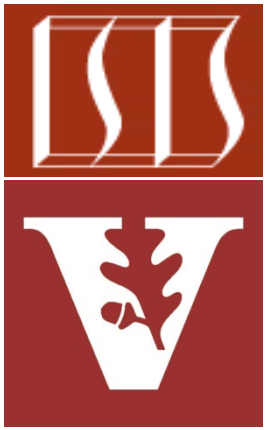
Android Concurrency: Java ReentrantReadWriteLock



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 Module

- Understand how ReentrantReadWriteLocks provide readers-writer lock semantics to concurrent Java programs

ReentrantReadWriteLock

Added in API level 1

extends Object
implements Serializable ReadWriteLock

java.lang.Object
↳java.util.concurrent.locks.ReentrantReadWriteLock

Class Overview

An implementation of `ReadWriteLock` supporting similar semantics to `ReentrantLock`.

This class has the following properties:

- **Acquisition order**

  This class does not impose a reader or writer preference ordering for lock access. However, it does support an optional *fairness* policy.

  Non-fair mode (default)

  When constructed as non-fair (the default), the order of entry to the read and write lock is unspecified, subject to reentrancy constraints. A nonfair lock that is continuously contended may indefinitely postpone one or more reader or writer threads, but will normally have higher throughput than a fair lock.

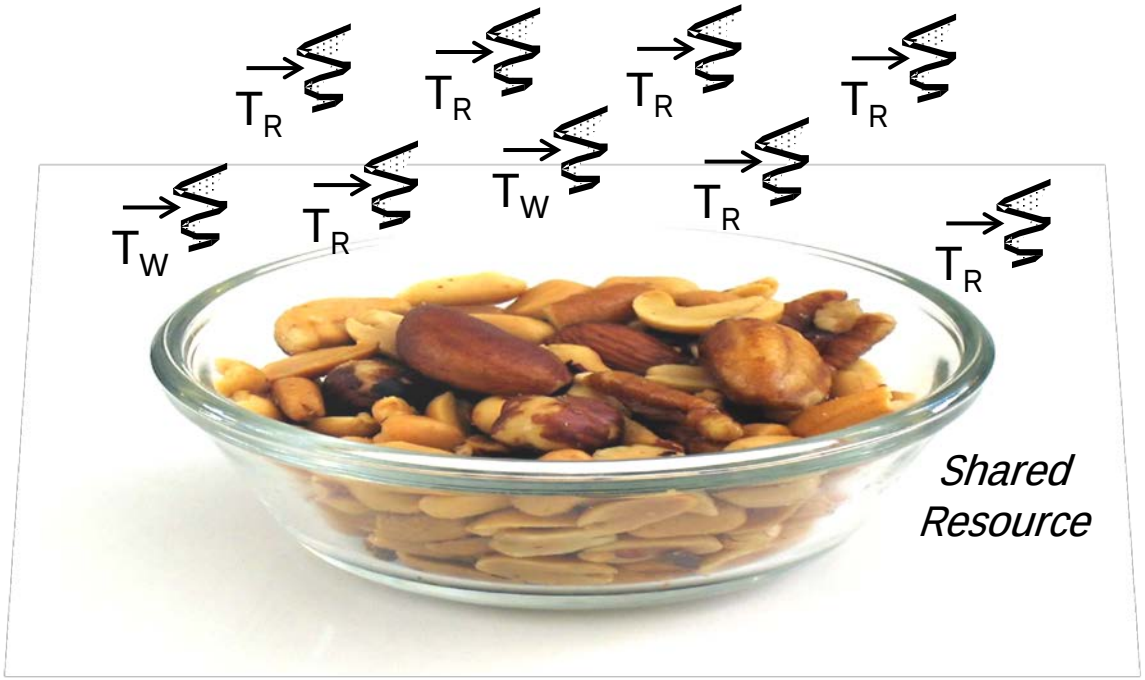
  Fair mode

  When constructed as fair, threads contend for entry using an approximately arrival-order policy. When the currently held lock is released, either the longest-waiting single writer thread will be assigned the write lock, or if there is a group of reader threads waiting longer than all waiting writer threads, that group will be assigned the read lock.

  A thread that tries to acquire a fair read lock (non-reentrantly) will block if either the write lock is held, or there is a waiting writer thread. The thread will not acquire the read lock until after the oldest currently waiting writer thread has acquired and released the write lock. Of course, if a waiting writer abandons its wait, leaving one or more reader threads as the longest waiters in the queue with the write lock free, then those readers will be assigned the read lock.

Overview of Readers-Writer Locks

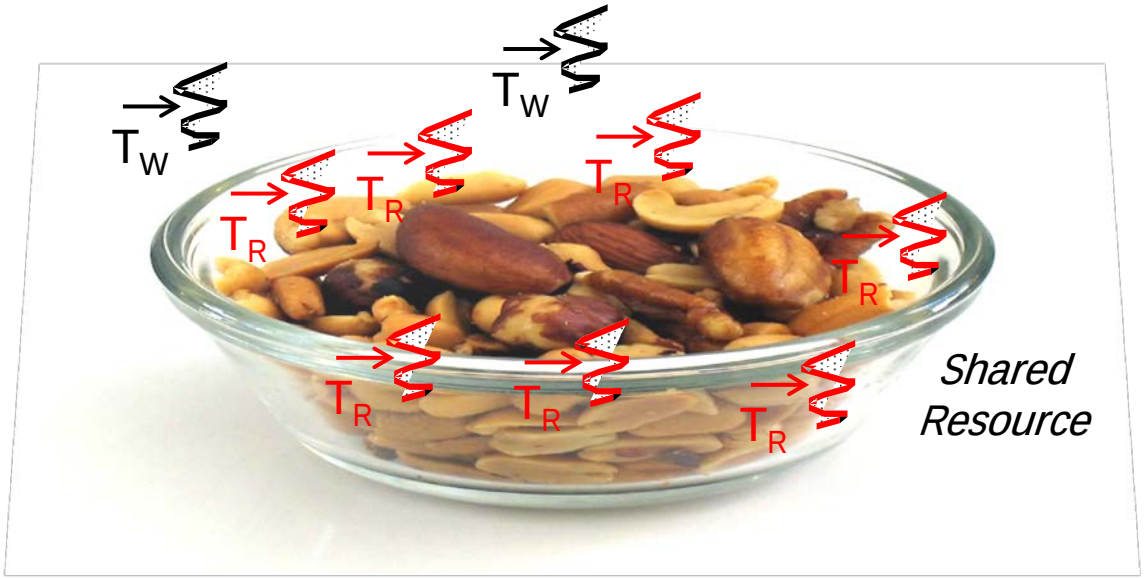
- A readers-writer lock is a synchronization mechanism often used in contexts with many concurrent threads



