



# Java Tutorial

CS 131a

Fall 2014



# Overview

- Classes and methods
  - Interfaces, generics
  - Input and output
  - **Threads**
  - Tips
- 
- Slides online: <http://rmarcus.info/slides.pdf>



# Classes

- All code is contained within a **class**
  - Has same name as file
- Classes can have **methods**
- Special method **main** is an entry point



## Example

```
// Basics.java
public class Basics {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```



## Example

```
// Basics.java  
public class Basics {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

Class name matches file name



## Example

```
// Basics.java
public class Basics {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Entry point method



## Example

```
// Basics.java
public class Basics {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Command line arguments

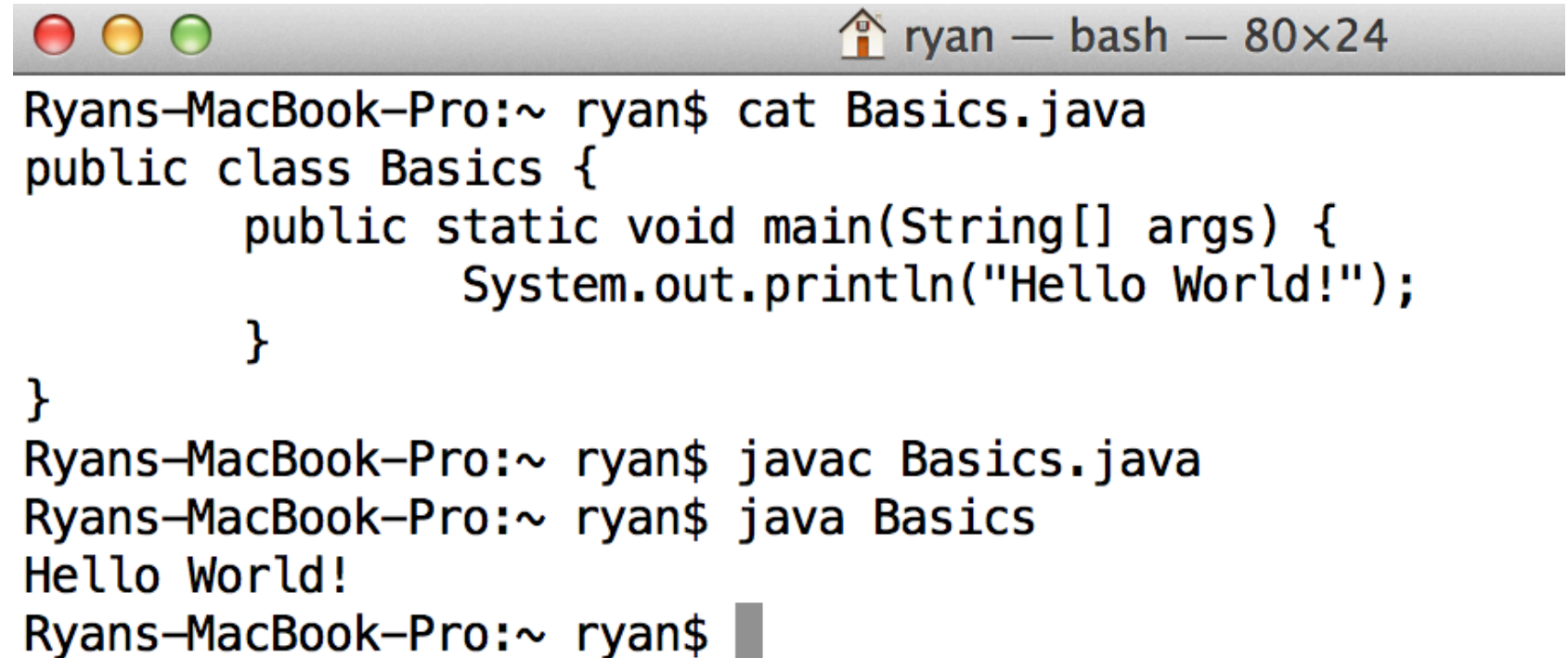


## Example

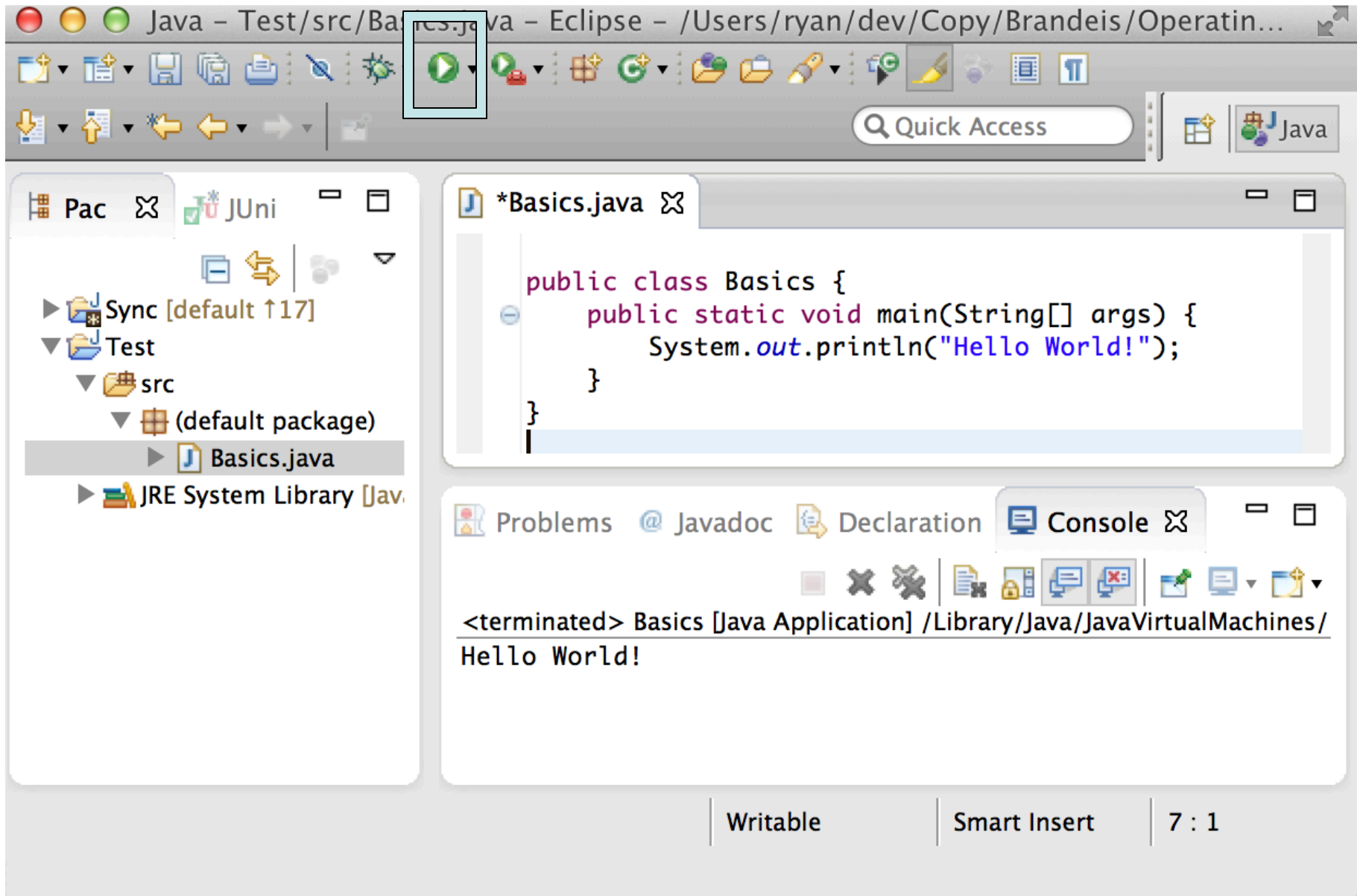
```
// Basics.java
public class Basics {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Prints "Hello World!"



A screenshot of a macOS terminal window. The title bar at the top shows three colored window control buttons (red, yellow, green) on the left and a home icon followed by the text "ryan — bash — 80x24" on the right. The terminal content shows a user named "ryan" at a machine named "Ryans-MacBook-Pro" in the home directory. The user runs "cat Basics.java" which displays