Poll Everywhere

PollEv.com/javabear text **javabear** to 22333



Why is it important to minimize the amount of work done by an `EventListener` in a Swing program?

- We want to make sure it fits into a lambda expression. (A)
- To be sure it finishes before Swing moves to the next event. (B)
- Swing will prioritize processing events with faster responses. (C)
- Inefficient code will lock up the event dispatch thread. (D)



Lecture 26: Concurrency

CS 2110

November 24, 2026

Today's Learning Outcomes

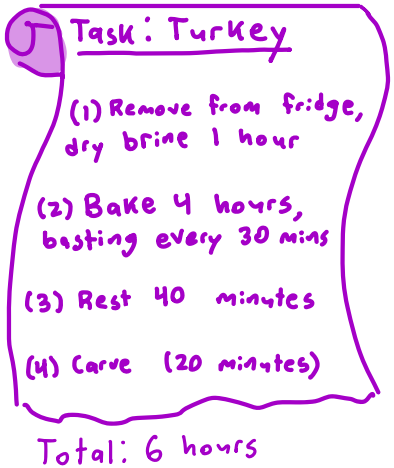
93. Describe advantages and disadvantages of concurrent code.

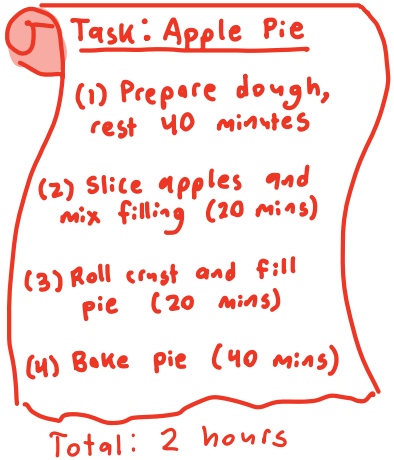
94. Use Java's Thread class to write concurrent code.

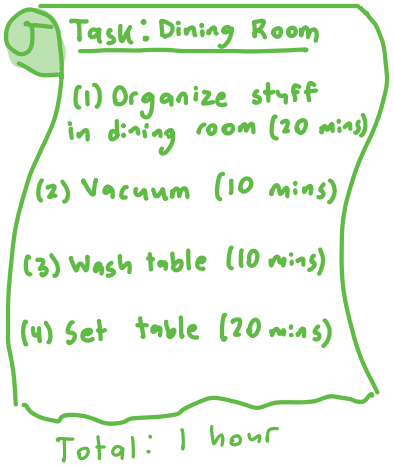
95. Identify race conditions in a given piece of concurrent code and list the possible states the program can be in after it executes.

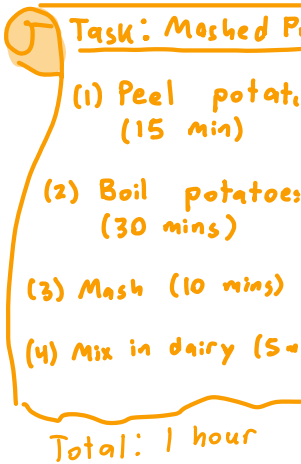
Motivation: Making a Thanksgiving Dinner

Preparing and hosting the meal requires completing a lot of separate tasks:









How can we take care of them efficiently?

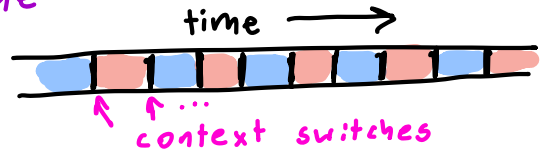
Concurrency

= The ability to have multiple independent procedures in progress at the same time.

We can achieve concurrency in two different ways:

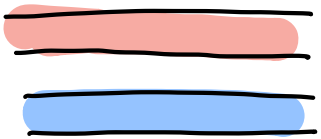
1. Time Slicing

Switching back in forth between multiple tasks, but actively working on one at a time