en.wikipedia.org/wiki/Readers-writer_lock has more info

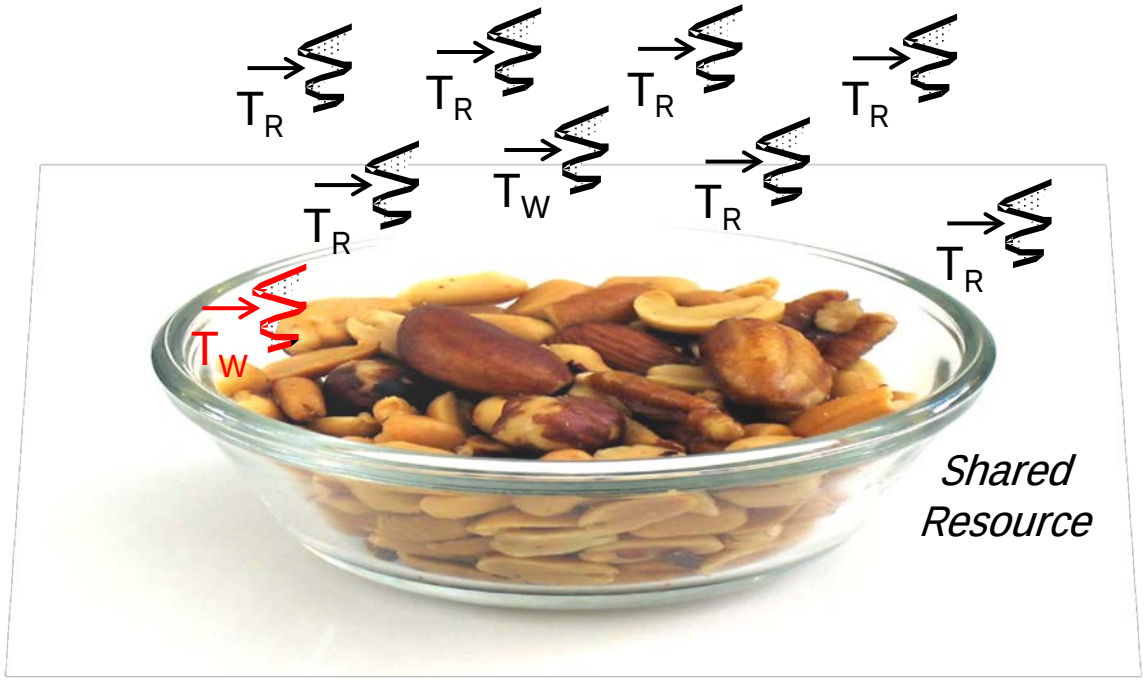
Overview of Readers-Writer Locks

- A readers-writer lock is a synchronization mechanism often used in contexts with many concurrent threads
- It allows access to a shared resource either by
  - Multiple threads that have read-only access or



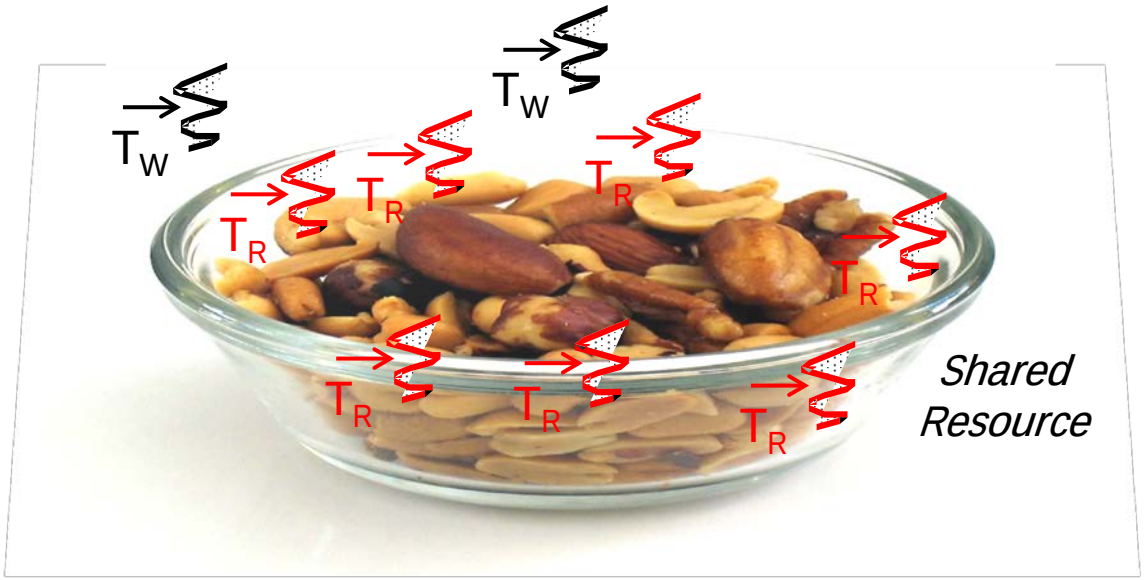
Overview of Readers-Writer Locks

- A readers-writer lock is a synchronization mechanism often used in contexts with many concurrent threads
- It allows access to a shared resource either by
  - Multiple threads that have read-only access or
  - Only one thread that has write access