the contents of a Java file. Then, the user runs "javac Basics.java" to compile the code. Finally, the user runs "java Basics" which executes the program and outputs "Hello World!". The prompt "ryan\$" is followed by a grey rectangular cursor.

```
Ryans-MacBook-Pro:~ ryan$ cat Basics.java
public class Basics {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
Ryans-MacBook-Pro:~ ryan$ javac Basics.java
Ryans-MacBook-Pro:~ ryan$ java Basics
Hello World!
Ryans-MacBook-Pro:~ ryan$
```





```
// Sample.java
public class Sample {
    private int sampleSize;

    public Sample(int n) {
        this.sampleSize = n;
    }

    public void printReport() {
        System.out.println("n = " + sampleSize);
    }
}
```

```
// Sample.java
public class Sample {
    private int sampleSize;

    public Sample(int n) {
        this.sampleSize = n;
    }

    public void printReport() {
        System.out.println("n = " + sampleSize);
    }
}
```

Class and file name match

```
// Sample.java
public class Sample {
    private int sampleSize;

    public Sample(int n) {
        this.sampleSize = n;
    }

    public void printReport() {
        System.out.println("n = " + sampleSize);
    }
}
```

Class field

```
// Sample.java
public class Sample {
    private int sampleSize;

    public Sample(int n) {
        this.sampleSize = n;
    }

    public void printReport() {
        System.out.println("n = " + sampleSize);
    }
}
```

## Constructor

