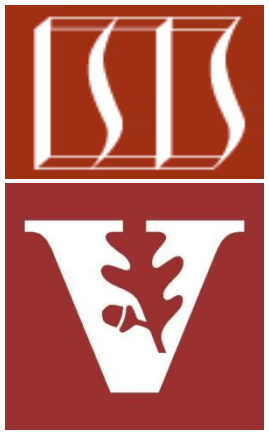
Examples of Java "Happens-Before" Relationships

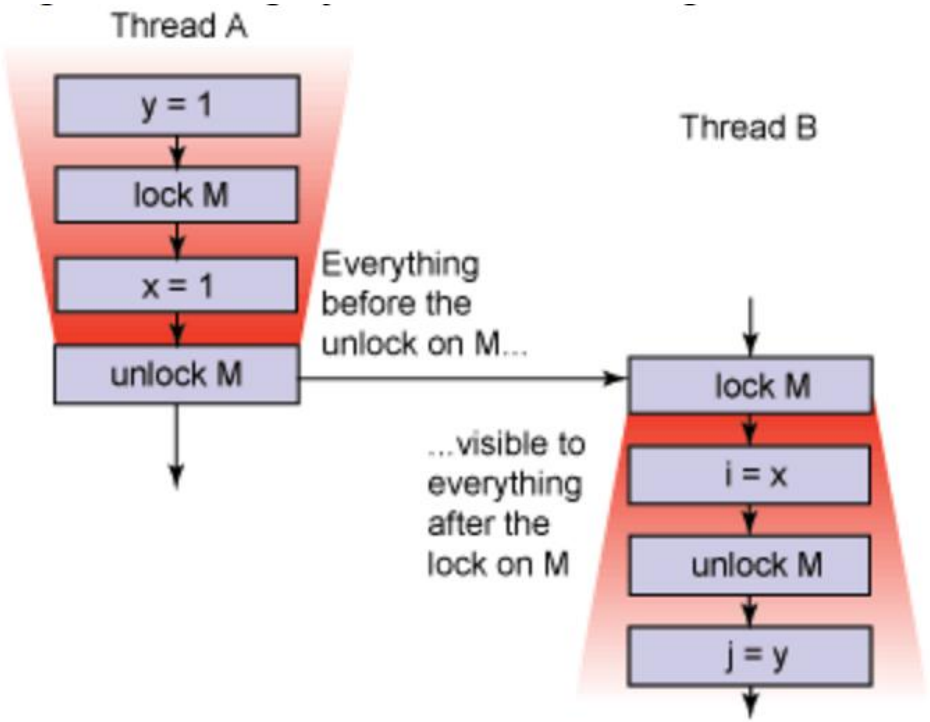


Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand what "happens-before" relationships mean in Java
- Recognize how Java Thread methods support "happens-before" relationships

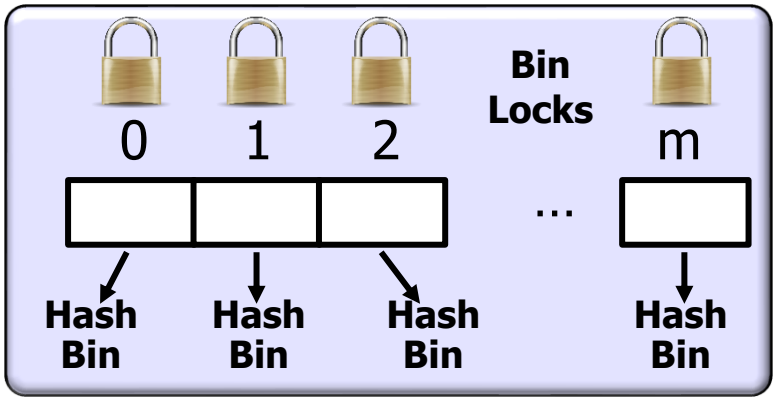


<<Java Class>>
Thread

- yield():void
- currentThread():Thread
- sleep(long):void
- sleep(long,int):void
- Thread()
- Thread(Runnable)
- Thread(String)
- start():void
- run():void
- exit():void
- interrupt():void
- interrupted():boolean
- isInterrupted():boolean
- isAlive():boolean
- setPriority(int):void
- getPriority():int
- join(long):void
- join(long,int):void
- join():void
- setDaemon(boolean):void
- isDaemon():boolean

Learning Objectives in this Part of the Lesson

- Understand what "happens-before" relationships mean in Java
- Recognize how Java Thread methods support "happens-before" relationships
- Know how Java collections support "happens-before" relationships