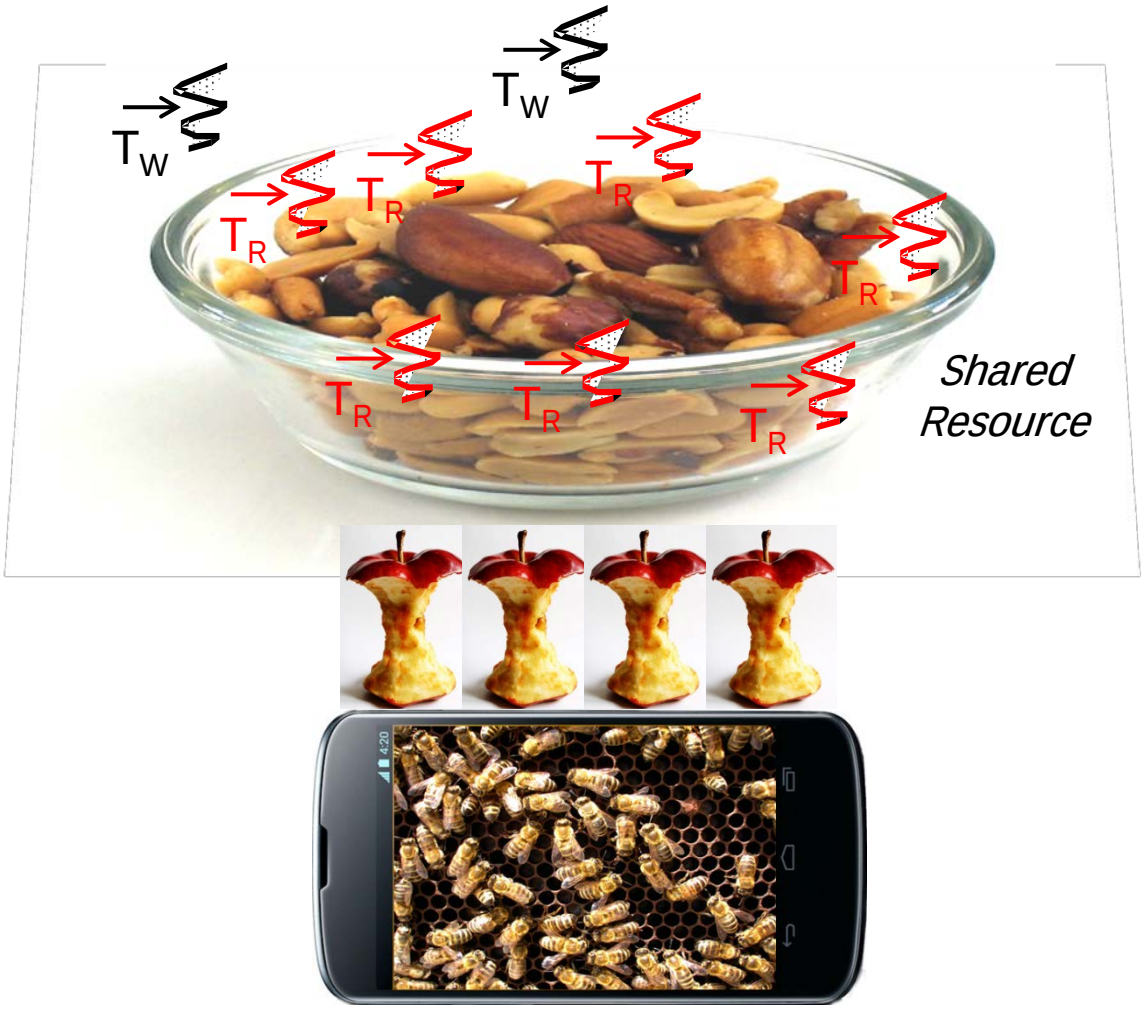
Overview of Readers-Writer Locks

- A readers-writer lock is a synchronization mechanism often used in contexts with many concurrent threads
- It allows access to a shared resource either by
- They may help improve performance when resources are *read* much more often than *written*



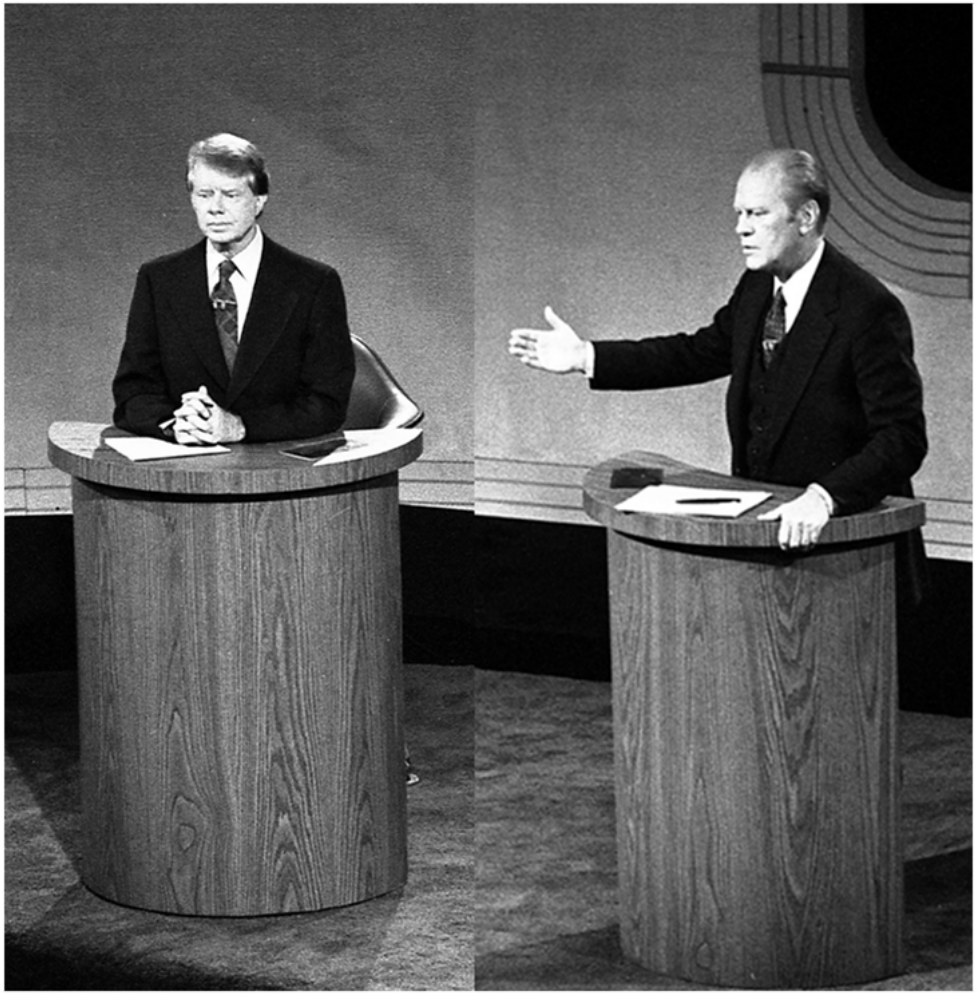
Overview of Readers-Writer Locks

- A readers-writer lock is a synchronization mechanism often used in contexts with many concurrent threads
- It allows access to a shared resource either by
- They may help improve performance when resources are *read* much more often than *written*
  - Especially on multi-core & multi-processor platforms



Overview of Readers-Writer Locks

- A readers-writer lock is a synchronization mechanism often used in contexts with many concurrent threads
- It allows access to a shared resource either by
- They may help improve performance when resources are *read* much more often than *writter*
- A human known use of the readers-writer locking protocol is formal political debates



See en.wikipedia.org/wiki/United_States_presidential_election_debates

Overview of ReentrantReadWriteLock

Overview of ReentrantReadWriteLock

- ReentrantReadWriteLocks implement the ReadWriteLock interface

ReentrantReadWriteLock

Added in API level 1

extends Object
implements Serializable ReadWriteLock

java.lang.Object
↳java.util.concurrent.locks.ReentrantReadWriteLock

Class Overview

An implementation of `ReadWriteLock` supporting similar semantics to `ReentrantLock`.

