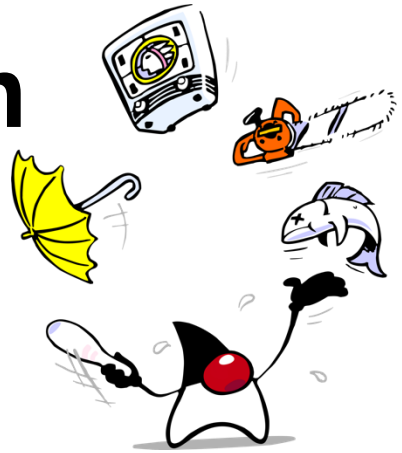
Concurrency and Parallelism (Multi-threading)

CO7005 Software Development Techniques



Dr Stuart Cunningham
s.cunningham@chester.ac.uk

Count the People…

Image source: https://bleedingcool.com/tv/massive-iconic-poster-of-the-simpsons-cast-hits-auction/

Count the People...get a friend to help

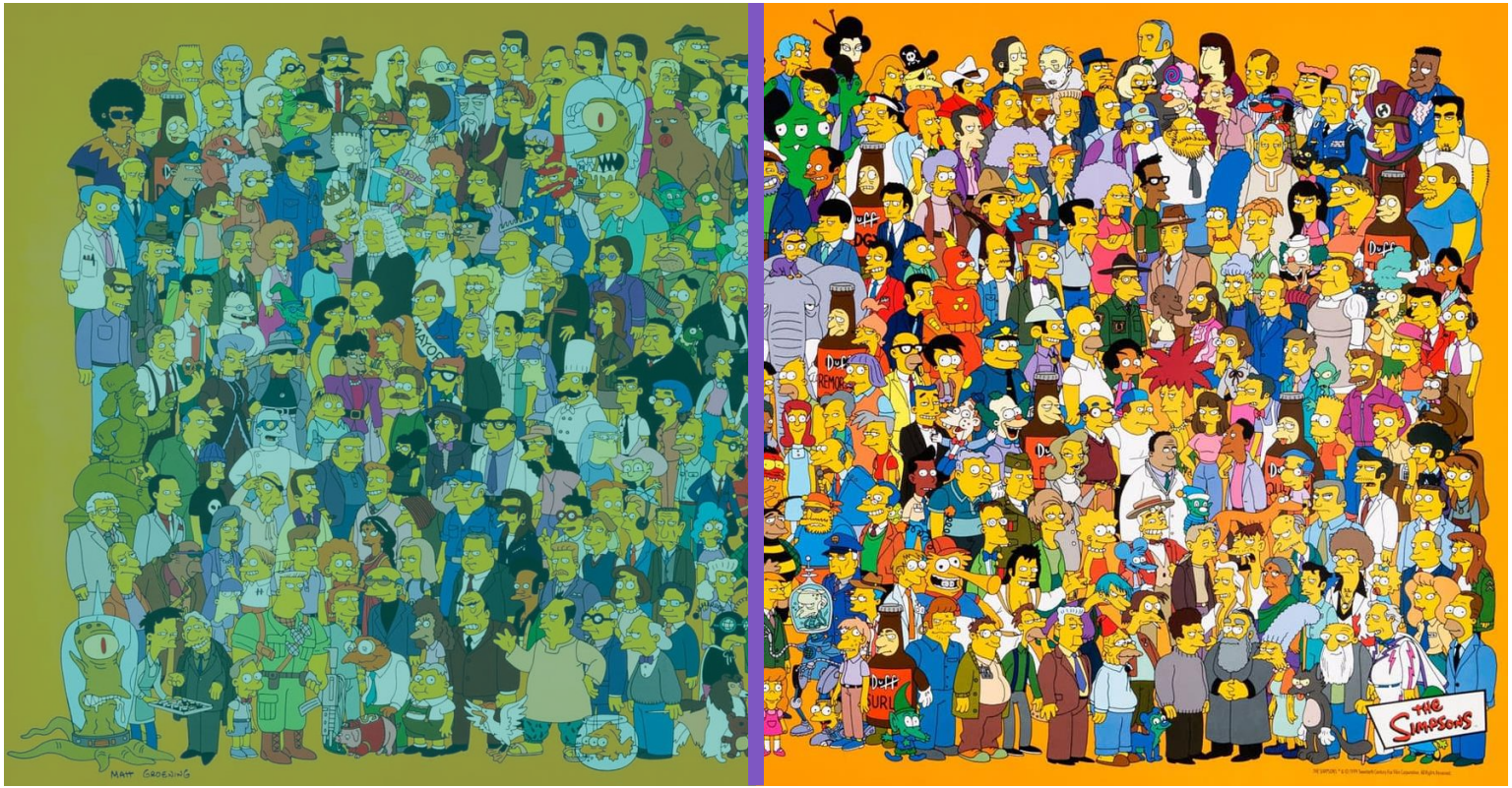


Image source: https://bleedingcool.com/tv/massive-iconic-poster-of-the-simpsons-cast-hits-auction/

Count the People…3 friends helping

Image source: https://bleedingcool.com/tv/massive-iconic-poster-of-the-simpsons-cast-hits-auction/

