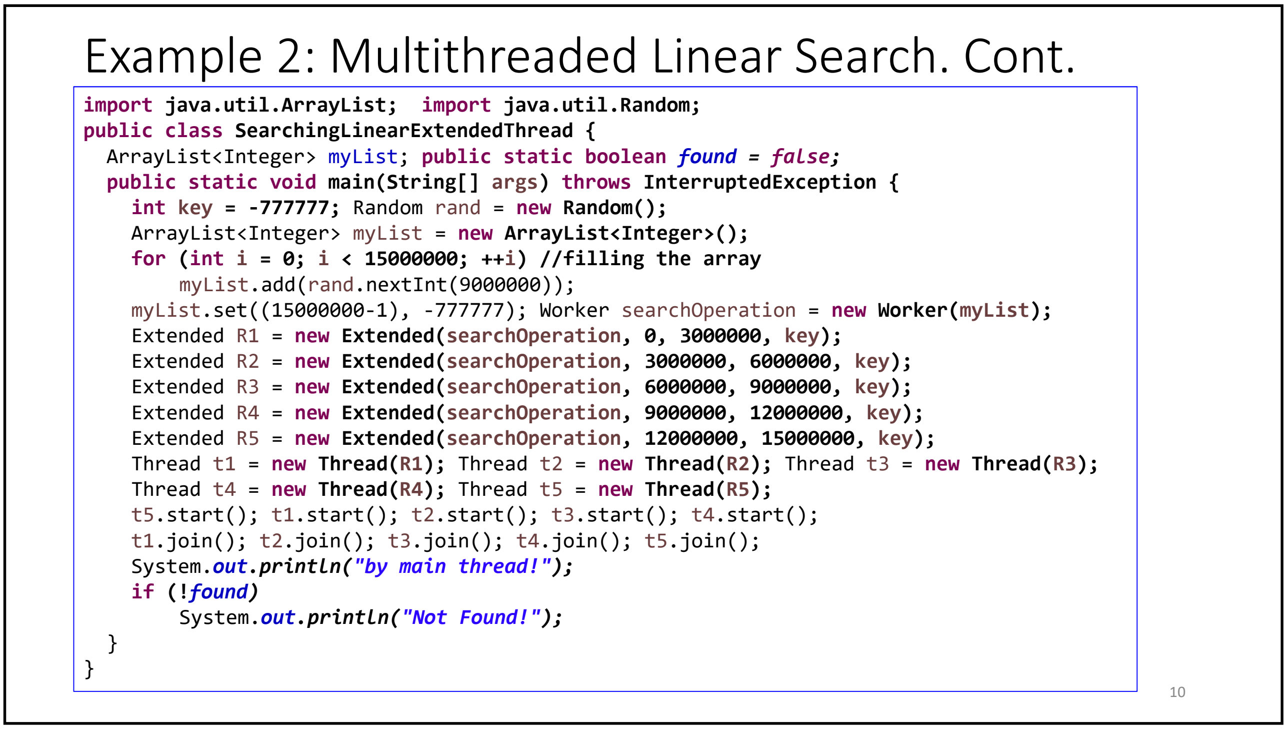
8



Example 2: Multithreaded Linear Search

```
class Worker {
    int from = 0, to = 3; int count = 0;
    ArrayList<Integer> myList;
    Worker(ArrayList<Integer> _myList_) {
        myList = _myList_;
    }
    public void machine(int from, int to, int key)
    throws InterruptedException {
    for (int i = from; i < to; ++i) {
     if (myList.get(i) == key) {
        System.out.println("Found by " +
        Thread.currentThread().getName() +" in
        location "+(i+1));
        SearchingLinearExtendedThread.found = true;
     }
    }
 }

}
```

```
class Extended extends Thread {
int from = 0, to = 3; int key = -1;
int count = 0;
Worker workerObj;
    Extended(Worker _workerObj_, int _from_, int
    _to_, int _key_) {
        from = _from_; to = _to_;
        workerObj = _workerObj_;
        key = _key_;
    }

    public void run() {
        try {
            workerObj.machine(from, to, key);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```



Example 2: Multithreaded Linear Search. Cont.