This class has the following properties:

- **Acquisition order**

  This class does not impose a reader or writer preference ordering for lock access. However, it does support an optional *fairness* policy.

  Non-fair mode (default)

  When constructed as non-fair (the default), the order of entry to the read and write lock is unspecified, subject to reentrancy constraints. A nonfair lock that is continuously contended may indefinitely postpone one or more reader or writer threads, but will normally have higher throughput than a fair lock.

  Fair mode

  When constructed as fair, threads contend for entry using an approximately arrival-order policy. When the currently held lock is released, either the longest-waiting single writer thread will be assigned the write lock, or if there is a group of reader threads waiting longer than all waiting writer threads, that group will be assigned the read lock.

  A thread that tries to acquire a fair read lock (non-reentrantly) will block if either the write lock is held, or there is a waiting writer thread. The thread will not acquire the read lock until after the oldest currently waiting writer thread has acquired and released the write lock. Of course, if a waiting writer abandons its wait, leaving one or more reader threads as the longest waiters in the queue with the write lock free, then those readers will be assigned the read lock.

developer.android.com/reference/java/util/concurrent/locks/ReentrantReadWriteLock.html

Overview of ReentrantReadWriteLock

- ReentrantReadWriteLocks implement the ReadWriteLock interface
  - Most of it is written in Java

```
public class ReentrantReadWriteLock
      implements ReadWriteLock ... {
  ...
```

See libcore/luni/src/main/java/java/util/concurrent/locks/ReentrantReadWriteLock.java

Overview of ReentrantReadWriteLock

- ReentrantReadWriteLocks implement the ReadWriteLock interface
- ReadLock & WriteLock implement the Lock interface

```
public class ReentrantReadWriteLock
    implements ReadWriteLock ... {
  ...
  /** Inner class providing
      readlock */
  ReentrantReadWriteLock.ReadLock
    readerLock;

  /** Inner class providing
      writelock */
  ReentrantReadWriteLock.WriteLock
    writerLock;
  ...
```

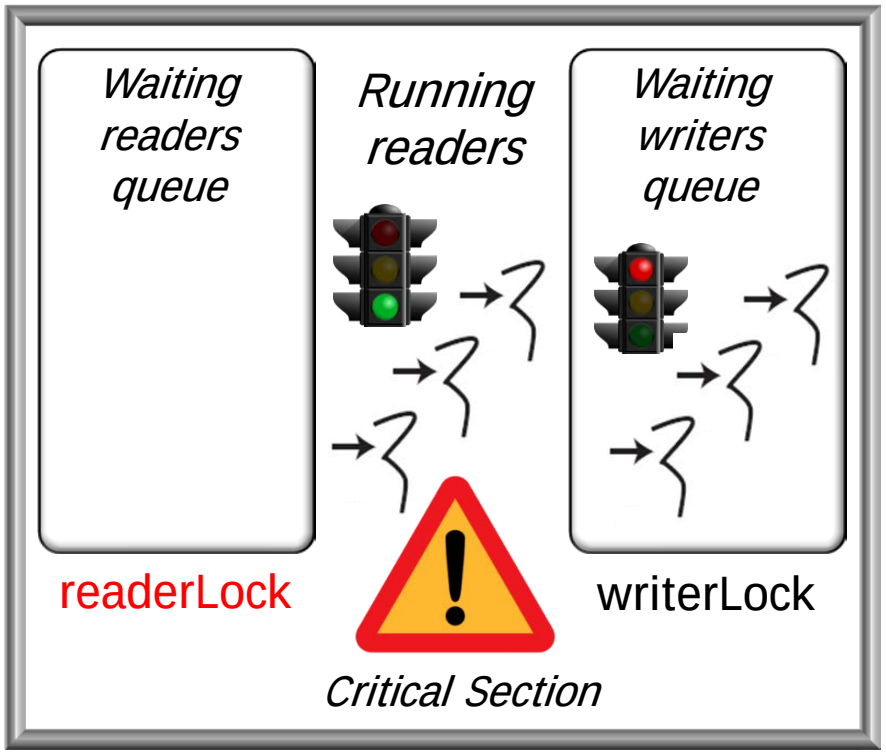
Overview of ReentrantReadWriteLock

- ReentrantReadWriteLocks implement the ReadWriteLock interface
- ReadLock & WriteLock implement the Lock interface

```
public class ReentrantReadWriteLock
    implements ReadWriteLock ... {
  ...
  /** Inner class providing
      readlock */
  ReentrantReadWriteLock.ReadLock
    readerLock;

  /** Inner class providing
      writelock */
  ReentrantReadWriteLock.WriteLock
    writerLock;
  ...
```

