A single computer can simultaneously run several programs.

Several computers can simultaneously access a shared server.

The server can itself be a distributed system with different data and programs on different nodes.

Single-threaded vs. multi-threaded execution:
- Single-threaded: Each program can have several threads of execution.
- Multi-threaded: Several threads can be run on a single processor pipeline. Each thread is run for a short time and then suspended, giving the effect that they are running simultaneously.
If the processor has multiple cores (modern CPUs often have 2, 4, 8 or more cores), the threads may be assigned to different cores (or more generally, to different “hardware threads”) and actually run simultaneously.

Threads in Java

- Threads are instances of class `Thread`  
  - Can create many, but they consume space & time
- The Java Virtual Machine created the initial `Thread` that executes your method `main`
- Threads have a priority  
  - Higher priority Threads are executed preferentially
  - A newly created `Thread` has initial priority equal to the `Thread` that created it (but can change)

### A Java Thread runs a Runnable object

```java
class PrimeRun implements Runnable {
    long a, b;
    PrimeRun(long a, long b) {
        this.a = a; this.b = b;
    }
    public void run() {
        // compute primes
        // in a..b
    }
}
PrimeRun p = new PrimeRun(143, 195);
new Thread(p).start();
```

Method `start()` will call `p`'s method `run()` in the new thread of execution.

### Another way of creating a Thread

```java
class PrimeThread extends Thread {
    long a, b;
    PrimeThread(long a, long b) {
        this.a = a; this.b = b;
    }
    public void run() {
        // compute primes
        // in a..b
    }
}
PrimeThread p = new PrimeThread(143, 195);
p.start();
```

Class `Thread` has methods to allow more control over threads.

### Class Thread has methods to handle threads

You can interrupt a thread, maintain a group of threads, set/change its priority, sleep it for a while, etc.

```
PrimeThread extends Thread, which implements Runnable
```
Race Condition

- Two or more simultaneous threads of execution (concurrency)
- Outcome depends on the exact order in which they are executed
- ... which cannot be predicted in advance
  - Betting on races does not guarantee winnings
  - Two chefs can cook great dishes one after the other, but not if they're trying to simultaneously use the same stove

Race conditions yield unexpected results

Suppose x is initially 0

Thread t1
\[
\text{x} = \text{x} + 1;
\]
... after finishing, \( x \neq 1 \), not 2! Why?

Thread t2
\[
\text{x} = \text{x} + 1;
\]

Synchronization

- Writing correct concurrent programs is very hard
  - Ideally, two threads would never access the same data
    - This is frequently unrealistic
- We need some form of synchronization
  - E.g. ensure a thread completes its read-modify-write sequence on a piece of data before another thread is allowed to touch it
  - E.g. ensure a thread accesses a resource only after another thread has finished accessing it
- There are many methods. We will only look at Java's \texttt{synchronized} keyword.

Fixing the \texttt{x= x + 1} bug

Only one thread can execute this method on a given counter at a time
The synchronized block

```java
Stack<String> s = new Stack<String>();

synchronized(s) {
    // This is a block of code
}
```

Only one thread can be executing a block B synchronized on s at any given time. All other threads trying to execute a block synchronized on s (need not be the same as B) must wait until the first thread finishes executing B.

The synchronized block is a primary tool for eliminating shared data problems. (There are others)

Accessing a stack in a threadsafe way

```java
private Stack<String> s = new Stack<String>();
public void doSomething() {
    String str;
    synchronized(s) {
        if (s.isEmpty()) return;
        str = s.pop();
    }
    // code to do something with str
}
```

- Put critical operations in a synchronized block
- The `Stack` object acts as a lock
- Only one thread can own the lock at a time
- Make synchronized blocks as small as possible

Locking on this, and synchronized methods

```java
public void doSomething() {
    // body
}
```

is equivalent to

```java
public synchronized void doSomething() {
    // body
}
```

A threadsafe Stack<T> class will have

- `public synchronized T pop() { … }`
- `public synchronized void push(…) { … }` etc

Synchronized collections

- Study class Collections and the following methods before working on A8:
  - `synchronizedCollection`
  - `synchronizedSet`
  - `synchronizedSortedSet`
  - `synchronizedList`
  - `synchronizedMap`
  - `synchronizedSortedMap`