Count the People…7 friends helping

Image source: https://bleedingcool.com/tv/massive-iconic-poster-of-the-simpsons-cast-hits-auction/

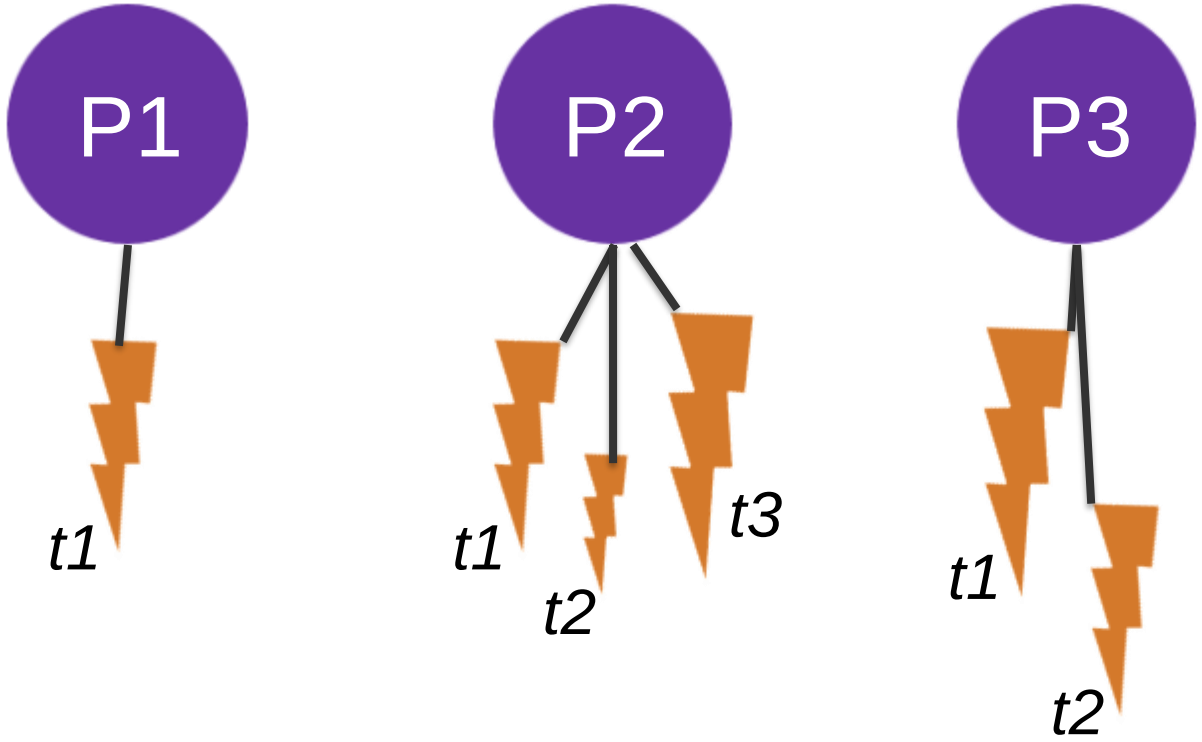
Processes

- Large sets instructions to be executed on one or more sets of data
- Every program has at least one process
- Crucially, processes have:
  - A dedicated memory allocation
  - Access to system resources
  - Independence from other processes
- Inter Process Communication (IPC) possible, but costly
- Scheduled and executed by the operating system