Overview of ReentrantReadWriteLock

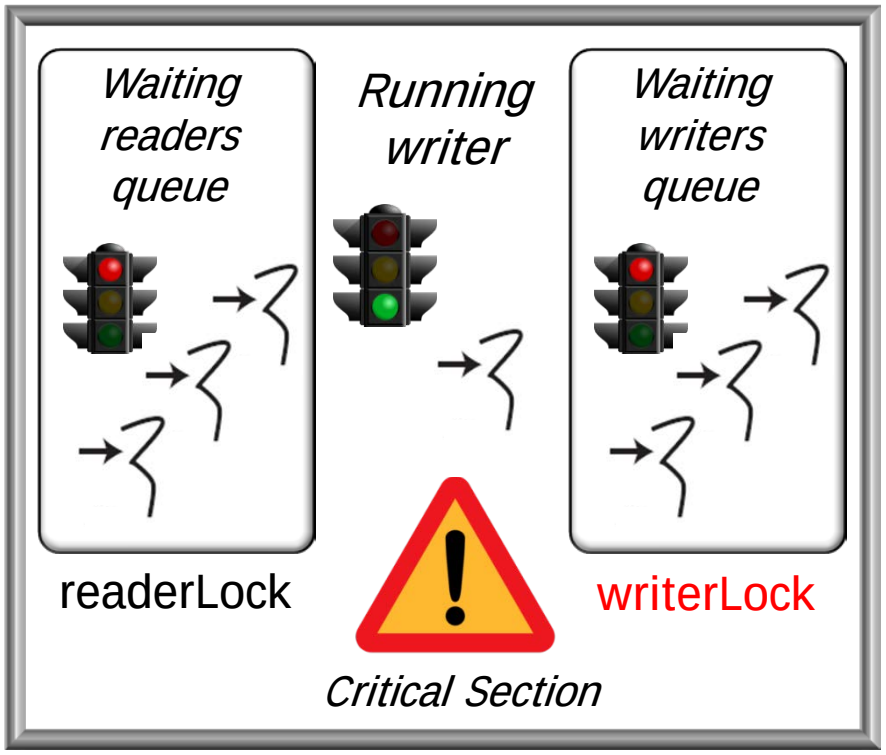
- ReentrantReadWriteLocks implement the ReadWriteLock interface
- ReadLock & WriteLock implement the Lock interface




```
public class ReentrantReadWriteLock
    implements ReadWriteLock ... {
  ...
  /** Inner class providing
      readlock */
ReentrantReadWriteLock.ReadLock
  readerLock;

/** Inner class providing
    writelock */
ReentrantReadWriteLock.WriteLock
  writerLock;
...
```

Overview of ReentrantReadWriteLock

- ReentrantReadWriteLocks implement the ReadWriteLock interface
- ReadLock & WriteLock implement the Lock interface

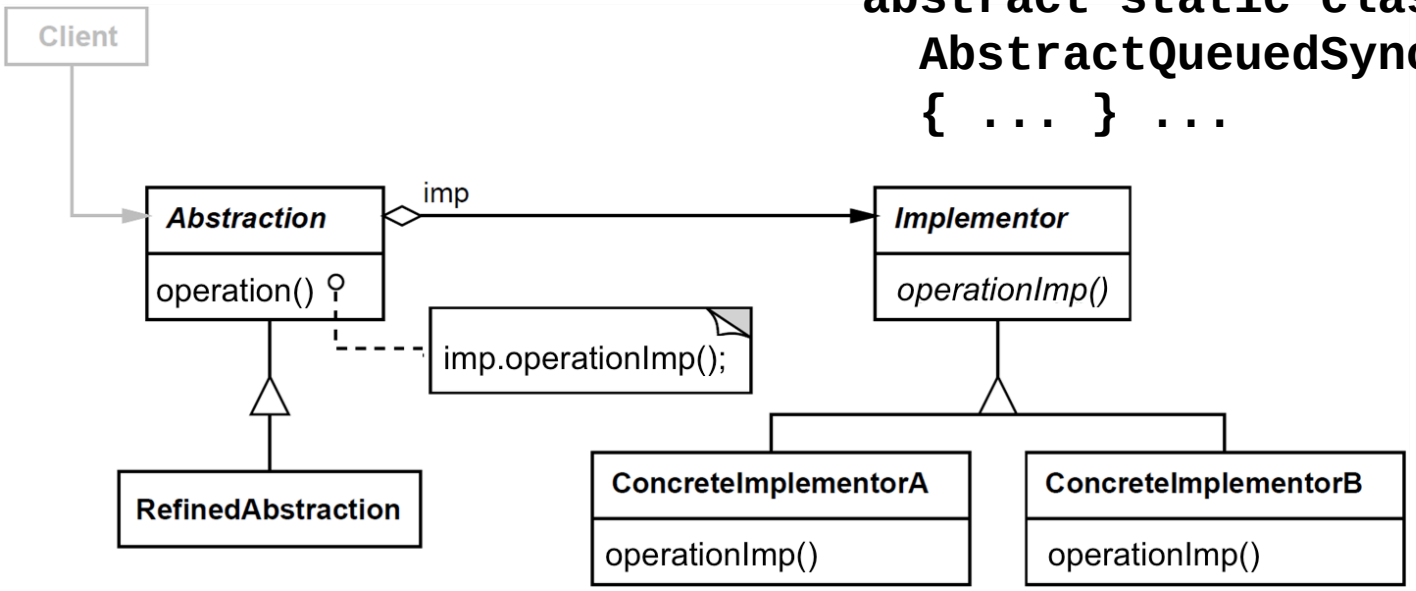



```
public class ReentrantReadWriteLock
    implements ReadWriteLock ... {
  ...
  /** Inner class providing
      readlock */
ReentrantReadWriteLock.ReadLock
  readerLock;

/** Inner class providing
    writelock */
ReentrantReadWriteLock.WriteLock
  writerLock;
...
```

Overview of ReentrantReadWriteLock

- ReentrantReadWriteLocks implement the ReadWriteLock interface
- ReadLock & WriteLock implement the Lock interface
- ReentrantReadWriteLock uses the *Bridge* pattern

```
public class ReentrantReadWriteLock
    implements ReadWriteLock ... {
  ...
  /** Performs sync mechanics */
  final Sync sync;

  /** Sync implementation for
      ReentrantReadWriteLock */
  abstract static class Sync extends
    AbstractQueuedSynchronizer
    { ... } ...
```



en.wikipedia.org/wiki/Bridge_pattern contains more on the *Bridge* pattern

Overview of ReentrantReadWriteLock

- ReentrantReadWriteLocks implement the ReadWriteLock interface
- ReadLock & WriteLock implement the Lock interface
- ReentrantReadWriteLock uses the *Bridge* pattern
  - Inherits functionality from AbstractQueuedSynchronizer

```
public class ReentrantReadWriteLock
    implements ReadWriteLock ... {
  ...
  /** Performs sync mechanics */
  final Sync sync;

  /** Sync implementation for
      ReentrantReadWriteLock */
  abstract static class Sync extends
    AbstractQueuedSynchronizer
    { ... } ...
```

