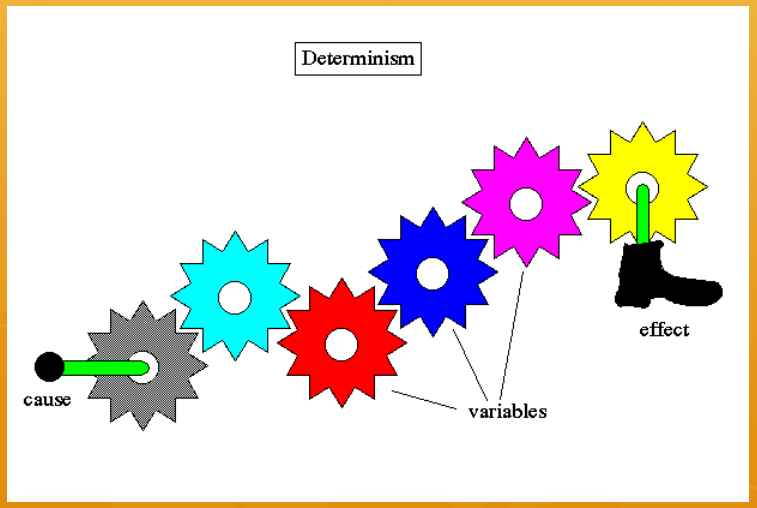
Deterministic testing
in a less deterministic world

Definitions

- Determinism:
- Random:
- Race Condition:

Definitions

- **Determinism:** The philosophy that everything has a cause, and that a particular cause leads to a unique effect.



- **Random:**
- **Race Condition:**

Definitions

- **Determinism:** .
- **Random:** lack of pattern or predictability in events
- **Race Condition:**

Definitions

- **Determinism:**
- Pseudo**random:** Algorithm that generates approximately random #s.
- **Race Condition:**

Definitions

- **Determinism:**
- Pseudo**random:**
- **Race Condition:** The output is dependent on the sequence or timing of other uncontrollable events.

Time then and now

- **Old way:**
  Do NOT do this!

```
public void badMethod(){
    Date now = new Date();
    if (DEADLINE.before(now)){
        //do something...
    }
}
```

Time then and now

- **Old way:**
  Do NOT do this!

```
public void badMethod(){
    Date now = new Date();
    if (DEADLINE.before(now)){
        //do something...
    }
}
```

- **Better way:**
  Java 8 introduced **LocalTime**

```
public void betterMethod(){
    LocalTime now = LocalTime.now();
    if (DEADLINE.isBefore(now)){
        //do something...
    }
}
```

Time then and now

- **Old way:**
  Do NOT do this!

```
public void badMethod(){
    Date now = new Date();
    if (DEADLINE.before(now)){
        //do something...
    }
}
```

- **Better way:**
  Java 8 introduced
  LocalTime and **Clock**

```
public void betterMethod(){
    LocalTime now = LocalTime.now();
    if (DEADLINE.isBefore(now)){
        //do something...
    }
}
```

```
public void bestMethod(Clock clock){
    LocalTime now = LocalTime.now(clock);
    if (DEADLINE.isBefore(now)){
        //do something...
    }
}
```

Time then and now

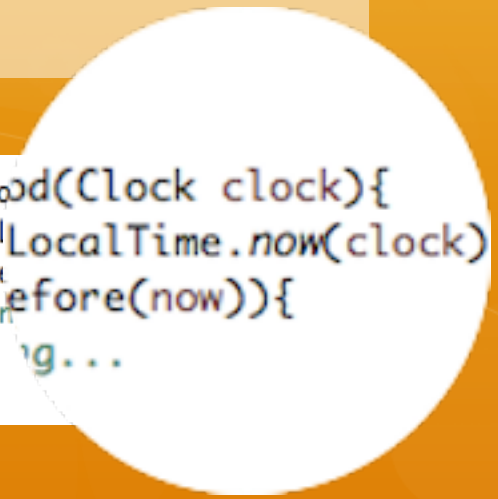
- **Old way:**
  Do NOT do this!

```
public void badMethod(){
    Date now = new Date();
    if (DEADLINE.before(now)){
        //do something...
    }
}
```