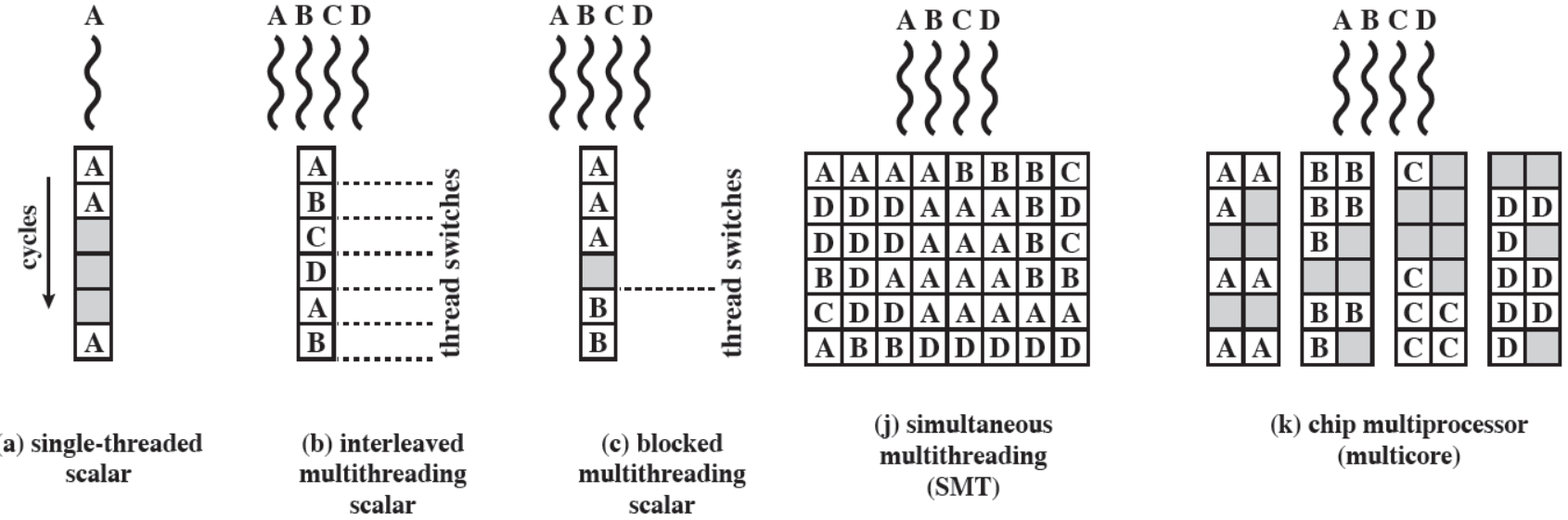
Threads

- A small series of instructions or tasks within a process
- A process must contain one, or more, threads
- Threads share the parent process resources, particularly *memory*
  - Therefore, utilise the same *instructions* and *data*
- Care required so that critical instructions don’t fall out of sequence
  - For instance: 5*(3+2) is quite different to 5*3+2
  - (the first returns 25, the second 17)
- *Synchronisation* helps ensure sequences are adhered to

Processes and Threads

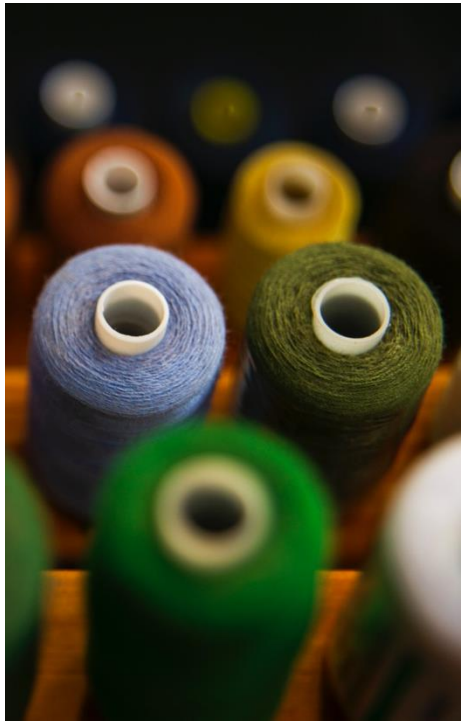


Multi-threading Approaches (Stallings 2022)



Parallel Processing in the 1980s

Threads in Java



- Multi-threading, controlled by the programmer
- Idle time can be utilised, increasing efficiency
- Every program has a `main` thread
- Threads can be in one of several states
  - Running, ready-to-run, suspended, resumed, blocked, terminated
- Java provides easy mechanisms to facilitate threads and support synchronization
  - Thread *class* and Runnable *interface*

Threads in Java

“…while one part of your program is sending a file over the Internet, another part can be reading keyboard input, and still another can be buffering the next block of data to send.”