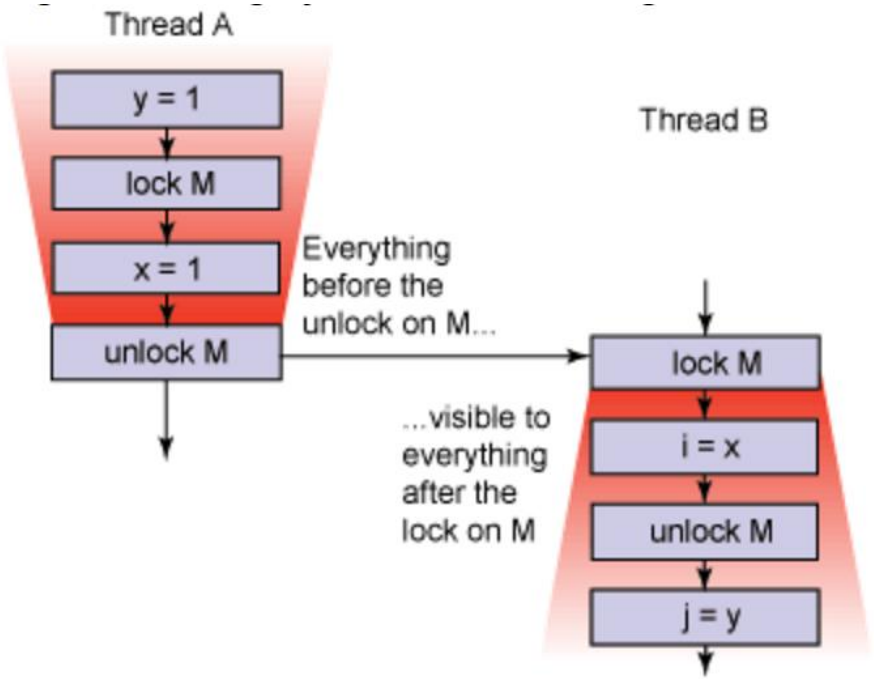


ConcurrentHashMap

Java Thread "Happens-Before" Relationships

Java Thread "Happens-Before" Relationships

- Methods in the Java Thread class establish "happen-before" relationships



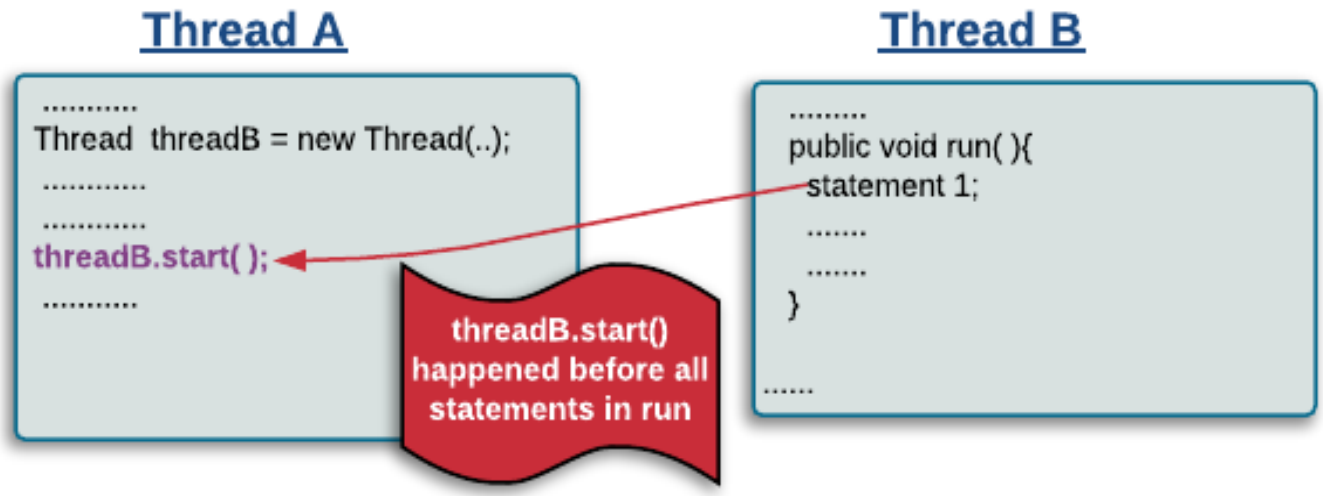
<<Java Class>>
Thread

- yield():void
- currentThread():Thread
- sleep(long):void
- sleep(long,int):void
- Thread()
- Thread(Runnable)
- Thread(String)
- start():void
- run():void
- exit():void
- interrupt():void
- interrupted():boolean
- isInterrupted():boolean
- isAlive():boolean
- setPriority(int):void
- getPriority():int
- join(long):void
- join(long,int):void
- join():void
- setDaemon(boolean):void
- isDaemon():boolean

See docs.oracle.com/javase/8/docs/api/java/lang/Thread.html

Java Thread "Happens-Before" Relationships

- Methods in the Java Thread class establish "happen-before" relationships
  - Starting a thread "happens-before" the run() hook method of the thread is called



<<Java Class>>
Thread

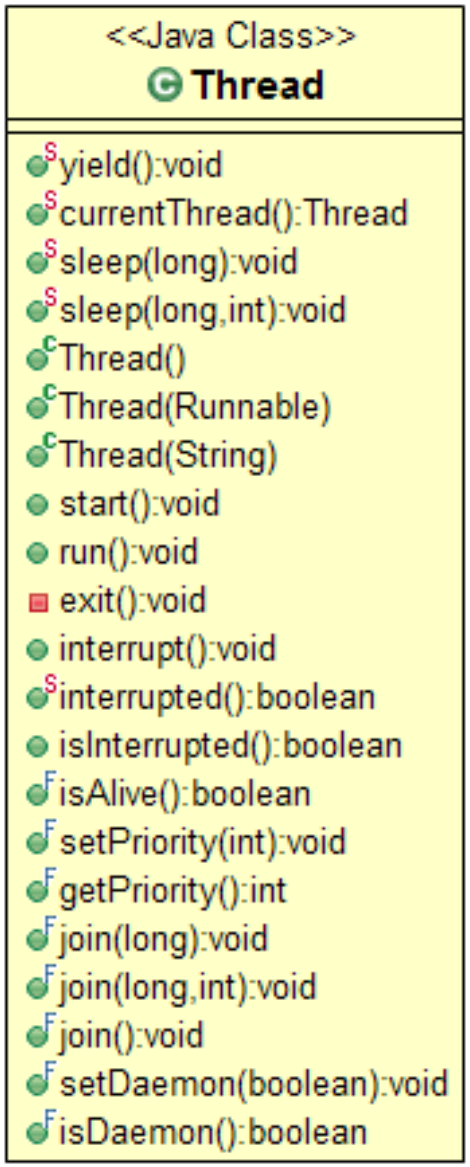
- yield():void
- currentThread():Thread
- sleep(long):void
- sleep(long,int):void
- Thread()
- Thread(Runnable)
- Thread(String)
- start():void
- run():void
- exit():void
- interrupt():void
- interrupted():boolean
- isInterrupted():boolean
- isAlive():boolean
- setPriority(int):void
- getPriority():int
- join(long):void
- join(long,int):void
- join():void
- setDaemon(boolean):void
- isDaemon():boolean

See www.logicbig.com/tutorials/core-java-tutorial/java-multi-threading/happens-before.html

Java Thread "Happens-Before" Relationships

- Methods in the Java Thread class establish "happen-before" relationships
  - Starting a thread "happens-before" the run() hook method of the thread is called

```
Thread threadB =
  new Thread(() ->
               System.out.println
                 ("hello world"));

threadB.start();

...
```

