Learning Objectives in this Part of the Lesson

- Recognize the synchronized methods/statements provided by Java build-in monitor objects to support *mutual exclusion*
- Understand how to fix the race conditions in the buggy concurrent Java app by using synchronized methods

The use of synchronized methods only provides a partial solution, however…
Partial Solution Using Java
Synchronized Methods
Partial Solution Using Java Synchronized Methods

See en.wikipedia.org/wiki/Crazy_Horse_Memorial
Partial Solution Using Java Synchronized Methods

- A concurrent producer/consumer app that passes messages via the class “BusySynchronizedQueue”

See [github.com/douglascraigschmidt/POSA/tree/master/ex/M3/Queues/BusySynchronizedQueue](http://github.com/douglascraigschmidt/POSA/tree/master/ex/M3/Queues/BusySynchronizedQueue)
Partial Solution Using Java Synchronized Methods

- The BusySynchronizedQueue is modeled on the Java ArrayBoundedQueue

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ArrayBoundedQueue.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/ArrayBoundedQueue.html)
Partial Solution Using Java Synchronized Methods

- UML class diagram showing the design of the BusySynchronizedQueue

Partial Solution Using Java Synchronized Methods

- UML sequence diagram of the BusySynchronizedQueue unit test

See github.com/douglascraigschmidt/POSA/tree/master/ex/M3/Queues/BusySynchronizedQueue/app/src/test/java/edu/vandy/busysynchronizedqueue
Partial Solution Using Java Synchronized Methods

- UML sequence diagram of the BusySynchronizedQueue unit test

The main thread coordinates the other threads in the test.
Partial Solution Using Java Synchronized Methods

- UML sequence diagram of the BusySynchronizedQueue unit test

<table>
<thead>
<tr>
<th>: BusySynchronizedQueueTest -&gt;</th>
<th>consumer : Thread -&gt;</th>
<th>producer : Thread -&gt;</th>
<th>busyQueue : BusySynchronizedQueue</th>
</tr>
</thead>
</table>

The consumer & producer threads generate & process messages sent via the BusySynchronizedQueue, respectively.
Partial Solution Using Java Synchronized Methods

- UML sequence diagram of the BusySynchronizedQueue unit test

The offer() & poll() methods are synchronized, so the test runs correctly, but is inefficient due to “busy waiting”!!
Implementation of the BusySynchronizedQueue
Implementation of the BusySynchronizedQueue

• Java synchronized methods protects critical sections from concurrent access

```java
class BusySynchronizedQueue<E> implements BoundedQueue<E> {
    private ListedList<E> mList;
    private int mCapacity;

    public BusySynchronizedQueue(int capacity) {
        mCapacity = capacity; mList = new LinkedList<>();
    }

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

    public E synchronized poll() { return mList.poll(); } ...
```

See [github.com/douglascraigschmidt/POSA/tree/master/ex/M3/Queues/BusySynchronizedQueue](https://github.com/douglascraigschmidt/POSA/tree/master/ex/M3/Queues/BusySynchronizedQueue)
Java synchronized methods protects critical sections from concurrent access

```java
class BusySynchronizedQueue<E>
    implements BoundedQueue<E> {
    private ListedList<E> mList;
    private int mCapacity;

    public BusySynchronizedQueue(int capacity) {
        mCapacity = capacity; mList = new LinkedList<>();
    }

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

    public synchronized E poll() { return mList.poll(); } ...
```

Only one synchronized method can be active in any given object
Java synchronized methods protect critical sections from concurrent access.

```java
class BusySynchronizedQueue<E> implements BoundedQueue<E> {
    private ListedList<E> mList;
    private int mCapacity;

    public BusySynchronizedQueue(int capacity) {
        mCapacity = capacity; mList = new LinkedList<>();
    }

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

    public E synchronized poll() { return mList.poll(); }
    ...
}
```

This constraint may actually be a liability for certain types of concurrently accessed objects, e.g., double-ended queues implemented as linked lists.
Java synchronized methods protects critical sections from concurrent access

• Adding the synchronized keyword has two effects

```java
class BusySynchronizedQueue<E> implements BoundedQueue<E> {
    private ListedList<E> mList;
    private int mCapacity;

    public BusySynchronizedQueue(int capacity) {
        mCapacity = capacity; mList = new LinkedList<>();
    }

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

    public synchronized E poll() { return mList.poll(); }

    ...
}
```

See docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html
Java synchronized methods protects critical sections from concurrent access

- Adding the synchronized keyword has two effects

```java
class BusySynchronizedQueue<E> implements BoundedQueue<E> {
    private LinkedList<E> mList;
    private int mCapacity;

    public BusySynchronizedQueue(int capacity) {
        mCapacity = capacity; mList = new LinkedList<>();
    }

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

    public E synchronized poll() { return mList.poll(); }
    ...
```

Invocations of offer() & poll() on the same object can’t interleave

i.e., each synchronized method is “atomic?”
Java synchronized methods protects critical sections from concurrent access
  • Adding the synchronized keyword has two effects

```java
class BusySynchronizedQueue<E> implements BoundedQueue<E> {
    private ListedList<E> mList;
    private int mCapacity;

    public BusySynchronizedQueue(int capacity) {
        mCapacity = capacity; mList = new LinkedList<>();
    }

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

    public E synchronized poll() { return mList.poll(); }
    ...
```

Establishes a “happens-before” relation to ensure visibility of state changes to all threads.

See en.wikipedia.org/wiki/Happened-before
Evaluating the Busy SynchronizedQueue
Evaluating the BusySynchronizedQueue

- There are limitations with Java synchronized methods when they are used alone

See [github.com/douglascraigschmidt/POSA/tree/master/ex/M3/Queues/BusySynchronizedQueue](https://github.com/douglascraigschmidt/POSA/tree/master/ex/M3/Queues/BusySynchronizedQueue)
There are limitations with Java synchronized methods when they are used alone.

```java
class BusySynchronizedQueue<E> implements BoundedQueue<E> {
    private ListedList<E> mList;
    private int mCapacity;

    public BusySynchronizedQueue(int capacity) {
        mCapacity = capacity; mList = new LinkedList<>();
    }

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

    public E synchronized poll() { return mList.poll(); }
    ...
```

Concurrent calls to these methods will "busy wait".

See en.wikipedia.org/wiki/Busy_waiting
There are limitations with Java synchronized methods when they are used alone.

```java
class BusySynchronizedQueue<E> implements BoundedQueue<E> {
    private ListedList<E> mList;
    private int mCapacity;

    public BusySynchronizedQueue(int capacity) {
        mCapacity = capacity; mList = new LinkedList<>;
    }

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

    public E synchronized poll() { return mList.poll(); }
    ...
}
```

Java built-in monitor objects therefore provide “wait” & “notify” mechanisms to avoid busy waiting.
End of Java Monitor
Objects: Synchronization
(Part 3)
1. Which of the following are limitations the BusySynchronizedQueue implementation of the app?
   
a. Only one synchronized method can be active in any given object
   
b. A “happens-before” relation must be provided to ensure visibility of state changes to all threads
   
c. Concurrent calls to its methods may “busy wait” if a resource is not available
   
d. Invocations of offer() & poll() on the same object can interleave