(Schildt 2022)

Creating Threads

- Instantiate a new Thread object
- Make the Thread *runnable* in one of two ways:

1. Extend the Thread class
2. Implement the runnable interface and its `run()` method
    - All code in, and methods called by, `run()` form the thread

- There are several ways to deploy threads in programs…

Threads in Java - Useful Methods

`Thread.start()`	Starts a thread's execution
`Thread.sleep()`	Pauses execution of a thread (pause time in milliseconds)
`Thread.join()`	Wait for thread to end (maximum waiting time in milliseconds)
`Thread.interrupt()`	Interrupts thread execution
`Thread.getName()`	Gets name of the thread, returns a String
`Thread.isAlive()`	True if a thread is still running

SimpleThread.java

- Creation of two threads
  - Main thread (automatic)
  - An on-the-fly thread (could add more if we wished)
- Effective, but not especially complex or practical

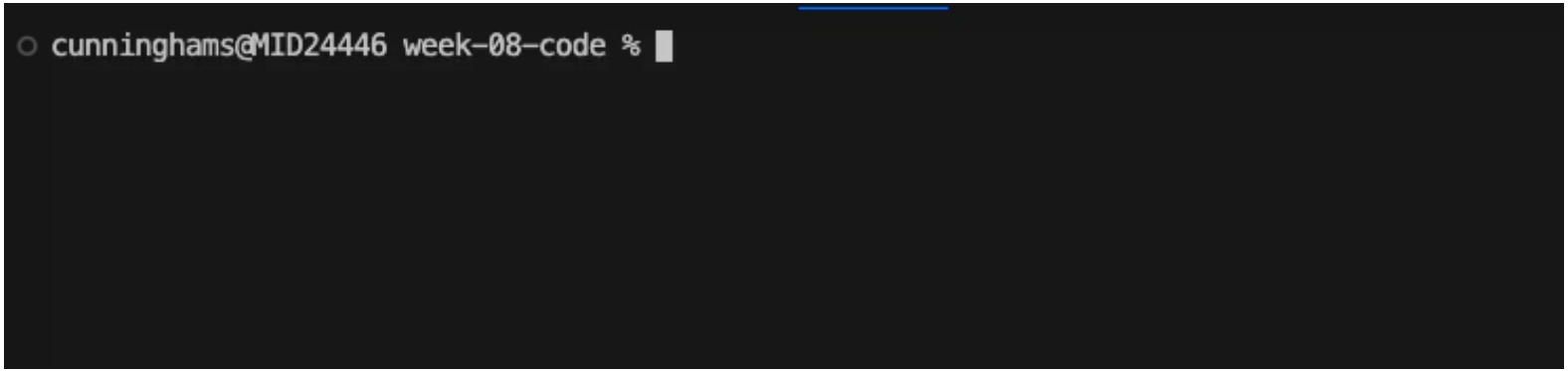
SimpleThread.java

```
>> java SimpleThread
Hello from the main thread!
Hello, I'm part of another thread!
```

```java
class SimpleThread implements Runnable
{
 // main method equipped to throw Interrupted Exception
 public static void main(String args[]) throws InterruptedException {
  // create new Thread object an on-the-fly instance
  Thread myThread = new Thread(new SimpleThread());
  System.out.println("Hello from the main thread!");
  // start running myThread
  myThread.start();
 }
 // the run method is the default action to be executed
 // when a thread starts
 public void run() {
  System.out.println("Hello, I'm part of another thread!");
 }
}
```

SleepingThreads.java

- Demonstrates use of the sleep()
- No need to implement runnable, only operates on main thread
- Using `Thread.sleep()`
- Used to add a delay to thread execution



ClassThreads.java

- Demonstrates three concurrent Thread instances
- A class ThreadDeploy implements Runnable
  - Instance variables and `run()` method to perform calculations
  - An ID instance variable is included
  - Utilises `sleep()` method from Runnable to add short delays
- ClassThreads program creates instances of ThreadDeploy
  - Corresponding Threads are instantiated using these objects
  - Each thread is activated using the `start()` Thread method

