Threads

Programs are sequential

- The code on line 1 executes before line 2
- One thing happens at a time, in a set order

Flattening a program

```
public void run(){
    printThree();
    printThree();
}
private void printThree()
    println("1");
    println("2");
    println("3");
}
```

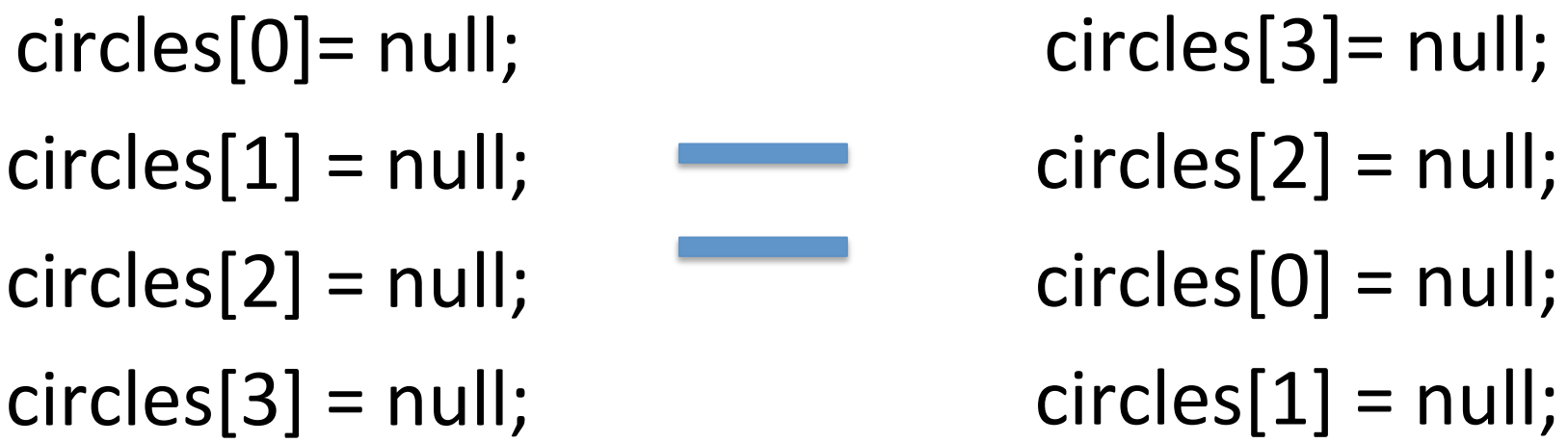
```
println("1");
println("2");
println("3");
println("1");
println("2");
println("3");
```

No matter how complicated it gets, it is still essentially a list of commands executed in order.

What if order doesn’t matter?

Simple Example

```
GOval [] circles = new GOval[4];
for(int i = 0; i<4; i++){
    circles[i] = null;
}
```


circles[0]= null;
circles[1] = null;
circles[2] = null;
circles[3] = null;
=
circles[3]= null;
circles[2] = null;
circles[0] = null;
circles[1] = null;

How can we take advantage of this?

Say you’re making dinner…

Threads

- Allow you to run two sets of instructions at one time.
- Helpful for when you have multiple processors. You can run one “mini-program” on each core, as long as the order doesn’t matter.

Core 1: **(thread 1)**	**Core 2:** **(thread 2)**
circles[0] = null;	circles[2] = null;
circles[1] = null;	circles[3] = null;

If each line takes 1 millisecond, it will only take 2 milliseconds, instead of 4.

Each set of instructions is a **thread**.

Creating a Thread

- Thread t = new Thread(*Runnable Object*)
  - can't just pass in random objects, like "NameSurferEntry"
- Object must implement Runnable
  - needs a method called run()
- Your code is a single thread. In the starter code, we use other threads to deal with mouse events and draw graphics.

Code example!