- **Better way:**
  Java 8 introduced
  LocalTime and **Clock**

```
public void betterMethod(){
    LocalTime now = LocalTime.now();
    if (DEADLINE.isBefore(now)){
        //do something...
    }
}
```

```
public void bestMetho
    LocalTime now = l
    if (DEADLINE.isBe
        //do somethin
    }
}
```



Embrace Pseudorandom

- **Naïve way:**
  Do NOT do this!

```
public void badRandom(){
    Random die = new Random();
    int roll = die.nextInt(6);
    switch(roll){
        case 0:  //do something
        case 1:  //do something
        case 2:  //do something
        case 3:  //do something
        case 4:  //do something
        case 5:  //do something
        default:
            throw new IllegalArgumentException(
                    "cannot happen: " + roll);
    }
}
```

Embrace Pseudorandom

- **Naïve way:**
  Do NOT do this!

```
public void badRandom(){
    Random die = new Random();
    int roll = die.nextInt(6);
    switch(roll){
        case 0:  //do something
        case 1:  //do something
        case 2:  //do something
        case 3:  //do something
        case 4:  //do something
        case 5:  //do something
        default:
            throw new IllegalArgumentException(
                    "cannot happen: " + roll);
    }
}
```

- **Better way:**
  Pass a **seed** to Random

```
public void betterRandom(long seed){
    Random die = new Random(seed);
    int roll = die.nextInt(6);
    switch(roll){
        case 0:  //do something
        case 1:  //do something
        case 2:  //do something
        case 3:  //do something
        case 4:  //do something
        case 5:  //do something
        default:
            throw new IllegalArgumentException(
                    "cannot happen: " + roll);
    }
}
```

Embrace Pseudo**random**

- **Naïve way:**
  Do NOT do this!

```
public void badRandom(){
    Random die = new Random();
    int roll = die.nextInt(6);
    switch(roll){
        case 0:  //do something
        case 1:  //do something
        case 2:  //do something
        case 3:  //do something
        case 4:  //do something
        case 5:  //do something
        default:
            throw new IllegalArgumentException(
                    "cannot happen: " + roll);
    }
}
```

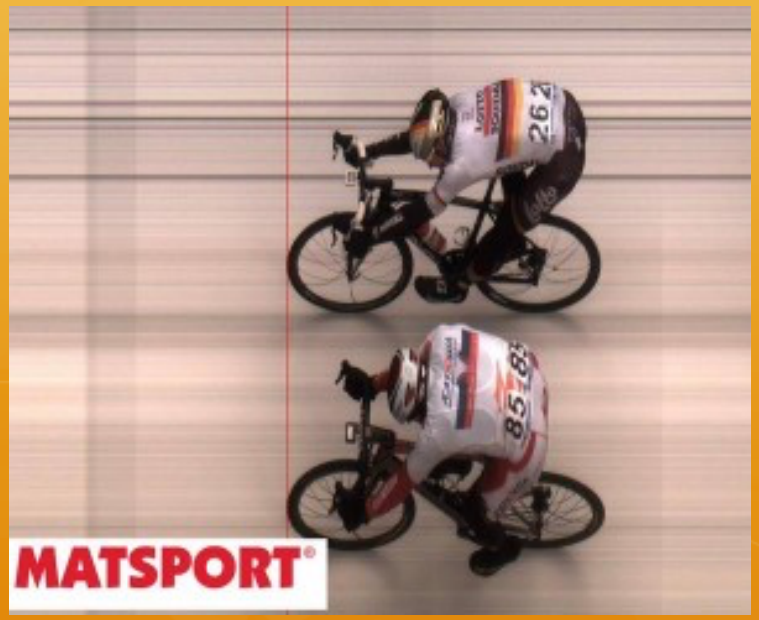
- **Better way:**
  Pass a **seed** to R

```
public void betterRandom(
    Random die = new Ra
    int roll = die.next
    switch(roll){
        case 0:  //do so
        case 1:  //do som
        case 2:  //do somet
        case 3:  //do somethin
        case 4:  //do something
        case 5:  //do something
        default:
            throw new IllegalArgumentException(
                    "cannot happen: " + roll);
    }
}
```

```
random(long seed){
w Random(seed);
nextInt(6);
omething
```