ClassThreads.java

First time I run ClassThreads

```
>> java ClassThreads
ID 2 tangent= -0.13235753235145759
ID 1 tangent= 1.7844248315940128
ID 3 tangent= 2.06595526138051O8
ID 1 tangent= 1.398382589287699
ID 3 tangent= 0.29841278656943165
ID 2 tangent= 19.669527820558873
ID 1 tangent= 0.353736878039122 57
ID 3 tangent= -1.1891624321183722
ID 2 tangent= 1.00924628838275 49
ID 2 tangent= 1.5922060242195704
ID 3 tangent= -12.599264789465776
ID 1 tangent= -0.5099711260626033
```

Second time I run ClassThreads

```
>> java ClassThreads
ID 2 tangent= -0.13235753235145759
ID 3 tangent= 2.06595526138051O8
ID 1 tangent= 1.7844248315940128
ID 1 tangent= 1.398382589287699
ID 2 tangent= 19.669527820558873
ID 3 tangent= 0.29841278656943165
ID 2 tangent= 1.00924628838275 49
ID 3 tangent= -1.1891624321183722
ID 1 tangent= 0.353736878039122 57
ID 1 tangent= -0.5099711260626033
ID 3 tangent= -12.599264789465776
ID 2 tangent= 1.5922060242195704
```

What's unusual about this situation?

```
class ThreadDeploy implements Runnable {

 private double[] angle;
 private int id;

 ThreadDeploy(double[] angle, int id){
  this.angle = angle;
  this.id = id;
 }

 // run method from Runnable
 public void run(){
  try {
   for (int i=0; i<angle.length; i++) {
    double ans = Math.tan(angle[i]);
    System.out.println("ID "+id+" tangent= "+ans);
    Thread.sleep(400);
   }
  }
  catch (InterruptedException e) {
   System.out.println("Thread error: "+e);
  }
 }
}
```

```
public class ClassThreads {
 public static void main(String[] args) {
  // insatiate new ThreadDeploy object
  double[] vals1 = {1.06, 0.95, 0.34, 2.67};
  ThreadDeploy tans1 = new ThreadDeploy(vals1,1);
  // construct thread from tans1
  Thread t1 = new Thread(tans1);

  // do the same thing for a second thread
  double[] vals2 = {3.01, 1.52, 0.79, 1.01};
  ThreadDeploy tans2 = new ThreadDeploy(vals2, 2);
  Thread t2 = new Thread(tans2);

  // do the same thing for a third thread
  double[] vals3 = {1.12, 0.29, 2.27, 1.65};
  ThreadDeploy tans3 = new ThreadDeploy(vals3, 3);
  Thread t3 = new Thread(tans3);

  // run the threads
  t1.start();
  t2.start();
  t3.start();
 }
}
```

MoreThreads.java

- Demonstrates three concurrent Thread instances
- A class TimerThread extends the Thread class (a sub-class)
- TimeThread's constructor makes use of the Thread parent attribute `name` and adds a `time` attribute
  - The `run()` method uses `getName()` from the Thread parent class
- MoreThreads creates three instances of TimerThread, allocating a name and time at instantiation