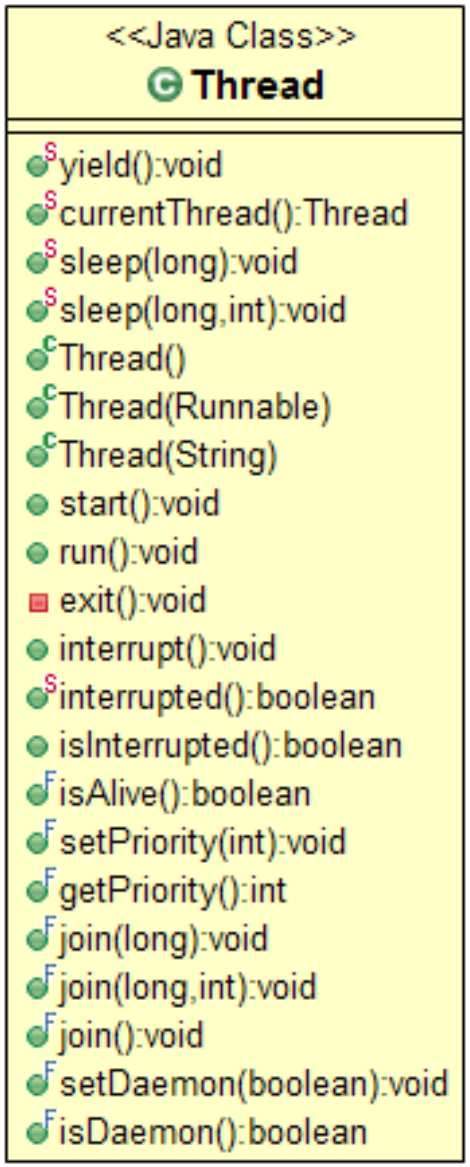


Java Thread "Happens-Before" Relationships

- Methods in the Java Thread class establish "happen-before" relationships
  - Starting a thread "happens-before" the run() hook method of the thread is called

```
Thread threadB =
  new Thread(() ->
               System.out.println
                 ("hello world"));

threadB.start();

...
```

Create & start threadB from within threadA

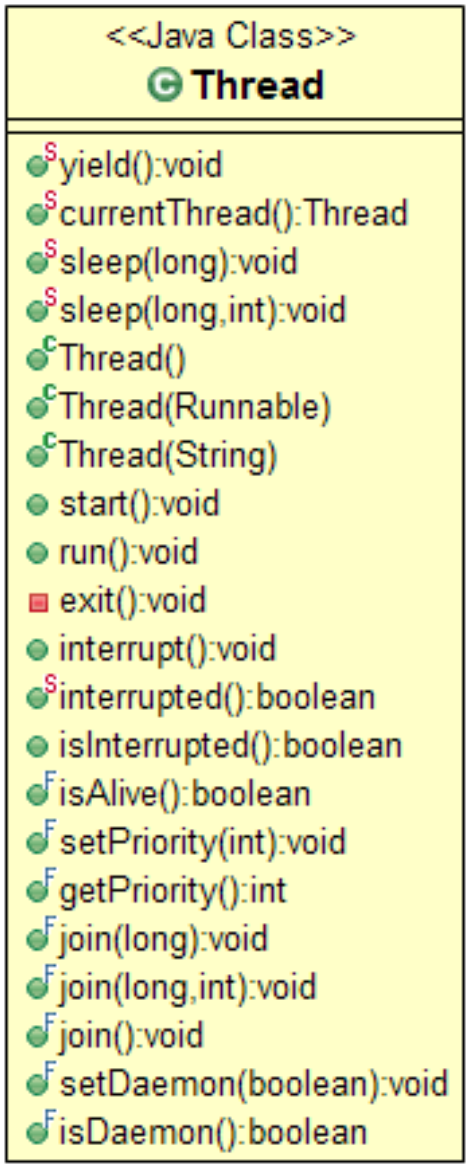


Java Thread "Happens-Before" Relationships

- Methods in the Java Thread class establish "happen-before" relationships
  - Starting a thread "happens-before" the run() hook method of the thread is called

```
Thread threadB =
  new Thread(() ->
               System.out.println
                 ("hello world"));

threadB.start();

...
```

This lambda expression plays the role of the run() hook method!

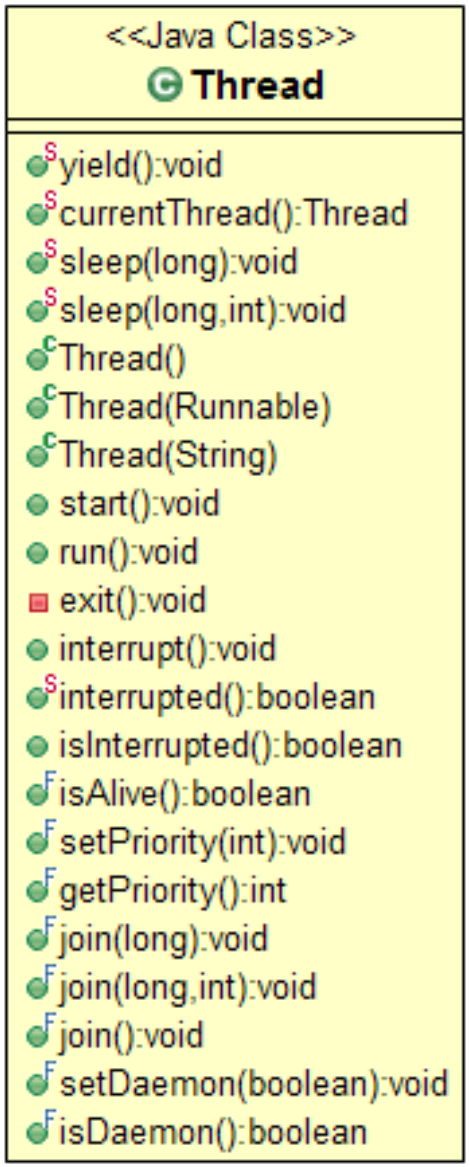


Java Thread "Happens-Before" Relationships

- Methods in the Java Thread class establish "happen-before" relationships
  - Starting a thread "happens-before" the run() hook method of the thread is called

```
Thread threadB =
  new Thread(() ->
               System.out.println
                 ("hello world"));

threadB.start();

...
```

threadA's call to the threadB.start() method (& associated changes it made to any shared state) will "happen before" threadB's run() hook method is called