```
// Sample.java
public class Sample {
    private int sampleSize;

    public Sample(int n) {
        this.sampleSize = n;
    }

    public void printReport() {
        System.out.println("n = " + sampleSize);
    }
}
```

“this” refers to current instance

```
// Sample.java
public class Sample {
    private int sampleSize;

    public Sample(int n) {
        this.sampleSize = n;
    }

    public void printReport() {
        System.out.println("n = " + sampleSize);
    }
}
```

Method signature



```
// Sample.java
public class Sample {
    private int sampleSize;

    public Sample(int n) {
        this.sampleSize = n;
    }

    public void printReport() {
        System.out.println("n = " + sampleSize);
    }
}
```

Field access



```
// Scope.java
public class Scope {
    private int n;
    public Scope() {
        n = 5;
    }
    public void printN() {
        int n = 1000;
        System.out.println(n);
        System.out.println(this.n);
    }
}
```

## Scope

```
// Scope.java
public class Scope {
    private int n;
    public Scope() {
        n = 5;
    }
    public void printN() {
        int n = 1000;
        System.out.println(n);
        System.out.println(this.n);
    }
}
```

## Scope



```
// Scope.java
```

```
public class Scope {  
    private int n;  
    public Scope() {  
        n = 5;  
    }  
    public void printN() {  
        int n = 1000;  
        System.out.println(n);  
        System.out.println(this.n);  
    }  
}
```

Output:

1000

5

Scope



# Interfaces

- An **interface** is a contract
- Set of methods that a class will implement
- Also a **type**

```
// Blog.java  
public interface Blog {  
    public String getBlogName();  
    public String getAuthorName();  
}
```

```
// Blog.java  
public interface Blog {  
    public String getBlogName();  
    public String getAuthorName();  
}
```

Same name as the file

```
// Blog.java  
public interface Blog {  
    public String getBlogName();  
    public String getAuthorName();  
}
```

“interface” instead of class



```
// Blog.java  
public interface Blog {  
    public String getBlogName();  
    public String getAuthorName();  
}
```

Signature only! No body!

```
// HuffPost.java
public class HuffPost implements Blog {
    @Override
    public String getBlogName() {
        return "The Huffington Post";
    }

    @Override
    public String getAuthorName() {
        return "A bunch of liberals";
    }
}
```

```
// HuffPost.java
public class HuffPost implements Blog {
    @Override
    public String getBlogName() {
        return "The Huffington Post";
    }

    @Override
    public String getAuthorName() {
        return "A bunch of liberals";
    }
}
```

Class name and file name match

```
// HuffPost.java
public class HuffPost implements Blog {
    @Override
    public String getBlogName() {
        return "The Huffington Post";
    }

    @Override
    public String getAuthorName() {
        return "A bunch of liberals";
    }
}
```

“implements” clause

```
// HuffPost.java
public class HuffPost implements Blog {
    @Override
    public String getBlogName() {
        return "The Huffington Post";
    }

    @Override
    public String getAuthorName() {
        return "A bunch of liberals";
    }
}
```

Optional annotation

```
// HuffPost.java
public class HuffPost implements Blog {
    @Override
    public String getBlogName() {
        return "The Huffington Post";
    }

    @Override
    public String getAuthorName() {
        return "A bunch of liberals";
    }
}
```

Matching signature with bodies

```
// HuffPost.java
public class HuffPost implements Blog {
    @Override
    public String getBlogName() {
        return "The Huffington Post";
    }

    @Override
    public String getAuthorName() {
        return "A bunch of liberals";
    }
}
```

Methods implemented

```
// GG.java
public class GG implements Blog {
    @Override
    public String getBlogName() {
        return "Gossip Girl";
    }

    @Override
    public String getAuthorName() {
        return "One secret I'll never tell";
    }
}
```

Another implementing class





```
public void test() {  
    Blog b1 = new HuffPost();  
    Blog b2 = new GG();  
  
    ArrayList<Blog> br = new ArrayList<Blog>();  
    br.add(b1);  
    br.add(b2);  
  
    System.out.println(br.get(0).getBlogName());  
    System.out.println(br.get(1).getAuthorName());  
}
```



```
public void test() {  
    Blog b1 = new HuffPost();  
    Blog b2 = new GG();  
  
    ArrayList<Blog> br = new ArrayList<Blog>();  
    br.add(b1);  
    br.add(b2);  
  
    System.out.println(br.get(0).getBlogName());  
    System.out.println(br.get(1).getAuthorName());  
}
```

Implementing classes are typed



```
public void test() {  
    Blog b1 = new HuffPost();  
    Blog b2 = new GG();  
  
    ArrayList<Blog> br = new ArrayList<Blog>();  
    br.add(b1);  
    br.add(b2);  
  
    System.out.println(br.get(0).getBlogName());  
    System.out.println(br.get(1).getAuthorName());  
}
```

Can be added to collections



```
public void test() {  
    Blog b1 = new HuffPost();  
    Blog b2 = new GG();  
  