```
import java.util.ArrayList;  import java.util.Random;
public class SearchingLinearExtendedThread {
  ArrayList<Integer> myList; public static boolean found = false;
  public static void main(String[] args) throws InterruptedException {
    int key = -777777; Random rand = new Random();
    ArrayList<Integer> myList = new ArrayList<Integer>();
    for (int i = 0; i < 15000000; ++i) //filling the array
        myList.add(rand.nextInt(9000000));
    myList.set((15000000-1), -777777); Worker searchOperation = new Worker(myList);
    Extended R1 = new Extended(searchOperation, 0, 3000000, key);
    Extended R2 = new Extended(searchOperation, 3000000, 6000000, key);
    Extended R3 = new Extended(searchOperation, 6000000, 9000000, key);
    Extended R4 = new Extended(searchOperation, 9000000, 12000000, key);
    Extended R5 = new Extended(searchOperation, 12000000, 15000000, key);
    Thread t1 = new Thread(R1); Thread t2 = new Thread(R2); Thread t3 = new Thread(R3);
    Thread t4 = new Thread(R4); Thread t5 = new Thread(R5);
    t5.start(); t1.start(); t2.start(); t3.start(); t4.start();
    t1.join(); t2.join(); t3.join(); t4.join(); t5.join();
    System.out.println("by main thread!");
    if (!found)
        System.out.println("Not Found!");
  }
}
```

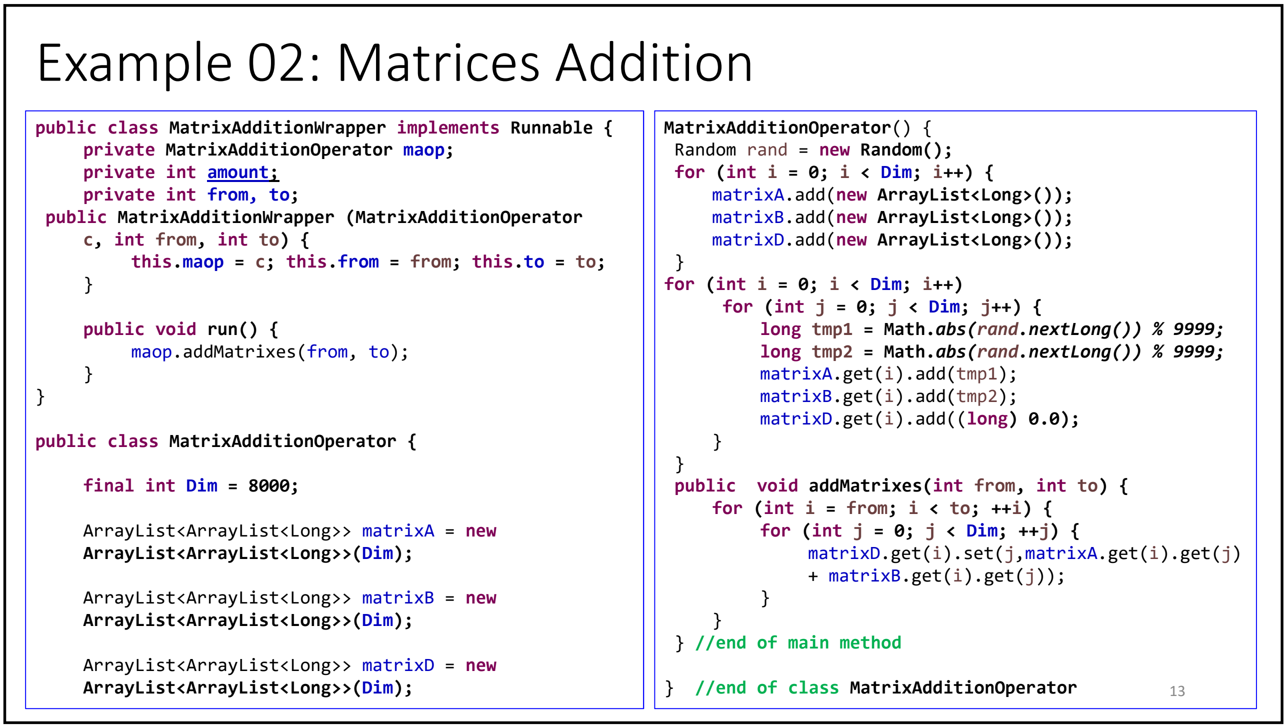
Example 00: Implementing Runnable Interface

```
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}
//oracle.com
```

Example 01: Implementing Runnable Interface