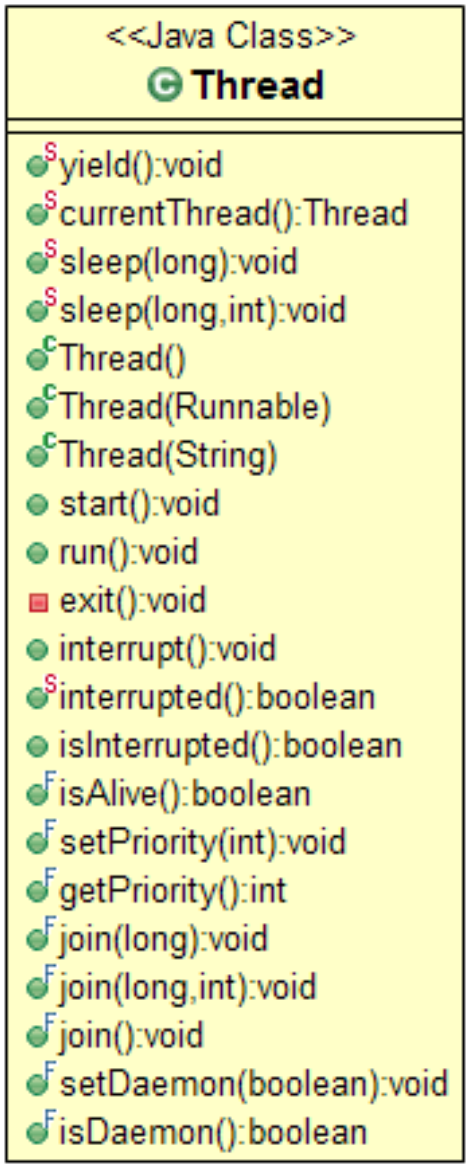


Java Thread "Happens-Before" Relationships

- Methods in the Java Thread class establish "happen-before" relationships
  - Starting a thread "happens-before" the run() hook method of the thread is called

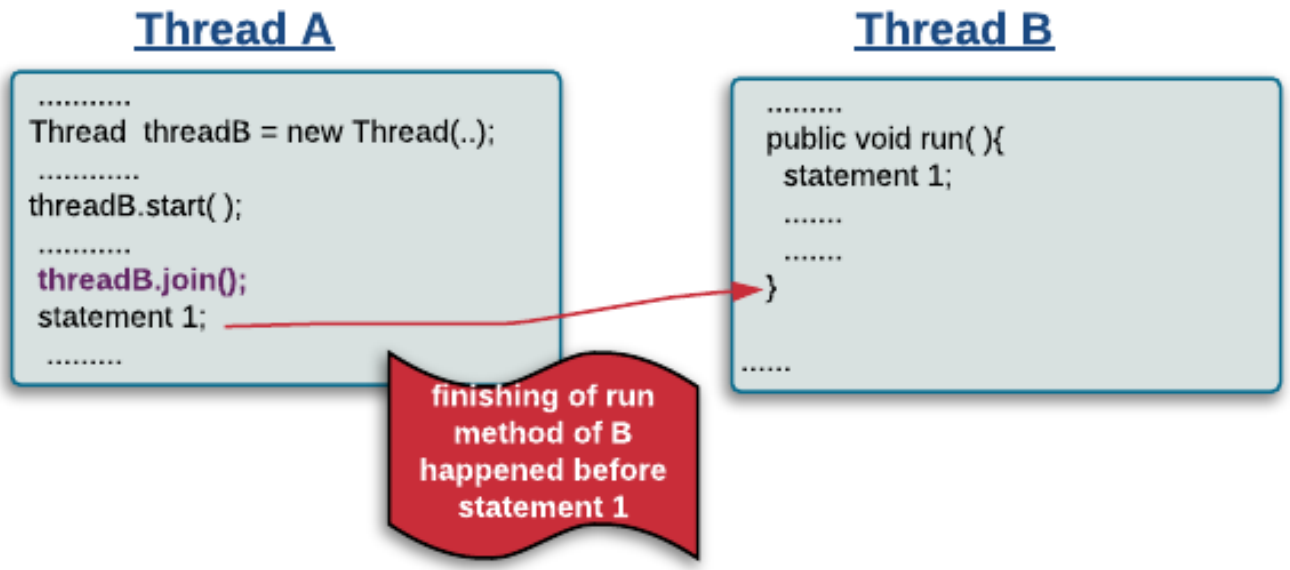
```
Thread threadB =
  new Thread(() ->
               System.out.println
                 ("hello world"));

threadB.start();

...
```

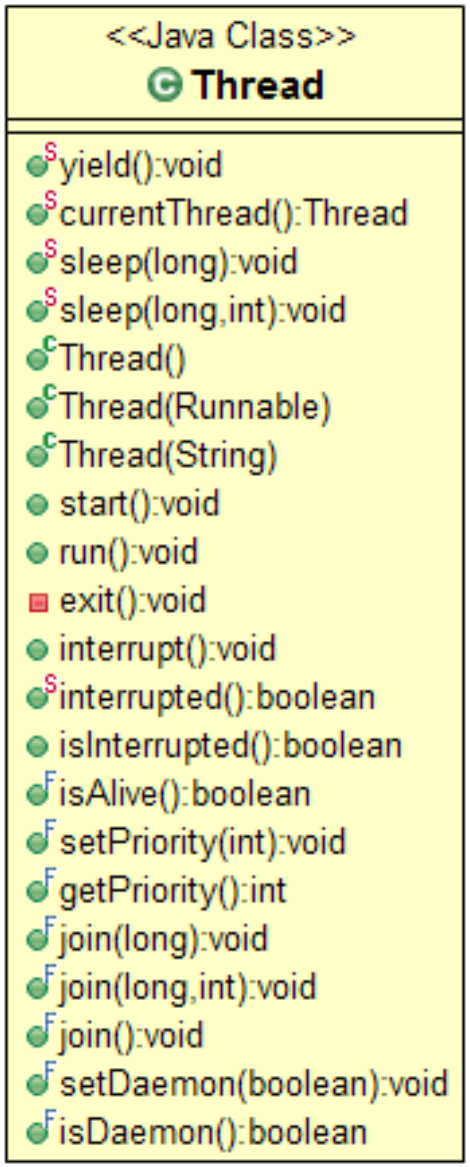
Likewise, the state of threadB will be consistent & visible before the run() hook method begins to execute