    ArrayList<Blog> br = new ArrayList<Blog>();  
    br.add(b1);  
    br.add(b2);  
  
    System.out.println(br.get(0).getBlogName());  
    System.out.println(br.get(1).getAuthorName());  
}
```

Can use methods declared in interface



```
public void test() {
```

```
    Blog b1 = new HuffPost();
```

```
    Blog b2 = new GG();
```

```
    ArrayList<Blog> br = new ArrayList<Blog>();
```

```
    br.add(b1);
```

```
    br.add(b2);
```

```
    System.out.println(br.get(0).getBlogName());
```

```
    System.out.println(br.get(1).getAuthorName());
```

```
}
```

Output:

The Huffington Post

One secret I'll never tell



# Input and Output

- “Standard” in and out
  - `System.out` and `System.in`
- File input and output
  - For refresher, go here:
  - <http://docs.oracle.com/javase/tutorial/essential/io/>



```
public void test() {  
    Scanner in = new Scanner(System.in);  
    String name = in.nextLine();  
    System.out.println("Hello, " + name + "!");  
}
```

```
public void test() {  
    Scanner in = new Scanner(System.in);  
    String name = in.nextLine();  
    System.out.println("Hello, " + name + "!");  
}
```

Create a new scanner on stdin



```
public void test() {  
    Scanner in = new Scanner(System.in);  
    String name = in.nextLine();  
    System.out.println("Hello, " + name + "!");  
}
```

Can be any InputStream

```
public void test() {  
    Scanner in = new Scanner(System.in);  
    String name = in.nextLine();  
    System.out.println("Hello, " + name + "!");  
}
```

Block until user hits "enter"

```
public void test() {  
    Scanner in = new Scanner(System.in);  
    String name = in.nextLine();  
    System.out.println("Hello, " + name + "!");  
}
```

Greet the user

# Threads

- Running two things at once
- Many classes provided to make your life easier
- Synchronization issues



# Threads

- Running two things at once
  - Sort of
- Many classes provided to make your life easier
- Synchronization issues





## Creating Threads

- Thread class, requires a Runnable
- Runnable (interface) requires one method, `run()`



## Creating Threads

- Thread class, requires a Runnable
  - Runnable (interface) requires one method, `run()`
- 
1. Create a new Runnable `r`
  2. Create a new thread `t`, `new Thread(r)`
  3. Start the thread, `t.start()`
  4. Wait for it, `t.join()`



```
public class GG implements Runnable {  
    @Override  
    public void run() {  
        System.out.print("Welcome back,");  
        System.out.println("Upper East Siders...");  
  
        // make some bad puns, process data, etc.  
  
        System.out.println("XOXO GG");  
    }  
}
```





```
public class GG implements Runnable {  
    @Override  
    public void run() {  
        System.out.print("Welcome back,");  
        System.out.println("Upper East Siders...");  
  
        // make some bad puns, process data, etc.  
  
        System.out.println("XOXO GG");  
    }  
}
```

## Implementing Runnable



```
public class GG implements Runnable {  
    @Override  
    public void run() {  
        System.out.print("Welcome back,");  
        System.out.println("Upper East Siders...");  
  
        // make some bad puns, process data, etc.  
  
        System.out.println("XOXO GG");  
    }  
}
```

## Optional annotation



```
public class GG implements Runnable {  
    @Override  
    public void run() {  
        System.out.print("Welcome back,");  
        System.out.println("Upper East Siders...");  
  
        // make some bad puns, process data, etc.  
  
        System.out.println("XOXO GG");  
    }  
}
```

As required by Runnable



```
public class GG implements Runnable {  
    @Override  
    public void run() {  
        System.out.print("Welcome back,");  
        System.out.println("Upper East Siders...");  
  
        // make some bad puns, process data, etc.  
  
        System.out.println("XOXO GG");  
    }  
}
```

Code that will execute in the thread



```
public class HuffPost implements Runnable {  
    @Override  
    public void run() {  
        System.out.print("Blah blah blah ");  
        System.out.println("Obama.");  
  
        // be insightful but kinda cheeky  
  
        System.out.println("MITT ROMNEY");  
    }  
}
```

## Another Runnable



```
public void runThread() {  
    Runnable r1 = new GG();  
    Runnable r2 = new HuffPost();  
  
    Thread t1 = new Thread(r1);  
    Thread t2 = new Thread(r2);  
  
    t1.start();  
    t2.start();  
  
    t1.join();  
    t2.join();  
}
```



```
public void runThread() {  
    Runnable r1 = new GG();  
    Runnable r2 = new HuffPost();  
  
    Thread t1 = new Thread(r1);  
    Thread t2 = new Thread(r2);  
  
    t1.start();  
    t2.start();  
  
    t1.join();  
    t2.join();  
}
```

Create two blogs (Runnable)



```
public void runThread() {  
    Runnable r1 = new GG();  
    Runnable r2 = new HuffPost();  
  