```
class Wrapper implements Runnable {
    int from = 0, to = 3; int count = 0;
    Wrapper() {}
    Wrapper(int _from_, int _to_) { from = _from_; to = _to_; }
    public void run() { worker(from, to); }
    private void worker(int from, int to) {
        for (int i = from; i < to; ++i) {
            System.out.println(Thread.currentThread().getName() + ": Welcome to Java
            Multithreading World! ");
        }
    }
}
```

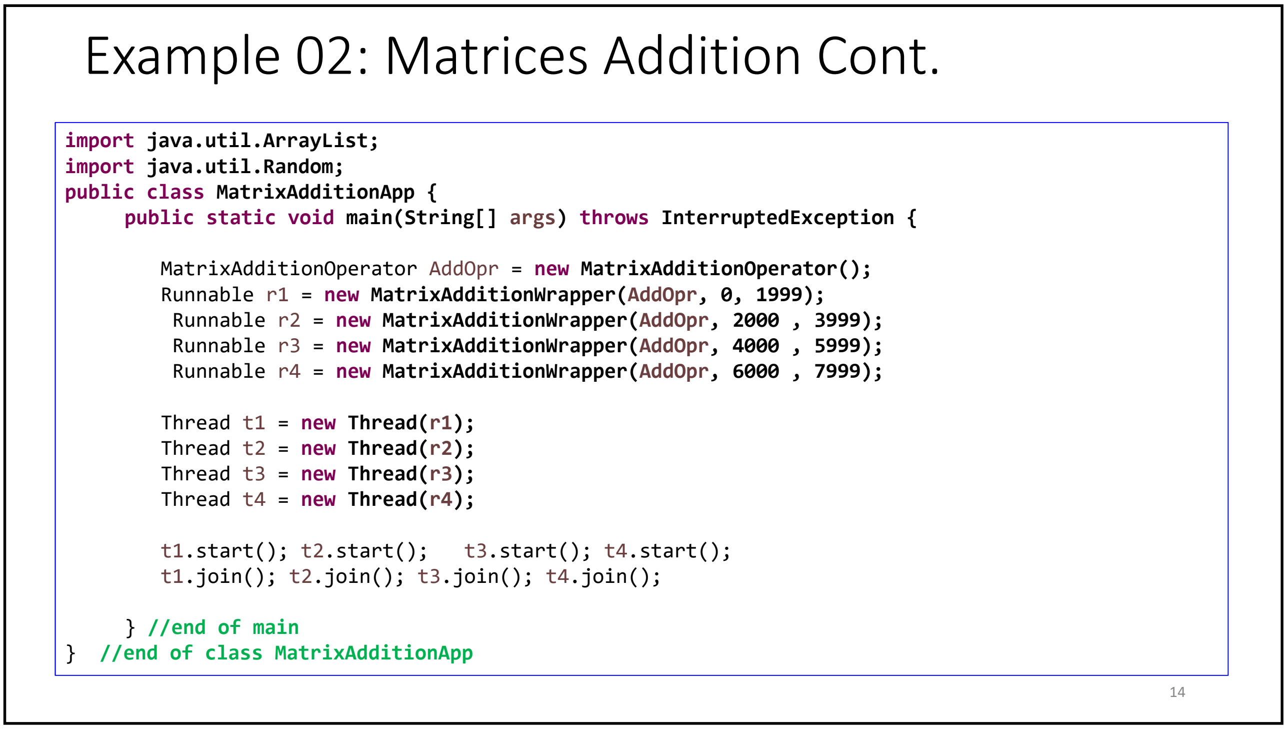
```
public class TestingMutliThreading {
    public static void main(String[] args) throws InterruptedException {
        Runnable R1 = new Wrapper();
        Thread t1 = new Thread(R1);
        Thread t2 = new Thread(R1);
        Thread t3 = new Thread(R1);
        t1.start(); t2.start(); t3.start(); t1.join(); t2.join(); t3.join();
        System.out.println("\nby main thread!");
    }
}
```



Example 02: Matrices Addition

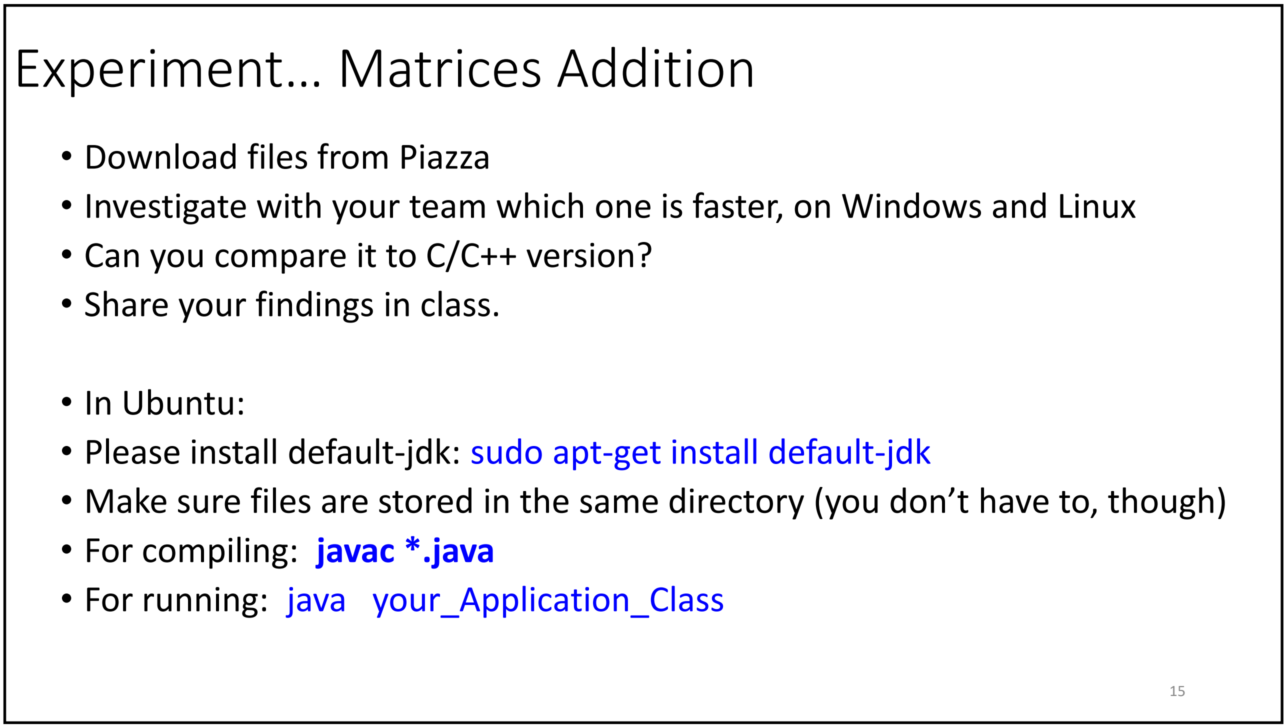
```
public class MatrixAdditionWrapper implements Runnable {
     private MatrixAdditionOperator maop;
     private int amount;
     private int from, to;
 public MatrixAdditionWrapper (MatrixAdditionOperator
     c, int from, int to) {
          this.maop = c; this.from = from; this.to = to;
     }

     public void run() {
          maop.addMatrixes(from, to);
     }
}

public class MatrixAdditionOperator {

     final int Dim = 8000;

     ArrayList<ArrayList<Long>> matrixA = new
     ArrayList<ArrayList<Long>>(Dim);

     ArrayList<ArrayList<Long>> matrixB = new
     ArrayList<ArrayList<Long>>(Dim);