Java Thread "Happens-Before" Relationships

- Methods in the Java Thread class establish "happen-before" relationships
  - Starting a thread "happens-before" the run() hook method of the thread is called
  - The termination of a thread "happens-before" a join() with the terminated thread

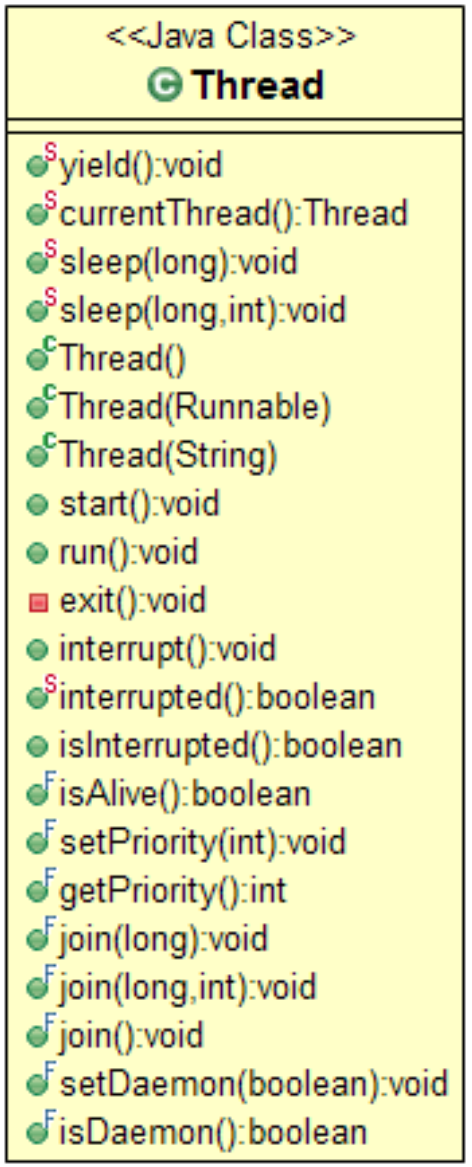




Java Thread "Happens-Before" Relationships

- Methods in the Java Thread class establish "happen-before" relationships
  - Starting a thread "happens-before" the run() hook method of the thread is called
  - The termination of a thread "happens-before" a join() with the terminated thread

```
Thread threadB =
  new Thread(() ->
               System.out.println
                 ("hello world"));
threadB.start();

...

threadB.join();
```

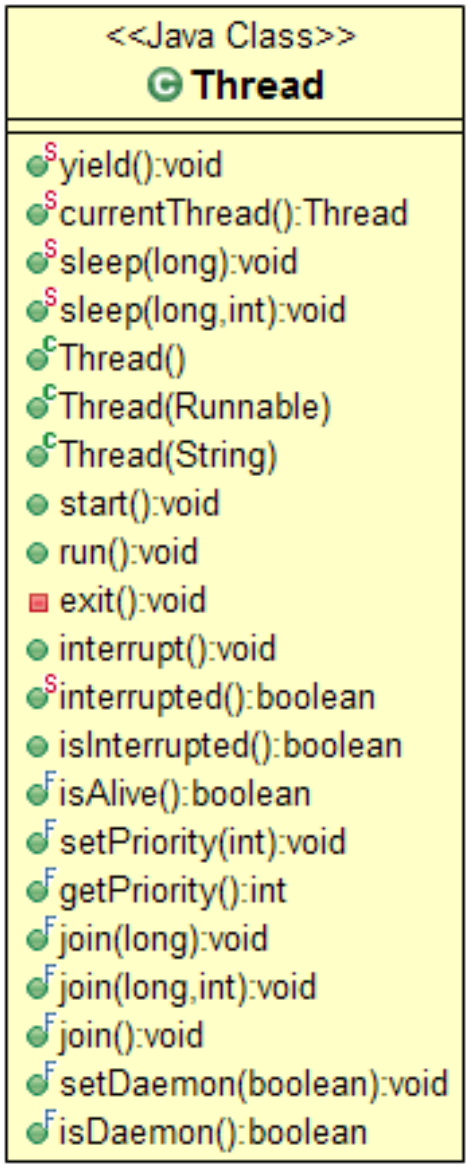


Java Thread "Happens-Before" Relationships

- Methods in the Java Thread class establish "happen-before" relationships
  - Starting a thread "happens-before" the run() hook method of the thread is called
  - The termination of a thread "happens-before" a join() with the terminated thread

```
Thread threadB =
  new Thread(() ->
               System.out.println
                 ("hello world"));
threadB.start();

...

threadB.join();
```

threadB terminates after its lambda expression run() processing completes



Java Thread "Happens-Before" Relationships

- Methods in the Java Thread class establish "happen-before" relationships
  - Starting a thread "happens-before" the run() hook method of the thread is called
  - The termination of a thread "happens-before" a join() with the terminated thread

```
Thread threadB =
  new Thread(() ->
               System.out.println
                 ("hello world"));
threadB.start();

...

threadB.join();
```

threadA waiting on join() only resumes its processing after threadB terminates

<<Java Class>>
Thread

- yield():void
- currentThread():Thread
- sleep(long):void
- sleep(long,int):void
- Thread()
- Thread(Runnable)
- Thread(String)
- start():void
- run():void
- exit():void
- interrupt():void
- interrupted():boolean
- isInterrupted():boolean
- isAlive():boolean
- setPriority(int):void
- getPriority():int
- join(long):void
- join(long,int):void
- join():void
- setDaemon(boolean):void
- isDaemon():boolean

Java Thread "Happens-Before" Relationships

- Methods in the Java Thread class establish "happen-before" relationships
  - Starting a thread "happens-before" the run() hook method of the thread is called
  - The termination of a thread "happens-before" a join() with the terminated thread

```
Thread threadB =
  new Thread(() ->
               System.out.println
                 ("hello world"));
threadB.start();

...

threadB.join();
```

After join() returns threadA must see all changes made to shared state by threadB that "happened before" it exited