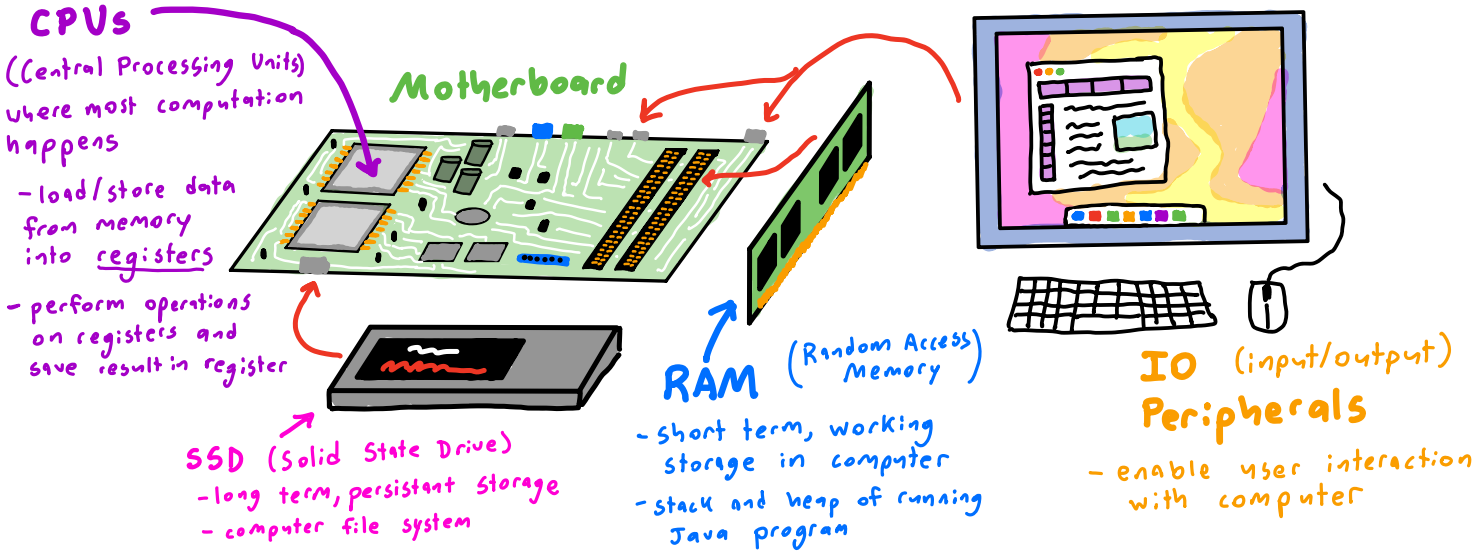
2. Parallelism

Simultaneously carrying out multiple tasks by splitting the work to different resource streams



Computer Hardware (High Level)

Various hardware components serve different functions in your computer and must work together to enable computation.



Operating Systems and Processes

A **process** is a running program on your computer.

The **operating system** is the main program that is responsible for managing the execution of all other processes.

- interfacing with hardware to dispatch input/output events
- initializes and manages all processes running on computer
  - when it creates a new process, the OS allocates it a block of memory (i.e., an address space) that process has exclusive access to
- enforces memory ownership and communication between processes
- schedules computation time on the CPU

Threads

= an abstraction that models a single, sequential execution of instructions that performs work within a process.

a process (e.g. Java program) begins with a single thread and can spawn more to achieve concurrency

- the process ends once all its threads stop (execute their last instruction)
- all threads share access to the same memory

Example, Java swing program has main thread and event dispatch thread

Poll Everywhere

PollEv.com/javabear text **javabear** to 22333



Which is **not** a potential benefit of introducing threads as an additional abstraction layer?

- Threads allow programs to better utilize hardware resources. **(A)** (parallelism)
- Threads enable multiple processes to share memory. **(B)** (no!)
- Threads can speed up the execution of a process. **(C)** (concurrency)
- Threads separate time and space usage concerns in the OS. **(D)** (thread / process)



Coding Demo: Java's Thread Class

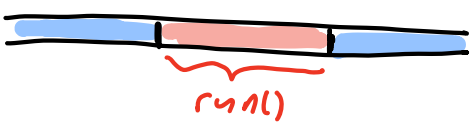


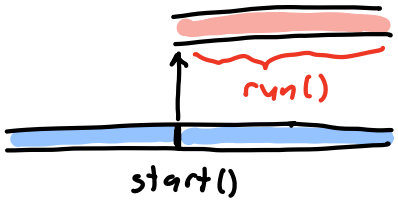
Thread Methods: `run()` and `start()`

Threads are constructed with a `Runnable` argument, a functional interface that models a sequence instructions.