     ArrayList<ArrayList<Long>> matrixD = new
     ArrayList<ArrayList<Long>>(Dim);
```

```
MatrixAdditionOperator() {
 Random rand = new Random();
 for (int i = 0; i < Dim; i++) {
     matrixA.add(new ArrayList<Long>());
     matrixB.add(new ArrayList<Long>());
     matrixD.add(new ArrayList<Long>());
 }
for (int i = 0; i < Dim; i++)
      for (int j = 0; j < Dim; j++) {
          long tmp1 = Math.abs(rand.nextLong()) % 9999;
          long tmp2 = Math.abs(rand.nextLong()) % 9999;
          matrixA.get(i).add(tmp1);
          matrixB.get(i).add(tmp2);
          matrixD.get(i).add((long) 0.0);
     }
 }
 public  void addMatrixes(int from, int to) {
     for (int i = from; i < to; ++i) {
          for (int j = 0; j < Dim; ++j) {
               matrixD.get(i).set(j,matrixA.get(i).get(j)
               + matrixB.get(i).get(j));
          }
     }
 } //end of main method

}  //end of class MatrixAdditionOperator
```



Example 02: Matrices Addition Cont.

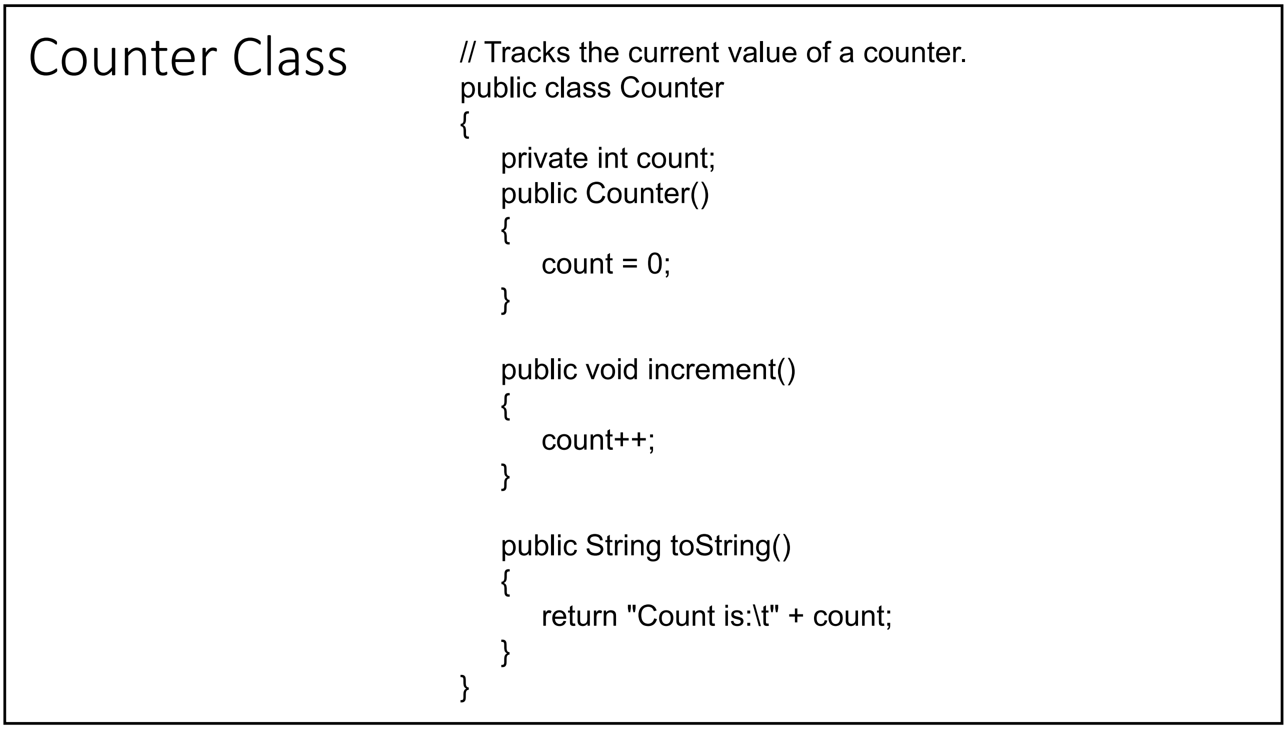
```
import java.util.ArrayList;
import java.util.Random;
public class MatrixAdditionApp {
     public static void main(String[] args) throws InterruptedException {

        MatrixAdditionOperator AddOpr = new MatrixAdditionOperator();
        Runnable r1 = new MatrixAdditionWrapper(AddOpr, 0, 1999);
         Runnable r2 = new MatrixAdditionWrapper(AddOpr, 2000 , 3999);
         Runnable r3 = new MatrixAdditionWrapper(AddOpr, 4000 , 5999);
         Runnable r4 = new MatrixAdditionWrapper(AddOpr, 6000 , 7999);

        Thread t1 = new Thread(r1);
        Thread t2 = new Thread(r2);
        Thread t3 = new Thread(r3);
        Thread t4 = new Thread(r4);

        t1.start(); t2.start();   t3.start(); t4.start();
        t1.join(); t2.join(); t3.join(); t4.join();

     } //end of main
}  //end of class MatrixAdditionApp
```



Experiment… Matrices Addition

- Download files from Piazza
- Investigate with your team which one is faster, on Windows and Linux
- Can you compare it to C/C++ version?
- Share your findings in class.

- In Ubuntu:
- Please install default-jdk: sudo apt-get install default-jdk
- Make sure files are stored in the same directory (you don’t have to, though)
- For compiling: **javac *.java**
- For running: java   your_Application_Class



Counter Class