    Thread t1 = new Thread(r1);  
    Thread t2 = new Thread(r2);  
  
    t1.start();  
    t2.start();  
  
    t1.join();  
    t2.join();  
}
```

Create two threads





```
public void runThread() {  
    Runnable r1 = new GG();  
    Runnable r2 = new HuffPost();  
  
    Thread t1 = new Thread(r1);  
    Thread t2 = new Thread(r2);  
  
    t1.start();  
    t2.start();  
  
    t1.join();  
    t2.join();  
}
```

Start both threads



```
public void runThread() {  
    Runnable r1 = new GG();  
    Runnable r2 = new HuffPost();  
  
    Thread t1 = new Thread(r1);  
    Thread t2 = new Thread(r2);  
  
    t1.start();  
    t2.start();  
  
    t1.join();  
    t2.join();  
}
```

Wait for  $t_1$  to finish



```
public void runThread() {  
    Runnable r1 = new GG();  
    Runnable r2 = new HuffPost();  
  
    Thread t1 = new Thread(r1);  
    Thread t2 = new Thread(r2);  
  
    t1.start();  
    t2.start();  
  
    t1.join();  
    t2.join();  
}
```

Wait for  $t_2$  to finish



## Many possible outputs

### **Output 1**

Welcome back, Upper East Siders  
Blah blah blah Obama.  
XOXO GG  
MITT ROMNEY

### **Output 2**

Blah blah blah Obama.  
Welcome back, Upper East Siders  
XOXO GG  
MITT ROMNEY

### **Output 3**

Blah blah blah Obama.  
Welcome back, Upper East Siders  
MITT ROMNEY  
XOXO GG

### **Output 4**

Blah blah blah Upper East Siders.  
Welcome back, Obama.  
MITT ROMNEY  
XOXO GG

**... and many more ...**

# Problems from Threads



- Race conditions
- Unstable ordering
- Starvation
- Deadlocks

# But, we have a partial solution!

- Locks!

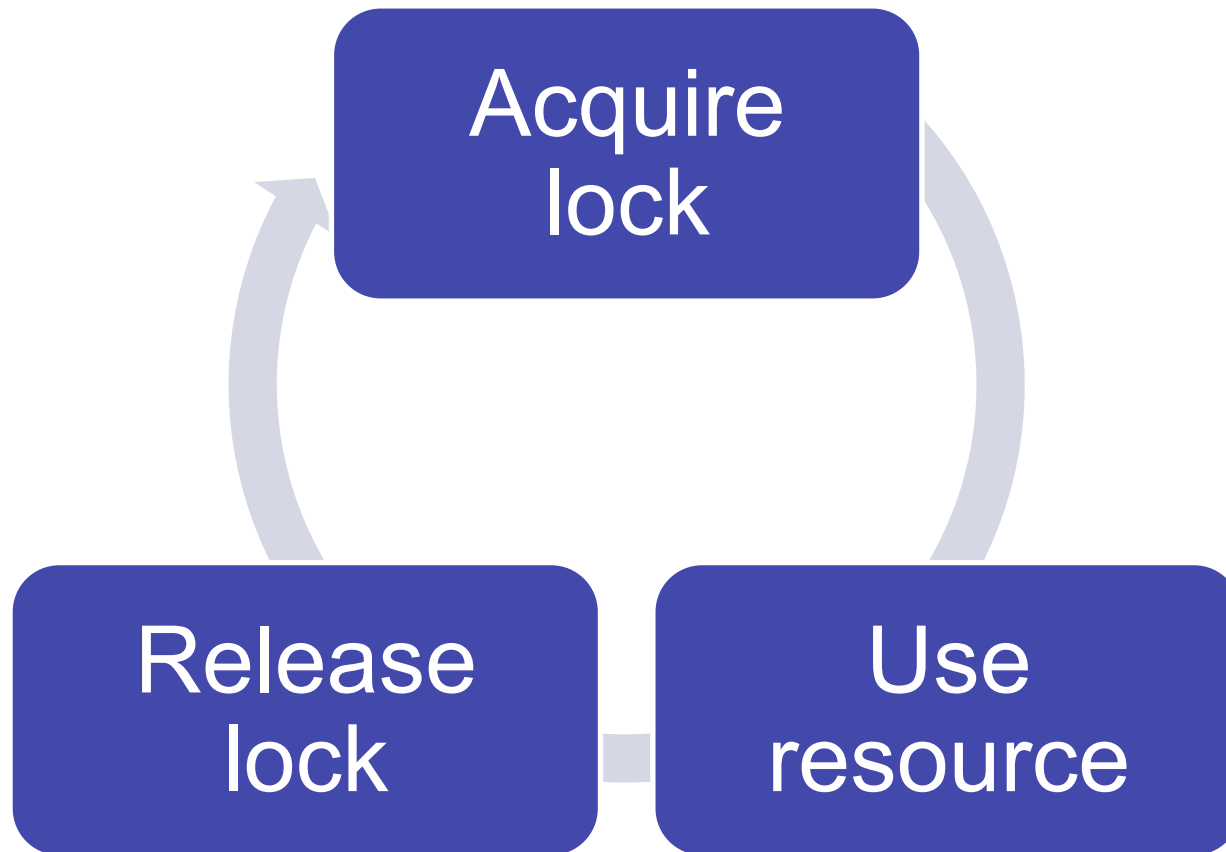


# Locks



- Mutual exclusion
- `acquire()`
- `release()`

# Acquire-Use-Release Loop







