Java Monitor Object
Motivating Example

Douglas C. Schmidt
da.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

- Understand what monitors are & know how Java built-in monitor objects can ensure mutual exclusion & coordination between threads
- Note a human-known use of monitors
- Recognize common synchronization problems in concurrent Java programs using an app case study

```
Concurrent calls to offer() & poll() corrupt internal state in the BuggyQueue fields
```
A Buggy Producer/Consumer App
A Buggy Producer/Consumer App

• A concurrent producer/consumer app that attempts to pass messages via an “BuggyQueue” class

See [github.com/douglascraigschmidt/POSA/tree/master/ex/M3/Queues/BuggyQueue](https://github.com/douglascraigschmidt/POSA/tree/master/ex/M3/Queues/BuggyQueue)
A Buggy Producer/Consumer App

- The BuggyQueue class is modeled on the Java ArrayBlockingQueue class

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ArrayBlockingQueue.html
A Buggy Producer/Consumer App

- UML class diagram showing the design of the BuggyQueue

See github.com/douglascraigschmidt/POSA/tree/master/ex/M3/Queues/BuggyQueue/app/src/main/java/edu/vandy/buggyqueue/model
A Buggy Producer/Consumer App

• UML sequence diagram of the BuggyQueue producer/consumer unit test

A Buggy Producer/Consumer App

- UML sequence diagram of the BuggyQueue producer/consumer unit test
A Buggy Producer/Consumer App

- UML sequence diagram of the BuggyQueue producer/consumer unit test

```
main()
```

```
start()
run()
```

```
new()
```

```
consumer : Thread
```

```
producer : Thread
```

```
buggyQueue : BuggyQueue
```

```
: BuggyQueueTest
```

```
BuggyQueueTest
```

```
```

```
A Buggy Producer/Consumer App : BuggyQueueTest
```

```
consumer : Thread
```

```
producer : Thread
```

```
buggyQueue : BuggyQueue
```

```
new()
```

```
new()
```

```
new()
```

```
start()
run()
```

```
run()
```

```
start()
```

```
```
A Buggy Producer/Consumer App

- UML sequence diagram of the BuggyQueue producer/consumer unit test

```
main() -> new() -> new()
          |    |    |
          v    v    v
start() -> run() -> run()
            |    |    |
            v    v    v
new() -> buggyQueue : BuggyQueue
        |    |    |
        v    v    v
consumer : Thread
        |    |    |
        v    v    v
producer : Thread
          |    |    |
          v    v    v
new() -> poll() -> offer("...")
```
A Buggy Producer/Consumer App

- UML sequence diagram of the BuggyQueue producer/consumer unit test

Since the offer() & poll() methods aren’t synchronized chaos & insanity will result when this app & unit test is run!!
The BuggyQueue Implementation
The BuggyQueue Implementation

- The BuggyQueue class is a simply wrapper around Java’s LinkedList class

```java
static class BuggyQueue<E> implements SimpleBlockingQueue<E> {
    private List<E> mList = new LinkedList<>(); ...

    public void offer(E e) {
        if (!isFull())
            mList.add(e); return true;
        else
            return false;
    }

    public E poll() {
        if (!isEmpty())
            return mList.remove(0);
        else
            return null;
    }

    ...
```

The BuggyQueue Implementation

- The BuggyQueue class is a simply wrapper around Java’s LinkedList class

```java
static class BuggyQueue<E> implements SimpleBlockingQueue<E> {
    private List<E> mList = new LinkedList<>(); ...

    public void offer(E e) {
        if (!isFull())
            mList.add(e); return true;
        else
            return false;
    }

    public E poll() {
        if (!isEmpty())
            return mList.remove(0);
        else
            return null;
    }

    ...}
```

This interface is a variant of what’s available in Java’s BlockingQueue interface
The BuggyQueue Implementation

- The BuggyQueue class is a simply wrapper around Java’s LinkedList class

```java
static class BuggyQueue<E> implements SimpleBlockingQueue<E> {
    private List<E> mList = new LinkedList<>(); ...

    public void offer(E e) {
        if (!isFull()) {
            mList.add(e); return true;
        } else {
            return false;
        }
    }

    public E poll() {
        if (!isEmpty()) {
            return mList.remove(0);
        } else {
            return null;
        }
    }

    ...
}

See docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html
```
The BuggyQueue Implementation

- The BuggyQueue class is a simply wrapper around Java’s LinkedList class

```java
static class BuggyQueue<E> implements SimpleBlockingQueue<E> {
    private List<E> mList = new LinkedList<>(); ...

    public void offer(E e) {
        if (!isFull())
            { mList.add(e); return true; } 
        else
            return false;
    }

    public E poll() {
        if (!isEmpty())
            return mList.remove(0);
        else
            return null;
    }

    ...
```
The BuggyQueue Implementation

- The BuggyQueue class is a simply wrapper around Java’s LinkedList class

```java
static class BuggyQueue<E> implements SimpleBlockingQueue<E> {
    private List<E> mList = new LinkedList<>(); ...

    public void offer(E e) {
        if (!isFull())
            { mList.add(e); return true; }
        else
            return false;
    }

    public E poll() {
        if (!isEmpty())
            return mList.remove(0);
        else
            return null;
    }

    ...
```

See en.wikipedia.org/wiki/Robot_B-9
End of Java Monitor Object
Motivating Example