```
// Tracks the current value of a counter.
public class Counter
{
    private int count;
    public Counter()
    {
        count = 0;
    }

    public void increment()
    {
        count++;
    }

    public String toString()
    {
        return "Count is:\t" + count;
    }
}
```

Demo One - Basic

```
public class Demo01
{
    public static void main(String[] args)
    {
        Counter myCounter    = new Counter();
        myCounter.increment();
        myCounter.increment();
        myCounter.increment();
        System.out.println(myCounter);
    }
}
```

The output of the program is:

Count is: 3

17

Defining and Starting a Thread

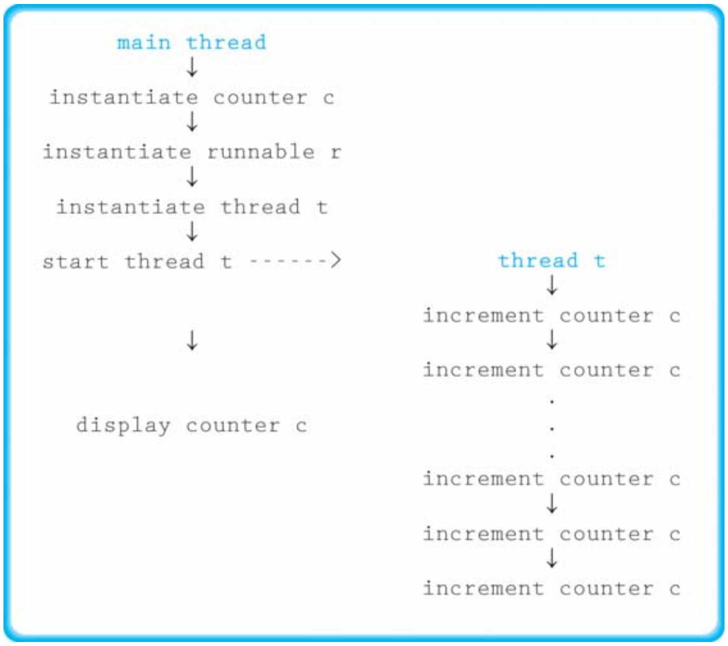
```
public class Increase  implements Runnable
{
  private Counter c;
  private int amount;
  public Increase (Counter c, int amount)
  {
    this.c = c;
    this.amount = amount;
  }
  public void run()
  {
  for (int i = 1; i <= amount; i++)
    c.increment();
  }
}
```

```
public class Demo02
{
  public static void main(String[] args) throws
   InterruptedException {
    Counter c = new Counter();
    Runnable r = new Increase(c, 10000);
    Thread t = new Thread(r);
    t.start();
    System.out.println("Count is: " + c);


  }


}
```

Output Varies: 86, 66, 44 ????

18

Demo Two - Threads



main thread

instantiate counter c

instantiate runnable r

instantiate thread t

start thread t ------>

display counter c

thread t

increment counter c

increment counter c

.
.
.

increment counter c

increment counter c

increment counter c

19

Demo Three - Join

```
public class Increase implements Runnable
{
  private Counter c;
  private int amount;
  public Increase (Counter c, int amount)
  {
    this.c = c; this.amount = amount;
  }
  public void run()
  {
  for (int i = 1; i <= amount; i++)
    c.increment();
  }
}
```

```
public class Demo02
{
        public static void main(String[] args) throws
         throws InterruptedException
         {
           Counter c = new Counter();
           Runnable r = new Increase(c, 10000);
           Thread t = new Thread(r);
           t.start();
           t.join();
           System.out.println("Count is: " + c);

        }
}
```

Output is 10000

20

Demo Four - Interference

```
public class Increase
        implements Runnable
{
  private Counter c;
  private int amount;
  public Increase (Counter c, int amount)
  {
    this.c = c; this.amount = amount;
  }
  public void run()
  {
  for (int i = 1; i <= amount; i++)
    c.increment();
  }
}
```

```
public class Demo04
{
  public static void main(String[] args)
       throws InterruptedException
  {
    Counter c = new Counter();
    Runnable r1 = new Increase(c, 5000);
    Runnable r2 = new Increase(c, 5000);
   Thread t1 = new Thread(r1);
   Thread t2 = new Thread(r2);
   t1.start();  t2.start();
   t1.join();   t2.join();
   System.out.println("Count is: " + c);
  }
}
```

Output Varies: 9861, 9478 ????

21

Interference, Race Condition (2)

Thread t1	**Thread t2**
Step 1: obtains value 12	
	Step 2: obtains value 12
Step 3: increments value to 13	
Step 4: stores the value 13	
	Step 5: increments value to 13
	Step 6: stores the value 13

A **Race Condition** occurs when two (or more) threads access a shared variable at the same time causing a semantic error that can potentially lead to unpredictable results. That is, it is a flaw that occurs in the timing or the ordering of events that leads to erroneous program behavior.

22

Interference, Race Condition, and Critical Section

```
public class Increase
        implements Runnable
{
  private Counter c;
  private int amount;
  public Increase (Counter c, int amount)
  {
    this.c = c; this.amount = amount;
  }
  public void run() // Critical Section
  {
  for (int i = 1; i <= amount; i++)
    c.increment();
  }
}
```