```
public class GG implements Runnable {  
    @Override  
    public void run() {  
        synchronized (System.out) {  
            System.out.print("Welcome back,");  
            System.out.println("Upper East Siders...");  
  
            // make some bad puns, process data, etc.  
  
            System.out.println("XOXO GG");  
        }  
    }  
}
```



```
public class GG implements Runnable {  
    @Override  
    public void run() {  
        synchronized (System.out) {  
            System.out.print("Welcome back,");  
            System.out.println("Upper East Siders...");  
  
            // make some bad puns, process data, etc.  
  
            System.out.println("XOXO GG");  
        }  
    }  
}
```

“synchronized” block



```
public class GG implements Runnable {  
    @Override  
    public void run() {  
        synchronized (System.out) {  
            System.out.print("Welcome back,");  
            System.out.println("Upper East Siders...");  
  
            // make some bad puns, process data, etc.  
  
            System.out.println("XOXO GG");  
        }  
    }  
}
```

Object to lock on (acquire)

```
public class GG implements Runnable {  
    @Override  
    public void run() {  
        synchronized (System.out) {  
            System.out.print("Welcome back,");  
            System.out.println("Upper East Siders...");  
  
            // make some bad puns, process data, etc.  
  
            System.out.println("XOXO GG");  
        }  
    }  
}
```

“Critical section”



```
public class HuffPost implements Runnable {  
    @Override  
    public void run() {  
        synchronized(System.out) {  
            System.out.print("Blah blah blah ");  
            System.out.println("Obama.");  
  
            // be insightful but kinda cheeky  
  
            System.out.println("MITT ROMNEY");  
        }  
    }  
}
```

Another Runnable



```
public void runThread() {  
    Runnable r1 = new GG();  
    Runnable r2 = new HuffPost();  
  
    Thread t1 = new Thread(r1);  
    Thread t2 = new Thread(r2);  
  
    t1.start();  
    t2.start();  
  
    t1.join();  
    t2.join();  
}
```

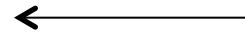


## Two possible outputs

### Output 1

Welcome back, Upper East Siders  
XOXO GG  
Blah blah blah Obama.  
MITT ROMNEY

**t1 gets lock  
first**



### Output 2

Blah blah blah Obama.  
MITT ROMNEY  
Welcome back, Upper East Siders  
XOXO GG

**t2 gets lock  
first**



# Locking Multiple Objects

- What if you need more than just `System.out`?
- Need to `acquire()` and `release()` multiple resources







```
public class ReadWrite1 implements Runnable {
    @Override
    public void run() {
        synchronized (System.out) {
            synchronized (System.in) {
                Scanner sc = new Scanner(System.in);
                String name = sc.nextLine();
                System.out.println("Hello " + name);
            }
        }
    }
}
```



```
public class ReadWrite1 implements Runnable {  
    @Override  
    public void run() {  
        synchronized (System.out) {  
            synchronized (System.in) {  
                Scanner sc = new Scanner(System.in);  
                String name = sc.nextLine();  
                System.out.println("Hello " + name);  
            }  
        }  
    }  
}
```

## Named ReadWrite1



```
public class ReadWrite1 implements Runnable {  
    @Override  
    public void run() {  
        synchronized (System.out) {  
            synchronized (System.in) {  
                Scanner sc = new Scanner(System.in);  
                String name = sc.nextLine();  
                System.out.println("Hello " + name);  
            }  
        }  
    }  
}
```

## Acquire System.out

```
public class ReadWrite1 implements Runnable {
    @Override
    public void run() {
        synchronized (System.out) {
            synchronized (System.in) {
                Scanner sc = new Scanner(System.in);
                String name = sc.nextLine();
                System.out.println("Hello " + name);
            }
        }
    }
}
```

Acquire System.in



```
public class ReadWrite1 implements Runnable {  
    @Override  
    public void run() {  
        synchronized (System.out) {  
            synchronized (System.in) {  
                Scanner sc = new Scanner(System.in);  
                String name = sc.nextLine();  
                System.out.println("Hello " + name);  
            }  
        }  
    }  
}
```

Greet the user



```
public class ReadWrite2 implements Runnable {
    @Override
    public void run() {
        synchronized (System.in) {
            synchronized (System.out) {
                Scanner sc = new Scanner(System.in);
                String name = sc.nextLine();
                System.out.println("Salve " + name);
            }
        }
    }
}
```