Overview of ReentrantReadWriteLock

- ReentrantReadWriteLocks implement the ReadWriteLock interface
- ReadLock & WriteLock implement the Lock interface
- ReentrantReadWriteLock uses the *Bridge* pattern
  - Inherits functionality from AbstractQueuedSynchronizer
  - Optionally implement fair or non-fair lock acquisition model

```
public class ReentrantReadWriteLock
     implements ReadWriteLock ... {
  ...
  public ReentrantReadWriteLock
                      (boolean fair) {
    sync = fair ? new FairSync()
                : new NonfairSync();
    readerLock =
      new ReadLock(this);
    writerLock =
      new WriteLock(this);
  }
  ...
```

Overview of ReentrantReadWriteLock

- ReentrantReadWriteLocks implement the ReadWriteLock interface
- ReadLock & WriteLock implement the Lock interface
- ReentrantReadWriteLock uses the *Bridge* pattern
  - Inherits functionality from AbstractQueuedSynchronizer
  - Optionally implement fair or non-fair lock acquisition model

```
public class ReentrantReadWriteLock
    implements ReadWriteLock ... {
  ...
  public ReentrantReadWriteLock
                      (boolean fair) {
    sync = fair ? new FairSync()
                : new NonfairSync();
    readerLock =
      new ReadLock(this);
    writerLock =
      new WriteLock(this);
  }
  ...
```

See prior part on “Java ReentrantLock”

Overview of ReentrantReadWriteLock

- ReentrantReadWriteLocks implement the ReadWriteLock interface
- ReadLock & WriteLock implement the Lock interface
- ReentrantReadWriteLock uses the *Bridge* pattern
  - Inherits functionality from AbstractQueuedSynchronizer
  - Optionally implement fair or non-fair lock acquisition model
  - Initialize the readerLock & writerLock data members

```
public class ReentrantReadWriteLock
    implements ReadWriteLock ... {
  ...
  public ReentrantReadWriteLock
                      (boolean fair) {
    sync = fair ? new FairSync()
                : new NonfairSync();
    readerLock =
      new ReadLock(this);
    writerLock =
      new WriteLock(this);
  }
  ...
```

See developer.android.com/reference/java/util/concurrent/locks/Lock.html

Overview of ReentrantReadWriteLock

- ReentrantReadWriteLocks implement the ReadWriteLock interface
- ReadLock & WriteLock implement the Lock interface
- ReentrantReadWriteLock uses the *Bridge* pattern
- Its key methods are writeLock() & ReadLock()

```
public class ReentrantReadWriteLock
    implements ReadWriteLock ... {
  ...
  /** Returns the lock used for
      writing */
  public ReentrantReadWriteLock.
                          WriteLock
    writeLock() {
      return writerLock;
    }

  /** Returns the lock used for
      reading */
  public ReentrantReadWriteLock.
                          ReadLock
    readLock() {
      return readerLock;
    }
  ...
```

Overview of ReentrantReadWriteLock

- ReentrantReadWriteLocks implement the ReadWriteLock interface
- ReadLock & WriteLock implement the Lock interface
- ReentrantReadWriteLock uses the *Bridge* pattern
- Its key methods are writeLock() & ReadLock()

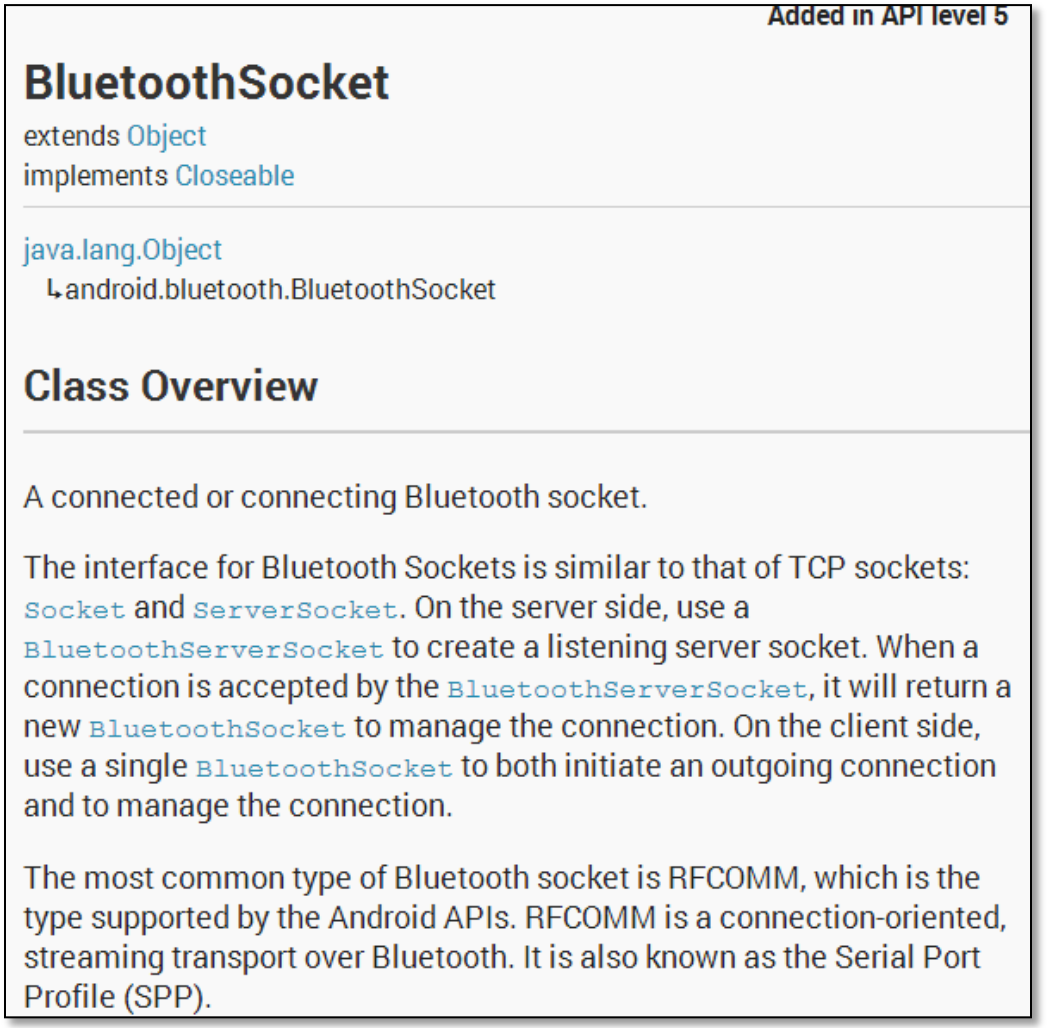
```
public class ReentrantReadWriteLock
    implements ReadWriteLock ... {
  ...
  /** Returns the lock used for
      writing */
  public ReentrantReadWriteLock.
                          WriteLock
    writeLock() {
      return writerLock;
    }

  /** Returns the lock used for
      reading */
  public ReentrantReadWriteLock.
                          ReadLock
    readLock() {
      return readerLock;
    }
  ...
```

Applications use objects returned from readLock() & writeLock() to access locks

Using ReentrantReadWriteLock in Android

Using ReentrantReadWriteLock in Android

- We show the BluetoothSocket's close() method



Added in API level 5

BluetoothSocket

extends Object
implements Closeable

java.lang.Object
↳android.bluetooth.BluetoothSocket

Class Overview

A connected or connecting Bluetooth socket.