```
public class Demo04
{
  public static void main(String[] args)
       throws InterruptedException
  {
    Counter c = new Counter();
    Runnable r1 = new Increase(c, 5000);
    Runnable r2 = new Increase(c, 5000);
   Thread t1 = new Thread(r1);
   Thread t2 = new Thread(r2);
   t1.start();   t2.start();
   t1.join();    t2.join();
   System.out.println("Count is: " + c);
  }
}
```

Output Varies: 9861, 9478 ????

23

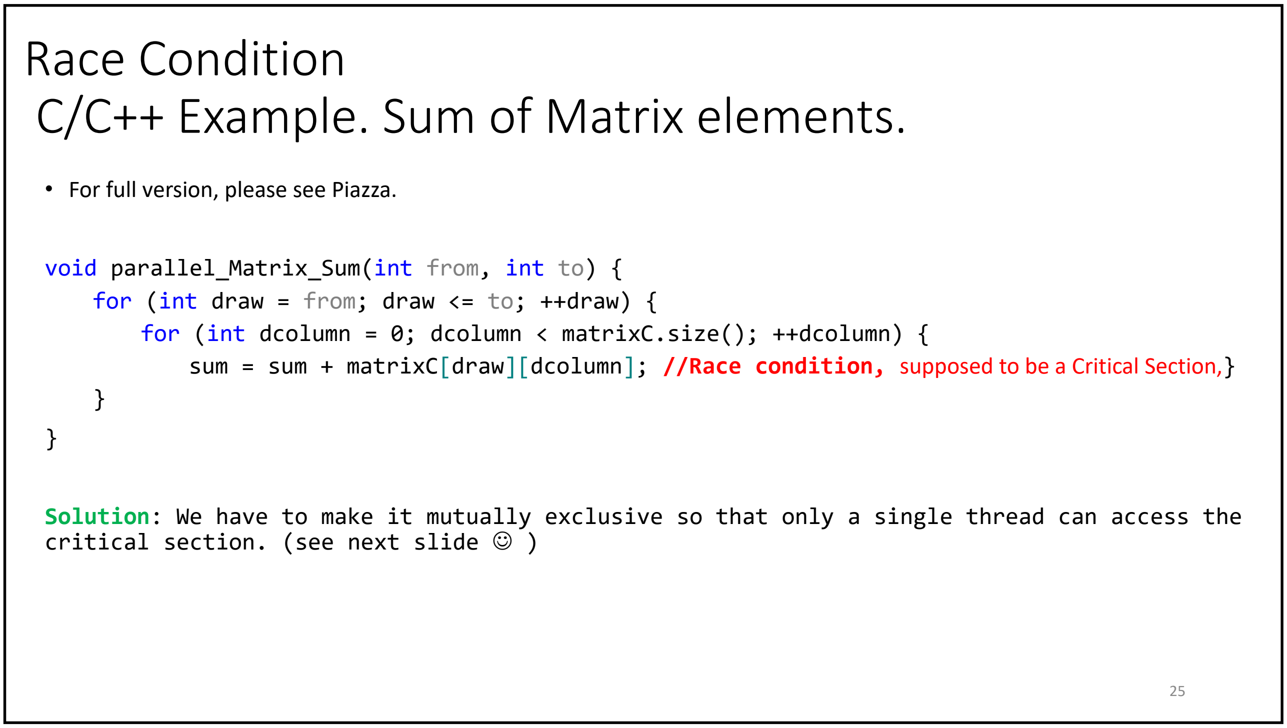
Synchronization Solution in Java

```
// Provides synchronized access
public class SyncCounter
{
  private int count;
  public SyncCounter()  {     count = 0;   }


  public synchronized void increment() // Make it Mutually
   Exclusive
  {
    count++;  // one thread at a time
  }
  public String toString()
  {
    return "Count is:\t" + count;
  }
} // end of class SyncCounter
```