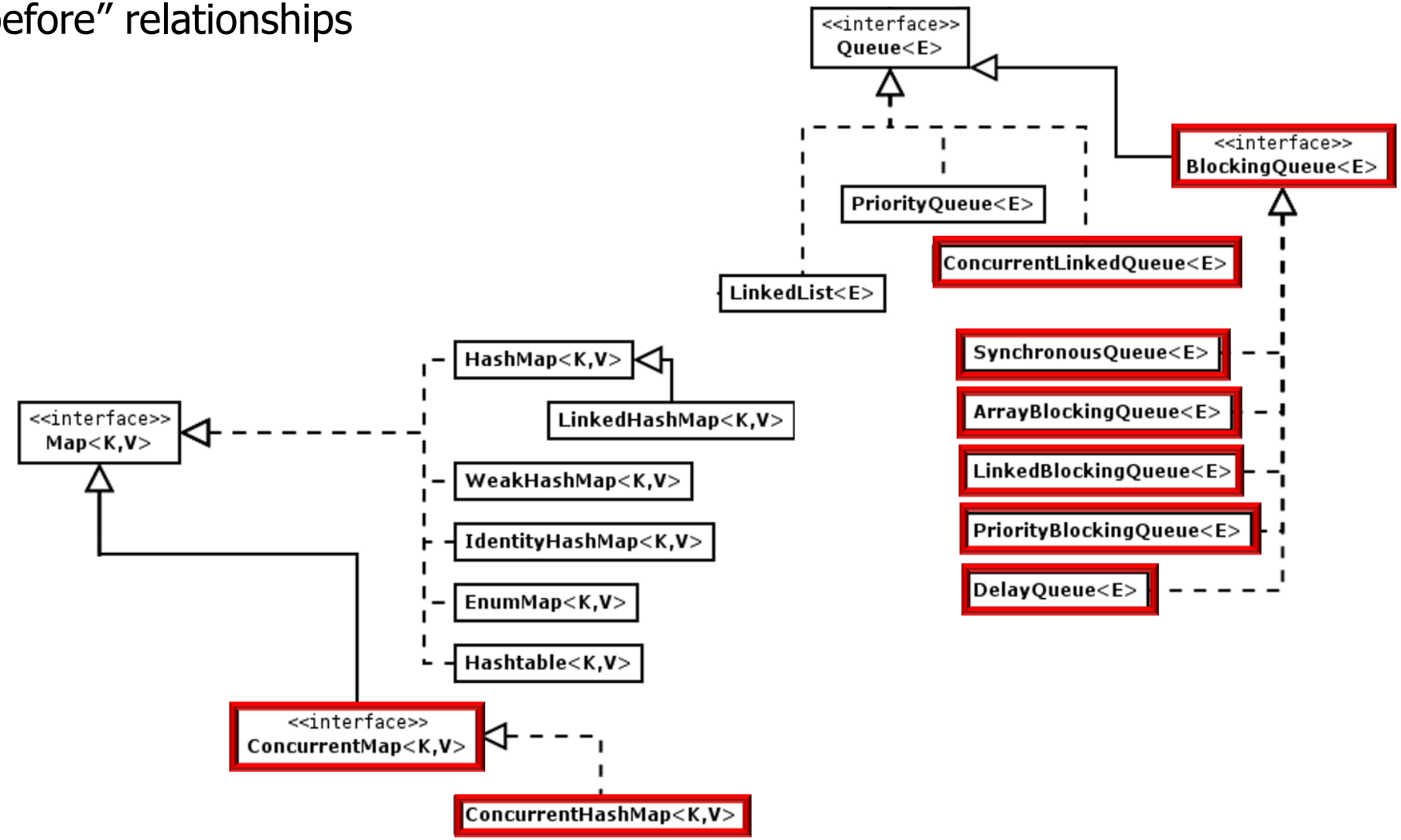
<<Java Class>>
Thread

- yield():void
- currentThread():Thread
- sleep(long):void
- sleep(long,int):void
- Thread()
- Thread(Runnable)
- Thread(String)
- start():void
- run():void
- exit():void
- interrupt():void
- interrupted():boolean
- isInterrupted():boolean
- isAlive():boolean
- setPriority(int):void
- getPriority():int
- join(long):void
- join(long,int):void
- join():void
- setDaemon(boolean):void
- isDaemon():boolean

Java Collections "Happens-Before" Relationships

Java Collections "Happens-Before" Relationships

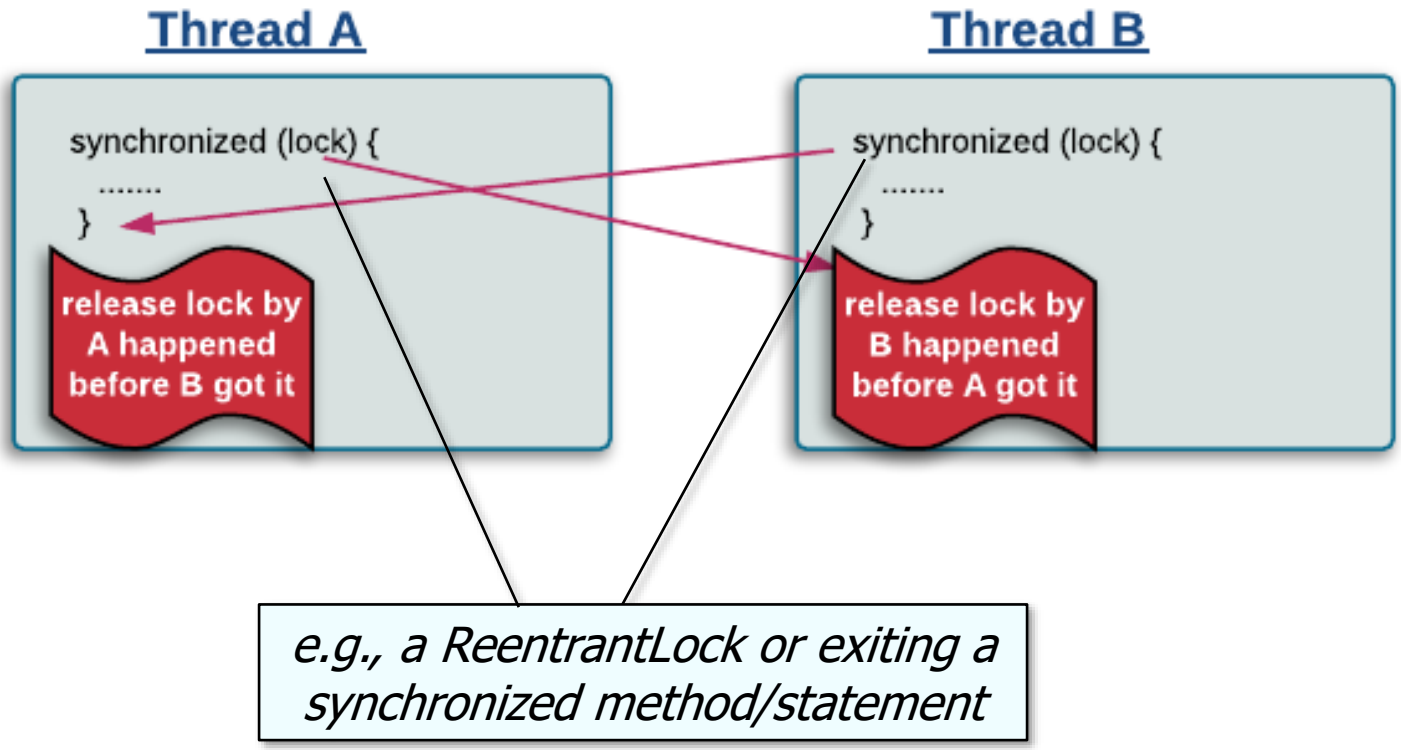
- Methods in java.util.concurrent package classes also establish "happen-before" relationships



See docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-summary.html#MemoryVisibility

Java Collections "Happens-Before" Relationships

- Methods in java.util.concurrent package classes also establish "happen-before" relationships
  - The release of a monitor lock "happens-before" every subsequent acquire on the same lock



See www.logicbig.com/tutorials/core-java-tutorial/java-multi-threading/happens-before.html

Java Collections "Happens-Before" Relationships

- Methods in java.util.concurrent package classes also establish "happen-before" relationships
  - The release of a monitor lock "happens-before" every subsequent acquire on the same lock

```
// Thread A
class ArrayBlockingQueue<E>
  ... { ...
  public void put(E e) ... {
    ...
    final ReentrantLock lock =
      this.lock;
    lock.lockInterruptibly();
    try { ...
    } finally {
        lock.unlock();
    }
  }
```

```
// Thread B
class ArrayBlockingQueue<E>
  ... { ...
  public E take() ... {
    final ReentrantLock lock
      = this.lock;
    lock.lockInterruptibly();
    try { ...
    } finally {
      lock.unlock();
    }
```

Java Collections "Happens-Before" Relationships

- Methods in java.util.concurrent package classes also establish "happen-before" relationships
  - The release of a monitor lock "happens-before" every subsequent acquire on the same lock

```
// Thread A
class ArrayBlockingQueue<E>
  ... { ...
  public void put(E e) ... {
    ...
    final ReentrantLock lock =
      this.lock;
    lock.lockInterruptibly();
    try { ...
    } finally {
        lock.unlock();
    }
  }
```

```
// Thread B
class ArrayBlockingQueue<E>
  ... { ...
  public E take() ... {
    final ReentrantLock lock
      = this.lock;
    lock.lockInterruptibly();
    try { ...
    } finally {
      lock.unlock();
    }
```

Consider the put() & take() methods in ArrayBlockingQueue

See earlier lessons on "*Java ReentrantLock*" & "*Java ConditionObject*"

Java Collections "Happens-Before" Relationships

- Methods in java.util.concurrent package classes also establish "happen-before" relationships
  - The release of a monitor lock "happens-before" every subsequent acquire on the same lock

```
// Thread A
class ArrayBlockingQueue<E>
  ... { ...
  public void put(E e) ... {
    ...
    final ReentrantLock lock =
      this.lock;
    lock.lockInterruptibly();
    try { ...
    } finally {
        lock.unlock();
    }
  }
```

```
// Thread B
class ArrayBlockingQueue<E>
  ... { ...
  public E take() ... {
    final ReentrantLock lock
      = this.lock;
    lock.lockInterruptibly();
    try { ...
    } finally {
      lock.unlock();
    }
```

Actions prior to "releasing" the ReentrantLock must happen-before actions subsequent to a successful "acquiring" of this lock

See earlier lessons on "*Java ReentrantLock*" & "*Java ConditionObject*"

Java Collections "Happens-Before" Relationships

- Methods in java.util.concurrent package classes also establish "happen-before" relationships
  - The release of a monitor lock "happens-before" every subsequent acquire on the same lock
  - Actions in a thread prior to placing an object into any concurrent collection "happen-before" actions subsequent to the access or removal of that element from the collection in another thread

```
Map<String, String> concurrentMap = new ConcurrentHashMap<>();

// Thread t1
concurrentMap.put("key", "value");

// Thread t2
String value = concurrentMap.get("key");
```

See upcoming lesson on "*Java Concurrent Collections*"

Java Collections "Happens-Before" Relationships

- Methods in java.util.concurrent package classes also establish "happen-before" relationships
  - The release of a monitor lock "happens-before" every subsequent acquire on the same lock
  - Actions in a thread prior to placing an object into any concurrent collection "happen-before" actions subsequent to the access or removal of that element from the collection in another thread

```
Map<String, String> concurrentMap = new ConcurrentHashMap<>();

// Thread t1
concurrentMap.put("key", "value");

// Thread t2
String value = concurrentMap.get("key");
```

Consider a ConcurrentHashMap that supports concurrent retrievals & high expected concurrency for updates