The interface for Bluetooth Sockets is similar to that of TCP sockets: `Socket` and `ServerSocket`. On the server side, use a `BluetoothServerSocket` to create a listening server socket. When a connection is accepted by the `BluetoothServerSocket`, it will return a new `BluetoothSocket` to manage the connection. On the client side, use a single `BluetoothSocket` to both initiate an outgoing connection and to manage the connection.

The most common type of Bluetooth socket is RFCOMM, which is the type supported by the Android APIs. RFCOMM is a connection-oriented, streaming transport over Bluetooth. It is also known as the Serial Port Profile (SPP).

See developer.android.com/reference/android/bluetooth/BluetoothSocket.html

Using ReentrantReadWriteLock in Android

- We show the BluetoothSocket's close() method

```
public final class BluetoothSocket
                   implements Closeable {
  ...
```

See frameworks/base/core/java/android/bluetooth/BluetoothSocket.java

Using ReentrantReadWriteLock in Android

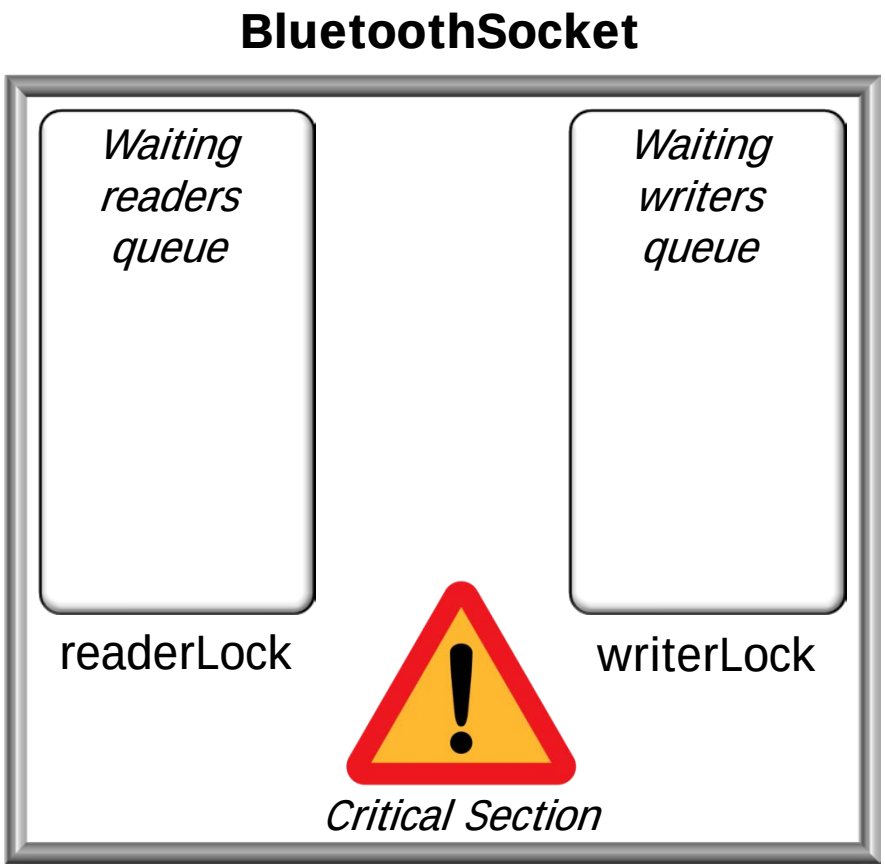
- We show the BluetoothSocket's close() method

```
public final class BluetoothSocket
                   implements Closeable {
...
```

See frameworks/base/core/java/android/bluetooth/BluetoothSocket.java

Using ReentrantReadWriteLock in Android

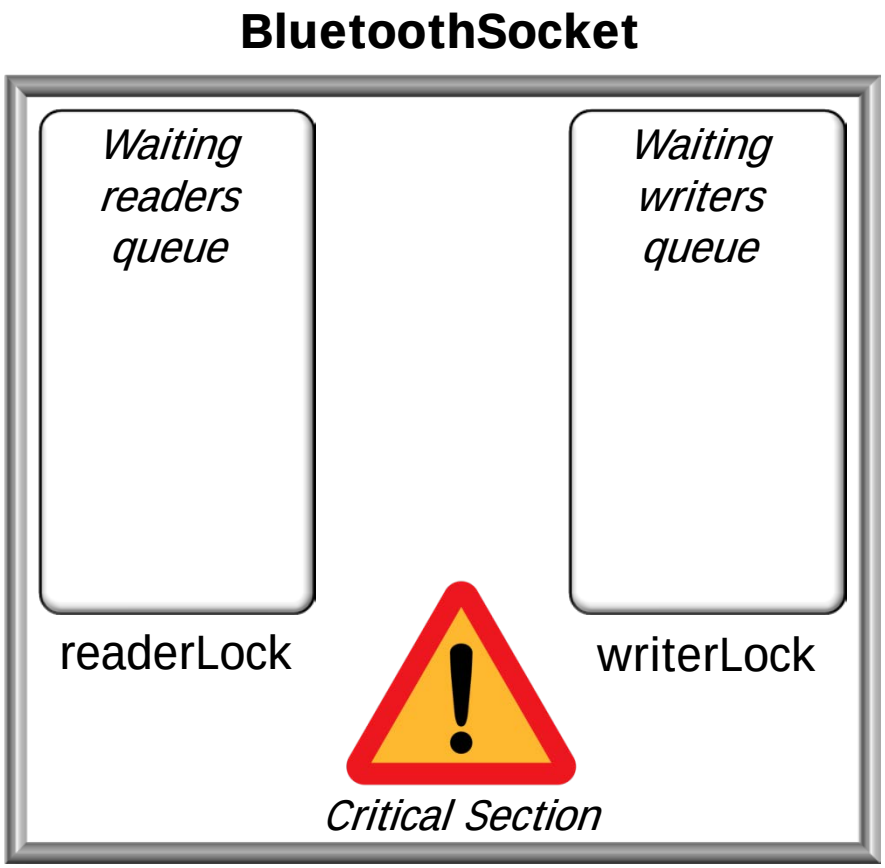
- We show the BluetoothSocket's close() method




```
public final class BluetoothSocket
                implements Closeable {
  ...
  ReentrantReadWriteLock mLock;

  BluetoothSocket(...) {
    ...
    mLock = new
      ReentrantReadWriteLock();
    ...
```