MoreThreads.java

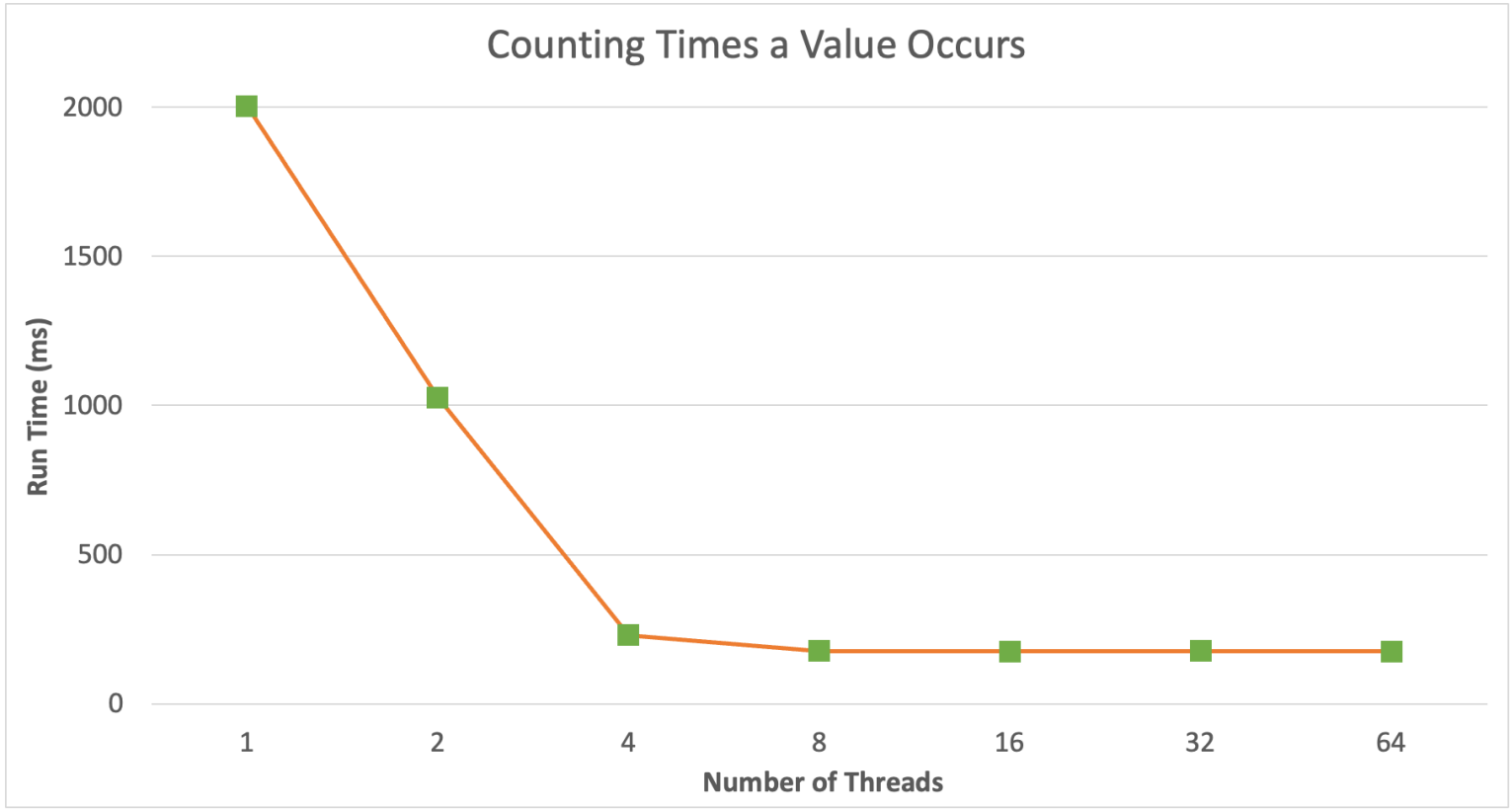
```
public class MoreThreads {

 public static void main(String[] args) {
  // declare instances of the threads we want to use
  TimerThread timer1 = new TimerThread("Tea", 150);
  TimerThread timer2 = new TimerThread("Boiled Egg", 360);
  TimerThread timer3 = new TimerThread("Toast", 30);

  // start each thread
  timer1.start();
  timer2.start();
  timer3.start();
 }
}
```

```
.
.
.
Tea time: 28
Toast time: 28
Boiled Egg time: 28
Tea time: 29
Toast time: 29
Boiled Egg time: 29
Toast time: 30
Tea time: 30
Boiled Egg time: 30
Boiled Egg time: 31
Tea time: 31
Toast is done!
```

ArrayCount.java

- Counting how many time a value appears (`args[0]`)
- Brute force search of a 3D array
- Search takes place by splitting the array into 2^n segments
  - $n = \{0, 1, 2, 3, 4, 5, 6\}$
- Generate and allocate a thread per segment
- Dividing the count over multiple threads
- Sums results from all threads before concluding

ArrayCount.java



References

Computer Chronicles series. Available at: http://www.archive.org

Schildt, H. (2022). *Java: a beginner's guide* (9th ed.). McGraw Hill.

Stallings, W. (2022). *Computer organization and architecture: designing for performance* (Global;Eleventh;). Pearson.