```
public class ReadWrite2 implements Runnable {  
    @Override  
    public void run() {  
        synchronized (System.in) {  
            synchronized (System.out) {  
                Scanner sc = new Scanner(System.in);  
                String name = sc.nextLine();  
                System.out.println("Salve " + name);  
            }  
        }  
    }  
}
```

## Named ReadWrite2



```
public class ReadWrite2 implements Runnable {  
    @Override  
    public void run() {  
        synchronized (System.in) {  
            synchronized (System.out) {  
                Scanner sc = new Scanner(System.in);  
                String name = sc.nextLine();  
                System.out.println("Salve " + name);  
            }  
        }  
    }  
}
```

Acquire System.in





```
public class ReadWrite2 implements Runnable {  
    @Override  
    public void run() {  
        synchronized (System.in) {  
            synchronized (System.out) {  
                Scanner sc = new Scanner(System.in);  
                String name = sc.nextLine();  
                System.out.println("Salve " + name);  
            }  
        }  
    }  
}
```

## Acquire System.out



```
public class ReadWrite2 implements Runnable {  
    @Override  
    public void run() {  
        synchronized (System.in) {  
            synchronized (System.out) {  
                Scanner sc = new Scanner(System.in);  
                String name = sc.nextLine();  
                System.out.println("Salve " + name);  
            }  
        }  
    }  
}
```

Greet the user (in Latin!)



## What happens?

- Write a driver that starts ReadWrite1, then starts ReadWrite2.
- Run the code, type my name, press enter.
- What happens?



## What happens?

- Write a driver that starts ReadWrite1, then starts ReadWrite2.
- Run the code, type my name, press enter.
- What happens?
- Nothing! What went wrong?

# What's going on?

## ReadWrite1

- Acquire System.out
- Acquire System.in
- Greet user
- Release System.in
- Release System.out

Out



## ReadWrite2

- Acquire System.in
- Acquire System.out
- Greet user
- Release System.out
- Release System.in

In



# What's going on?

## ReadWrite1

- Acquire System.out
- Acquire System.in
- Greet user
- Release System.in
- Release System.out

Out



## ReadWrite2

- Acquire System.in
- Acquire System.out
- Greet user
- Release System.out
- Release System.in

In



# What's going on? CONTEXT SWITCH!

## ReadWrite1

- Acquire System.out
- Acquire System.in
- Greet user
- Release System.in
- Release System.out

Out



## ReadWrite2

- Acquire System.in
- Acquire System.out
- Greet user
- Release System.out
- Release System.in

In



# What's going on?

## ReadWrite1

- Acquire System.out
- Acquire System.in
- Greet user
- Release System.in
- Release System.out

Out



## ReadWrite2

- Acquire System.in
- Acquire System.out
- Greet user
- Release System.out
- Release System.in

In





# What's going on?

## ReadWrite1

- *Acquire System.out*