Control **Race Conditions**

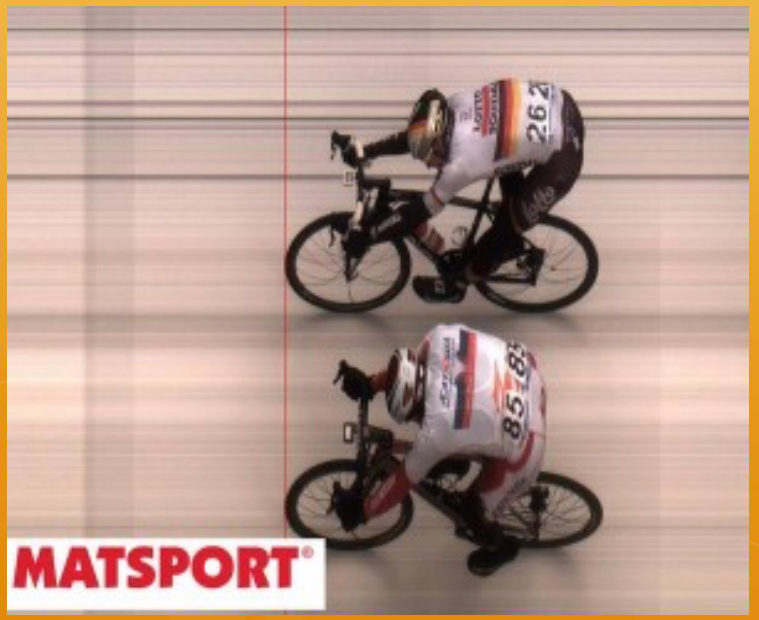
- **Thread Safety:**
  Many strategies to avoid corrupting data that is manipulated by multiple threads.
  Use:
  java.util.concurrent



- **Reproducible:**
- **Producer-Consumer:**

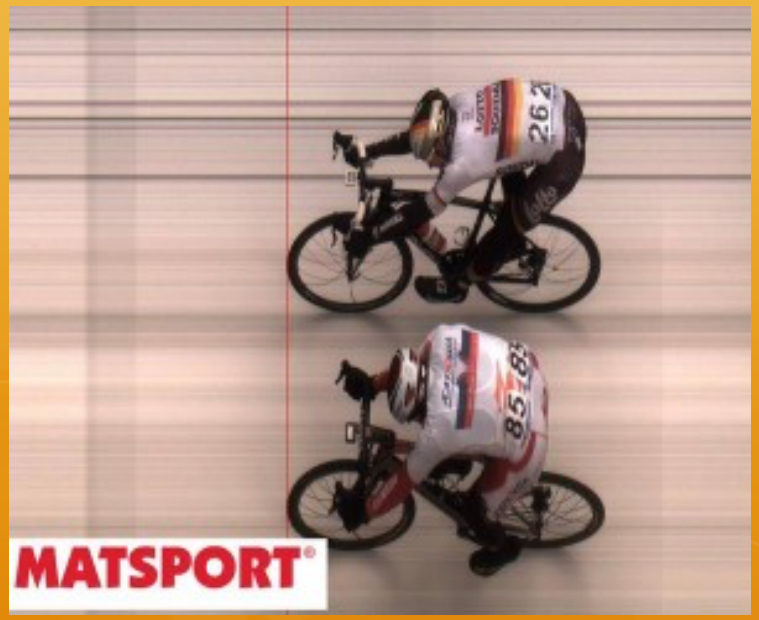
Control **Race Conditions**

- **Thread Safety:**
- **Reproducible:** More important for this discussion.
- **Producer-Consumer:**



Control **Race Conditions**

- **Thread Safety:**
- **Reproducible:**
- **Producer-Consumer:** One thread adds to a Queue, the other removes.