The thread exposes two methods to carry out these instructions:

1. **`run()`** execute the code inside the Thread's runnable on the current execution thread
   * don't ever call this!
2. **`start()`**
   - create a new execution thread
   - call `run()` on the new thread
   - proceed immediately on old thread



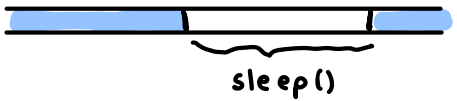


Thread Methods: `sleep()` and `join()`

Sometimes, we may wish to temporarily pause the execution of instructions on one thread. Java offers many methods for doing this:

`sleep()`

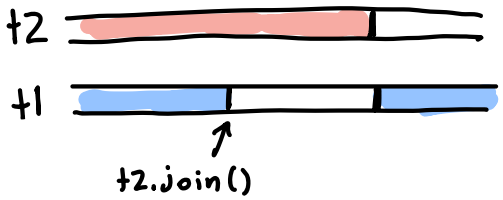
pause execution on this execution thread for the specified amount of time



`join()`

pause execution on this execution thread until the target thread stops

"hey, I'll wait up so you can join up with me"

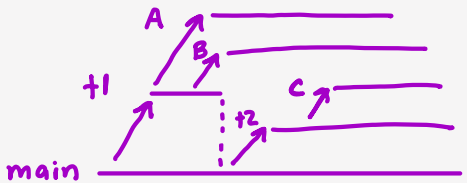


Poll Everywhere

PollEv.com/javabear text **javabear** to 22333

What is the maximum number of threads that can be executing concurrently in this code snippet.

```
public static void main(String[] args) throws InterruptedException {
  Thread t1 = new Thread(() -> {
      new Thread(() -> System.out.print("A")).start();
      new Thread(() -> System.out.print("B")).start();
  });
  Thread t2 = new Thread(() -> {
      new Thread(() -> System.out.print("C")).start();
  });
  t1.start();
  t1.join();
  t2.start();
}
```



1	(A)
2	(B)
3	(C)
4	(D)
5	(E)
6	(F)

Aside: Variable Capture

We often define the `Runnable` parameter inline within the `Thread` constructor using a lambda expression.

The `Runnable.run()` method has no parameters. How can we share/pass information to this new thread?

Recall: Variable capture

we can reference variables in scope in the surrounding method within the definition of a lambda expression

warning: only works for
- static variables
- fields

} lambda expression body will reference same variable when it executes

- "effectively" final local variables } value copied into lambda expression

never reassigned

* trying to capture non-ef local variable will cause compiler error

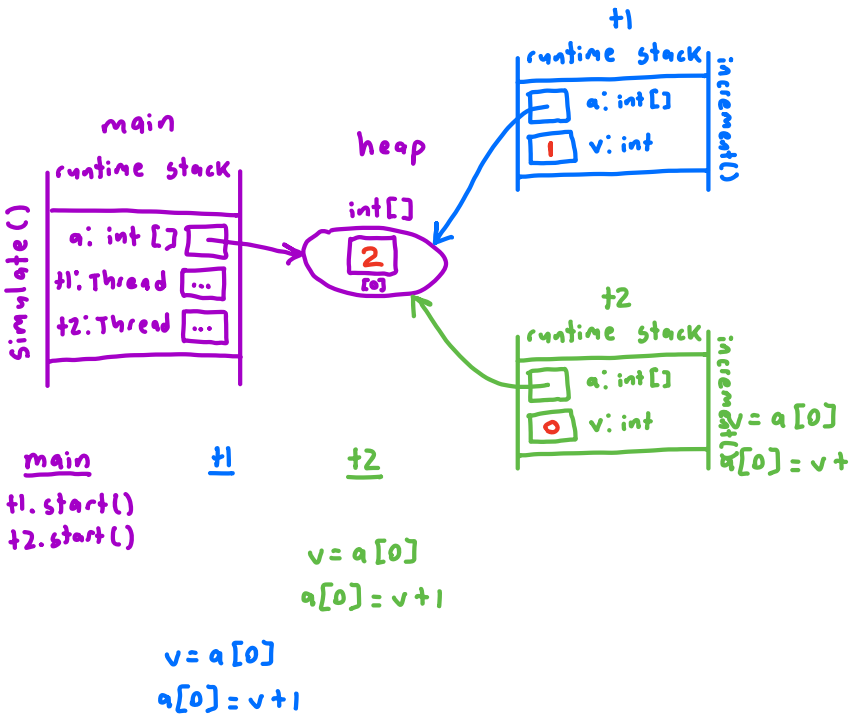
Sharing State Between Threads

```
static void increment(int[] a) {
  int v = a[0];
  a[0] = v + 1;
}

void simulate() throws InterruptedException {
  int[] a = {0}; // shared mutable variable

  Thread t1 = new Thread(() -> increment(a));
  Thread t2 = new Thread(() -> increment(a));

  t1.start(); t2.start();
  t1.join();  t2.join();
  System.out.println(a[0]);
}
```