```
public class Demo05
{
  public static void main(String[] args) throws
   InterruptedException
  {
    SyncCounter sc = new SyncCounter();
    Runnable r1 = new Increase2(sc, 5000);
    Runnable r2 = new Increase2(sc, 5000);
    Thread t1 = new Thread(r1);
    Thread t2 = new Thread(r2);
    t1.start(); t2.start();
    t1.join(); t2.join();
    System.out.println("Count is: " + sc);
  }
} // end of class Demo05
```

Output is 10000

24

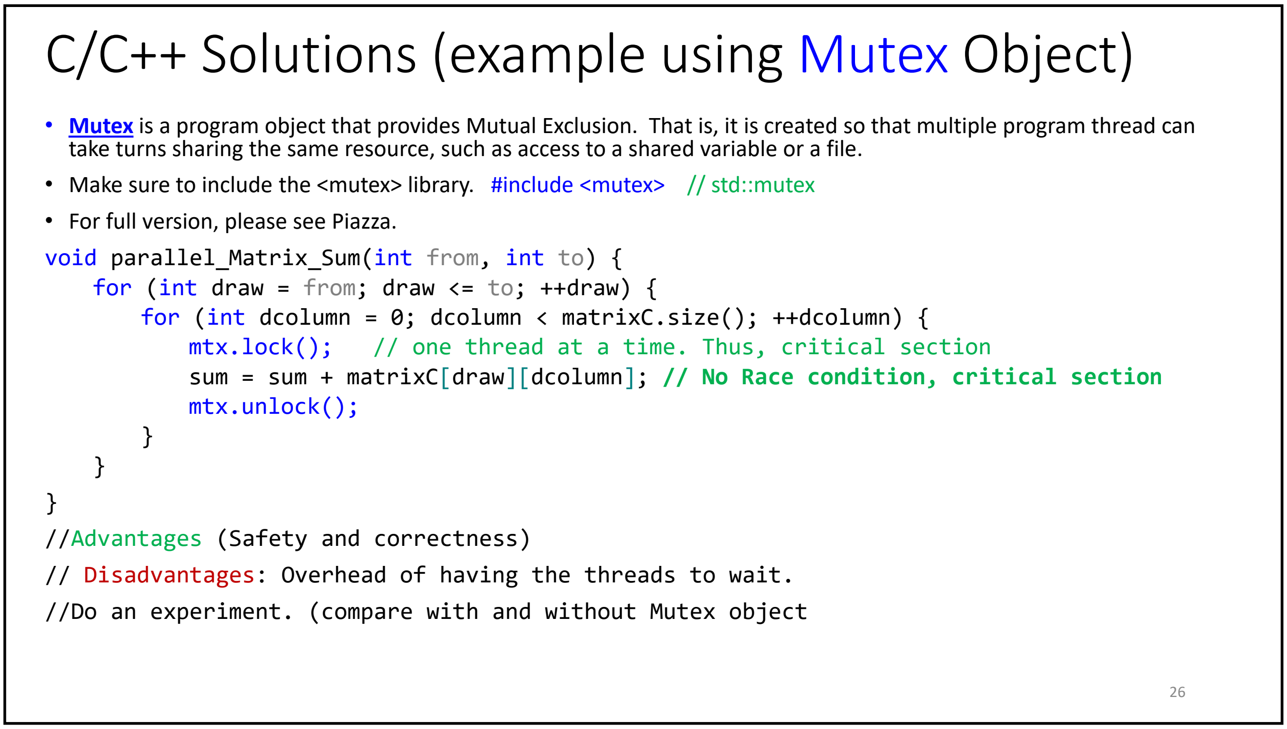


Race Condition
C/C++ Example. Sum of Matrix elements.

- For full version, please see Piazza.