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html

Java Collections "Happens-Before" Relationships

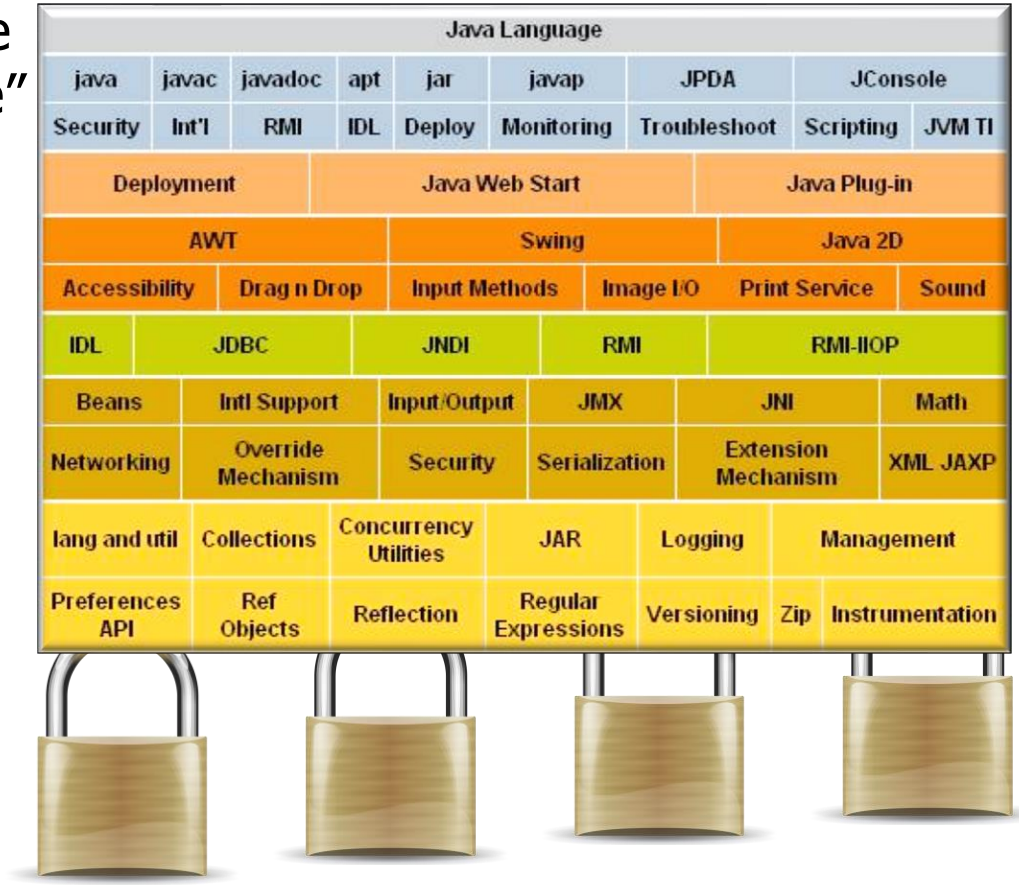
- Methods in java.util.concurrent package classes also establish "happen-before" relationships
  - The release of a monitor lock "happens-before" every subsequent acquire on the same lock
  - Actions in a thread prior to placing an object into any concurrent collection "happen-before" actions subsequent to the access or removal of that element from the collection in another thread

```
Map<String, String> concurrentMap = new ConcurrentHashMap<>();

// Thread t1
concurrentMap.put("key", "value");

// Thread t2
String value = concurrentMap.get("key");
```

Placing a "key/value" element into a ConcurrentHashMap must happen-before accessing or removing this element from the map

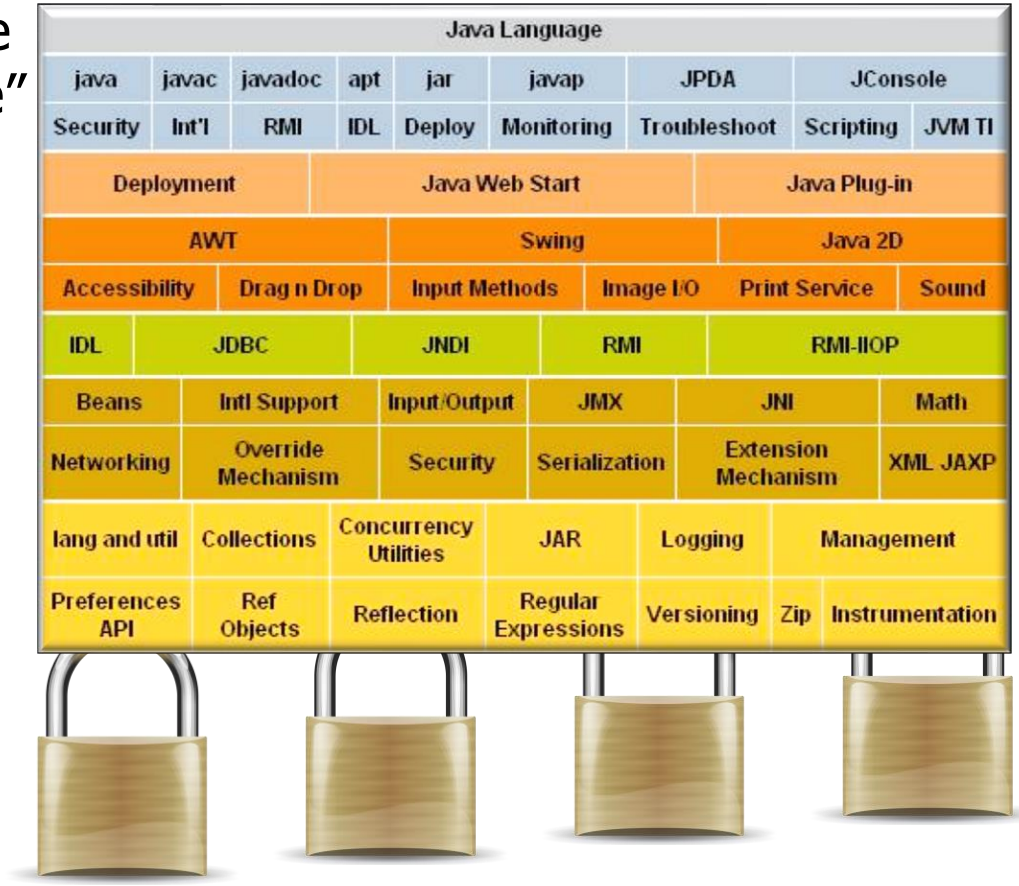
Java Collections "Happens-Before" Relationships

- Java's class libraries are responsible for ensuring these "happens-before" relationships are preserved



Java Collections "Happens-Before" Relationships

- Java's class libraries are responsible for ensuring these "happens-before" relationships are preserved



You don't need to understand all the nitty-gritty details of Java's memory model – you just need to understand how to use synchronizers properly!

End of Examples of "Happens-Before" Relationships