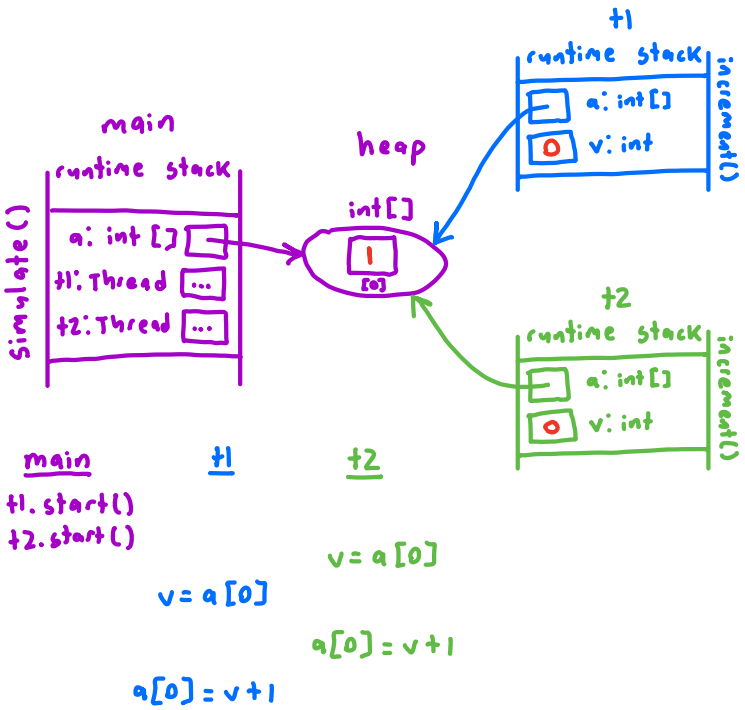
Sharing State Between Threads (v2)

```
static void increment(int[] a) {
  int v = a[0];
  a[0] = v + 1;
}

void simulate() throws InterruptedException {
  int[] a = {0}; // shared mutable variable

  Thread t1 = new Thread(() -> increment(a));
  Thread t2 = new Thread(() -> increment(a));

  t1.start(); t2.start();
  t1.join();  t2.join();
  System.out.println(a[0]);
}
```



Race Conditions

= a phomenon in multi-threaded code where different post-conditions are reached depending on the specific order in which the instructions across multiple threads are executed.

- while the order that the instructions run on one thread is deterministic, we have no guarantees about how these sequences will interleave.
  ↑ since OS can schedule thread execution however it chooses

Race conditions occur when multiple threads attempt to write to a variable within the same window of time
- Last write determines final value
- Delay between reading old value and writing new value

Poll Everywhere

PollEv.com/javabear text **javabear** to 22333

How many different values could be printed at the end of this code snippet?

```
int[] a = {0};
Thread ta = new Thread(() -> a[0] += 1);
Thread tb = new Thread(() -> a[0] -= 1);
Thread tc = new Thread(() -> a[0] += 2);
ta.start(); tb.start(); tc.start();
ta.join();  tb.join();  tc.join();
System.out.print(a[0]);
```

any int between -1 and 3

3	(A)
4	(B)
5	(C)
6	(D)

Poll Everywhere

PollEv.com/javabear text **javabear** to 22333



How many different values could be printed at the end of this code snippet?

```
int[] a = {0};
Thread ta = new Thread(() -> a[0] += 1);
Thread tb = new Thread(() -> a[0] -= 1);
Thread tc = new Thread(() -> a[0] += 2);
ta.start(); tb.start(); tc.start();
ta.join();  tb.join();  tc.join();
System.out.print(a[0]);
```

Example: print 1

ta	tb	tc
read a[0] = 0		
	read a[0] = 0	
assign a[0] = 0+1		
	assign a[0] = 0-1	
		read a[0] = -1
		assign a[0] = -1+2

print a[0] = 1

*Exercise: Can you determine the operation sequences that result in the other 4 print outputs?

Preview: Synchronization

= the process of coordinating the execution of multiple threads, particularly their access and modification of shared state, within a piece of concurrent code.

Sometimes, we'll need to carefully time (i.e., strategically pause) work on different threads to avoid race conditions and achieve the desired behavior.

We'll see how to get this working when we return next week.

For now, we'll strategically pause so you can enjoy your Thanksgiving!