```
void parallel_Matrix_Sum(int from, int to) {
    for (int draw = from; draw <= to; ++draw) {
        for (int dcolumn = 0; dcolumn < matrixC.size(); ++dcolumn) {
            sum = sum + matrixC[draw][dcolumn]; //Race condition, supposed to be a Critical Section,}
    }
}
```

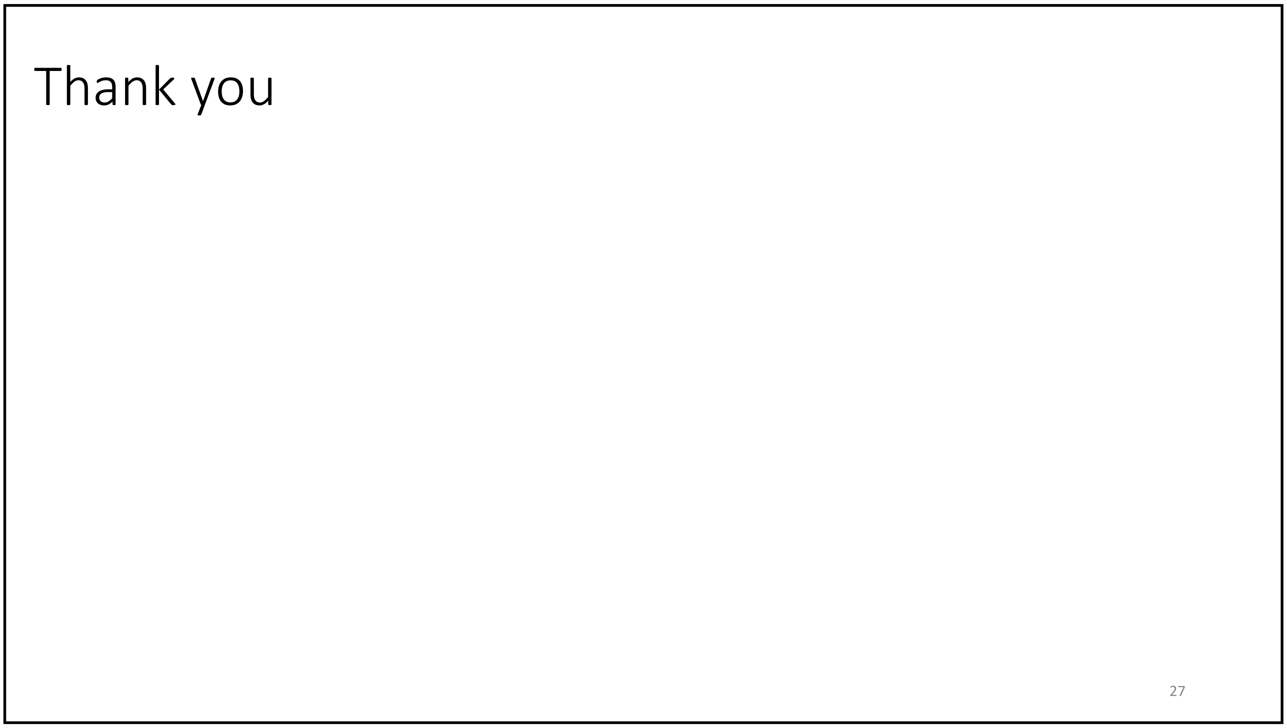
Solution: We have to make it mutually exclusive so that only a single thread can access the critical section. (see next slide ☺)



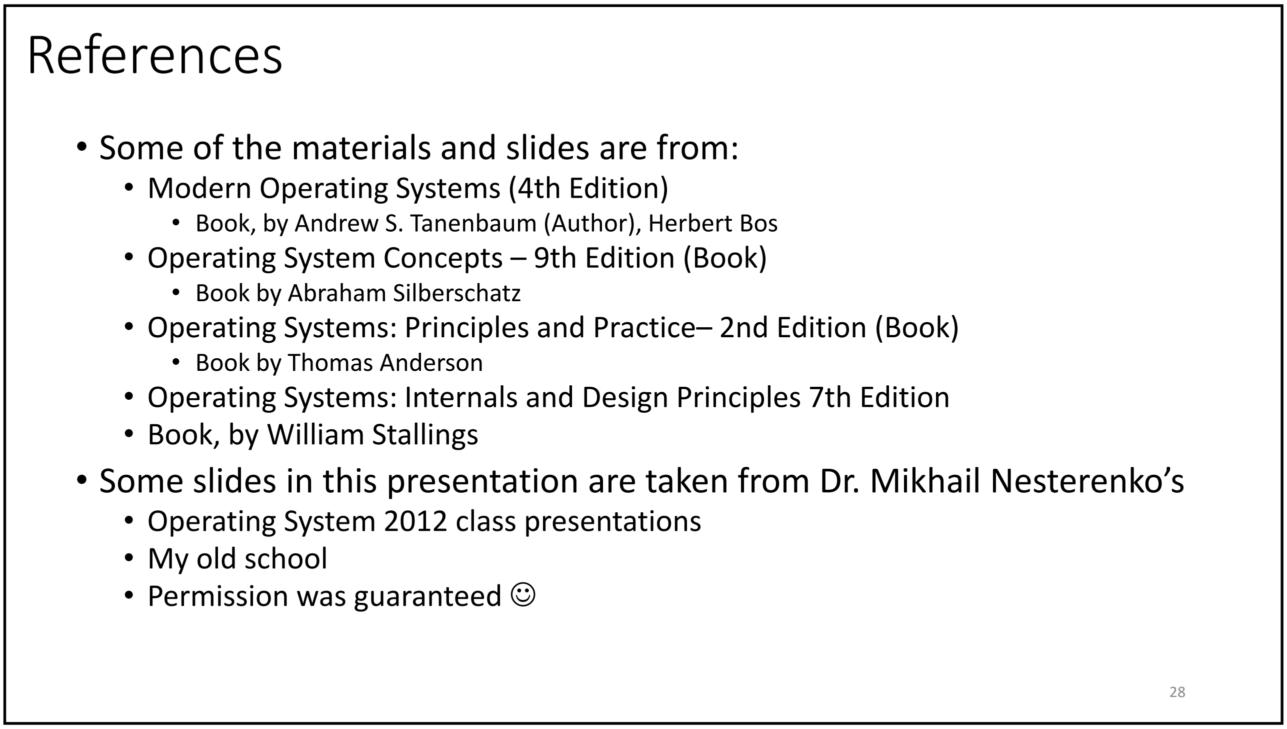
C/C++ Solutions (example using Mutex Object)

- **Mutex** is a program object that provides Mutual Exclusion. That is, it is created so that multiple program thread can take turns sharing the same resource, such as access to a shared variable or a file.
- Make sure to include the <mutex> library. #include <mutex> // std::mutex
- For full version, please see Piazza.

```
void parallel_Matrix_Sum(int from, int to) {
    for (int draw = from; draw <= to; ++draw) {
        for (int dcolumn = 0; dcolumn < matrixC.size(); ++dcolumn) {
            mtx.lock();   // one thread at a time. Thus, critical section
            sum = sum + matrixC[draw][dcolumn]; // No Race condition, critical section
            mtx.unlock();
        }
    }
}
//Advantages (Safety and correctness)
// Disadvantages: Overhead of having the threads to wait.
//Do an experiment. (compare with and without Mutex object
```



Thank you



References

- Some of the materials and slides are from:
  - Modern Operating Systems (4th Edition)
    - Book, by Andrew S. Tanenbaum (Author), Herbert Bos
  - Operating System Concepts – 9th Edition (Book)
    - Book by Abraham Silberschatz
  - Operating Systems: Principles and Practice– 2nd Edition (Book)
    - Book by Thomas Anderson
  - Operating Systems: Internals and Design Principles 7th Edition
  - Book, by William Stallings
- Some slides in this presentation are taken from Dr. Mikhail Nesterenko's
  - Operating System 2012 class presentations
  - My old school
  - Permission was guaranteed ☺

References 2

- Dr. Saleh Alnaeli, The easiest and fastest path to CS.
  - IBM.com
- Object-Oriented Data Structures Using Java (3rd edition),
  - by Dale, Joyce, and Weems (code and slides)
- Algorithm Design: Foundations, Analysis, and Internet Examples
  - by Michael T. Goodrich, Roberto Tamassia
- Data Structures and Algorithms in Java 6/E
  - by Michael T. Goodrich, Roberto Tamassia, Michael H. Goldwasser
  - {slides, text, and code}
- Oracle, Java Documentation
  - https://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html
- Intel (pictures, slides 6 and 7)
  - Intel.com

29

30

30