- *Acquire System.in*
- *Greet user*
- *Release System.in*
- *Release System.out*

Out



## ReadWrite2

- *Acquire System.in*
- *Acquire System.out*
- *Greet user*
- *Release System.out*
- *Release System.in*

In



# What's going on?

## CONTEXT SWITCH!

### ReadWrite1

- *Acquire System.out*
- Acquire System.in
- Greet user
- Release System.in
- Release System.out

Out



### ReadWrite2

- *Acquire System.in*
- *Acquire System.out*
- Greet user
- Release System.out
- Release System.in

In



# What's going on?

## ReadWrite1

- *Acquire System.out*
- *Acquire System.in*
- Greet user
- Release System.in
- Release System.out

Out



## ReadWrite2

- *Acquire System.in*
- *Acquire System.out*
- Greet user
- Release System.out
- Release System.in

In



## What's going on?

### ReadWrite1

- *Acquire System.out*
- *Acquire System.in*
- Greet user
- Release System.in
- Release System.out

Out

**DEAD LOCK!**



### ReadWrite2

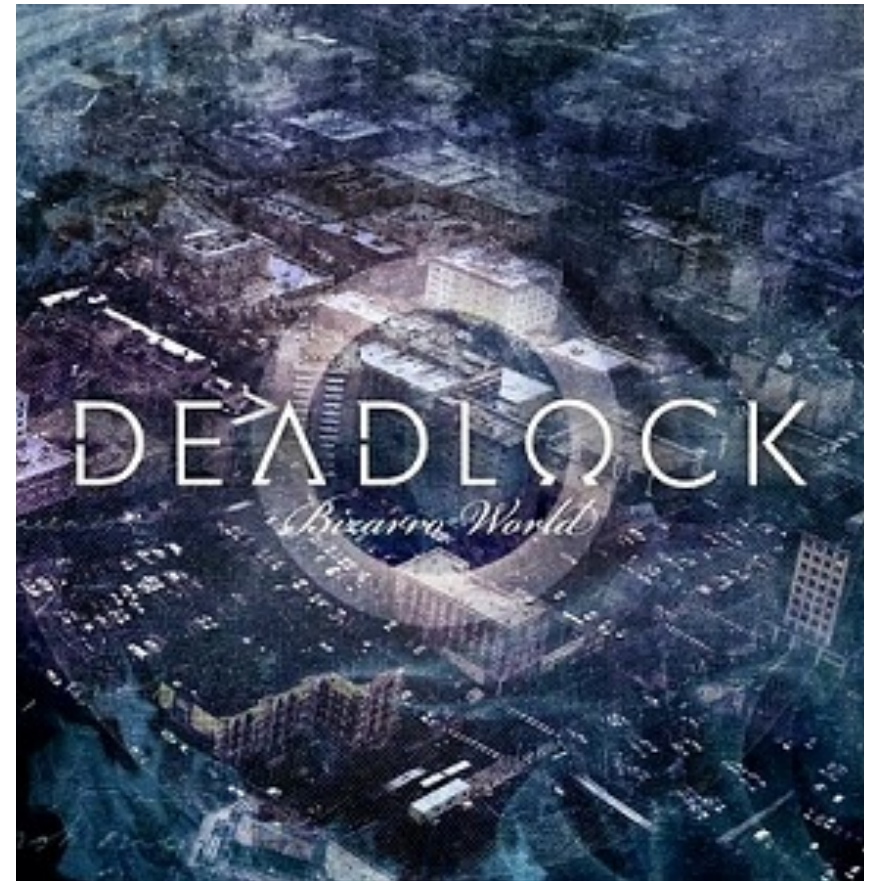
- *Acquire System.in*
- *Acquire System.out*
- Greet user
- Release System.out
- Release System.in

In



# Deadlock

“A deadlock is a situation in which two or more competing actions are each waiting for the other to finish, and thus neither ever does. “



(not the German metal band)



## How can we fix it?

### **ReadWrite1**

- Acquire System.out
- Acquire System.in
- Greet user
- Release System.in
- Release System.out

### **ReadWrite2**

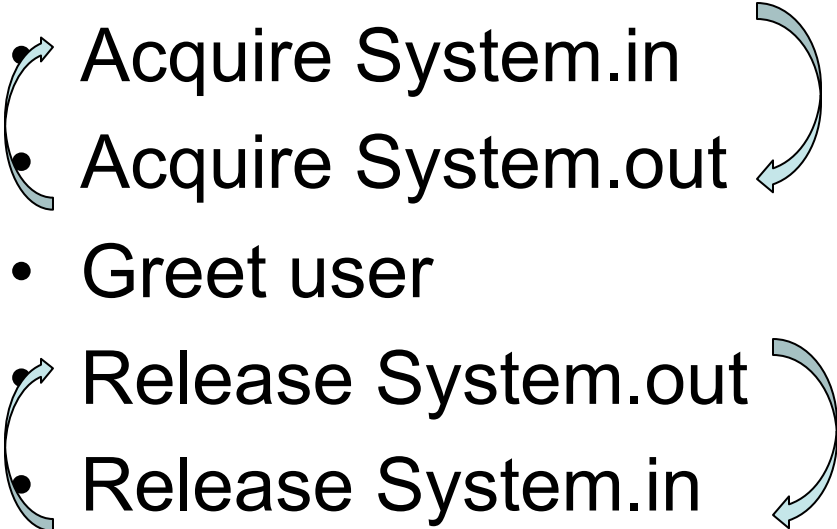
- Acquire System.in
- Acquire System.out
- Greet user
- Release System.out
- Release System.in

## How can we fix it?

### ReadWrite1

- Acquire System.out
- Acquire System.in
- Greet user
- Release System.in
- Release System.out

### ReadWrite2

- Acquire System.in
  - Acquire System.out
  - Greet user
  - Release System.out
  - Release System.in
- 



## How can we fix it?

### **ReadWrite1**

- Acquire System.out
- Acquire System.in
- Greet user
- Release System.in
- Release System.out

### **ReadWrite2**

- Acquire System.out
- Acquire System.in
- Greet user
- Release System.in
- Release System.out

**Lesson: always lock objects in the same order**



## Special Synchronized Methods

- You can also use “synchronized” in a method signature

```
public synchronized void myMethod() {  
    // code  
}
```

- Same as:

```
Object myMethodLockObj;  
public void myMethod() {  
    synchronized (myMethodLockObj) {  
  
    }  
}
```



# Special Java Concurrent Classes

- Take a look at `java.util.concurrent`
- For PA1, you must not use `synchronized` and must use ONLY `ArrayBlockingQueue`.
- PA2, you must use ONLY `synchronized` and must NOT use `java.util.concurrent.*`



# Godspeed

- PA1 Tutorial TOMORROW (9/16) at 7pm  
HERE.
  - **Highly recommended**
- Program early, program often
  - **Do not put it off. You will regret it and we will show no mercy.**
- Concurrency is **hard**
  - Utilize us (TA's) and your peers during office hours or via LATTE forums.