Control **Race Conditions**

- **Producer-Consumer:** Simple Example.

```
/** WARNING: do not use as synchronization monitor */
public class CommandManager {

    private List<Command> commandsForNextCycle = new LinkedList<Command>();

    public synchronized void add(Command command) {
        commandsForNextCycle.add(command);
    }

    public synchronized List<Command> processAll() {
        List<Command> commandsToReturn = commandsForNextCycle;
        //cutoff:  from this point on, we are collecting commands for the next cycle.
        commandsForNextCycle = new LinkedList<Command>();
        return commandsToReturn;
    }
}
```

Bring it all **Together**

- **FleetingDataAndServices:**
  I like to create a single class that holds all of these critical Services together.
- **What Data?**
  Commands.
- **What Services?**
  Clock,
  Random Seed.

Bring it all **Together**

- **FleetingDataAndServices:**
  I like to create a single class that holds all of these critical Services together.

- **What Data?**
  Commands.

- **What Services?**
  Clock,
  Random Seed.

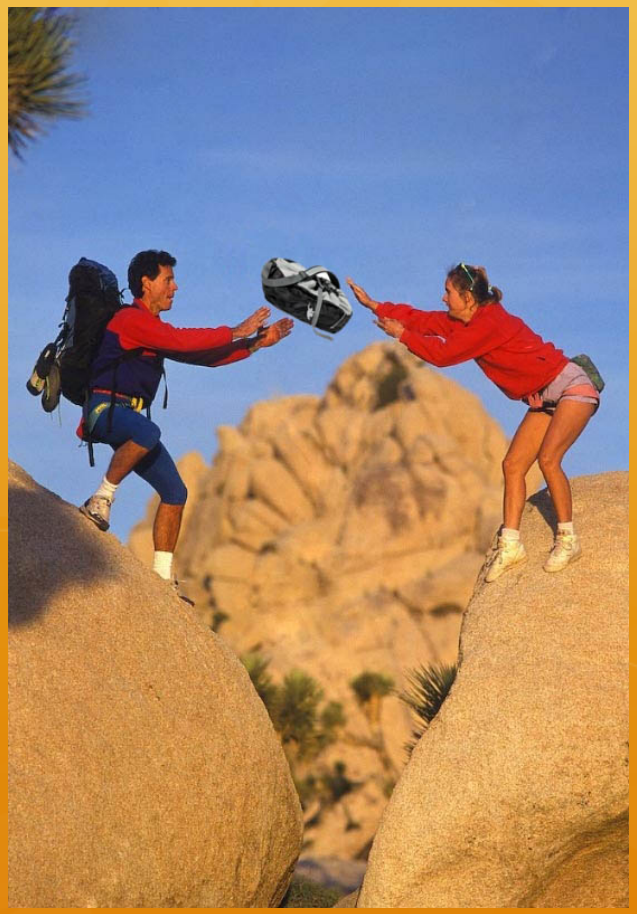
- **And…**
  Pass it around!

Bring it all **Together**

- **FleetingDataAndServices:**
  I like to create a single class that holds all of these critical Services together.
- **What Data?**
  Commands.
- **What Services?**
  Clock,
  Random Seed.
- **And…**
  Pass it around!

Bring it all **Together**

- **FleetingDataAndServices:**
  I like to create a single class that holds all of these critical Services together.
- **What Data?**
  Commands.
- **What Services?**
  Clock,
  Random Seed.
- **And…**
  Pass it around!



Fleeting**Data**And**Services**

```
private FleetingDataAndServices(Clock clock, long randomSeed, List<Command> commands, Audi
    this.randomSeed = randomSeed;
    random = new Random(randomSeed);
    this.clock = Clock.fixed(clock.instant(), clock.getZone());
    this.commands = commands;
    this.auditor = auditor;
    this.tempo = tempo;
}
```

Pass it around!

Fleeting**Data**And**Services**

Pass it around!

Bring it all **Together**

- **log it** wisely

Bring it all **Together**

- **log it** wisely
- and you've got yourself…

Bring it all **Together**

- **log it** wisely
- and you've got yourself…

A nice,

Bring it all **Together**

- **log it** wisely
- and you've got yourself…

A nice, testable

Bring it all **Together**

- **log it** wisely
- and you've got yourself…

A nice, testable system.

Deterministic testing
in a less deterministic world

Blog: **arturofalck.wordpress.com**
Code: github.com/afalck/offbeet-utils