Using ReentrantReadWriteLock in Android

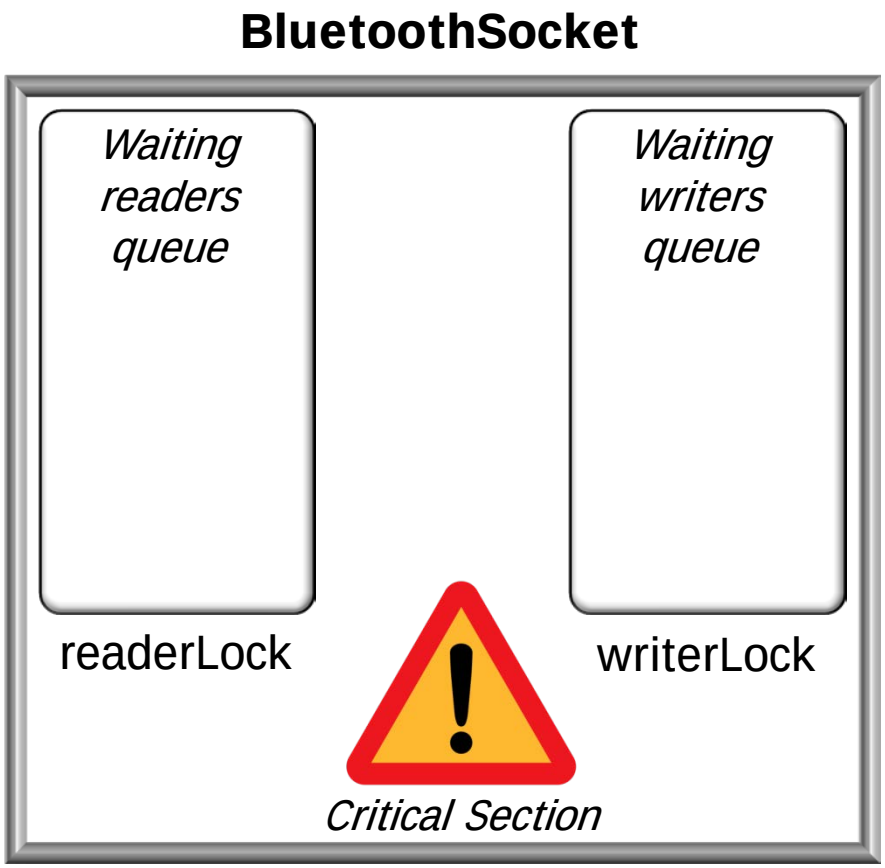
- We show the BluetoothSocket's close() method




```
public final class BluetoothSocket
                implements Closeable {
  ...
  ReentrantReadWriteLock mLock;

  BluetoothSocket(...) {
    ...
    mLock = new
      ReentrantReadWriteLock();
    ...
```

Using ReentrantReadWriteLock in Android

- We show the BluetoothSocket's close() method

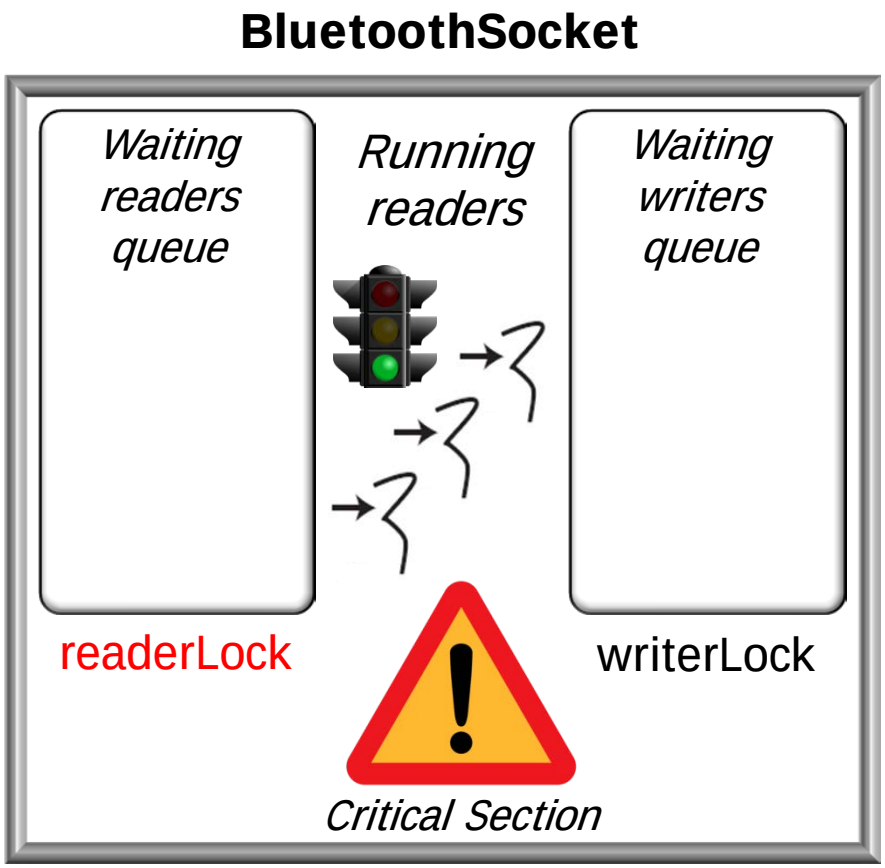



```
public final class BluetoothSocket
                   implements Closeable {
  ...
  public void close() ... {
    ...
```

It's safe to call close() multiple times on the same object

Using ReentrantReadWriteLock in Android

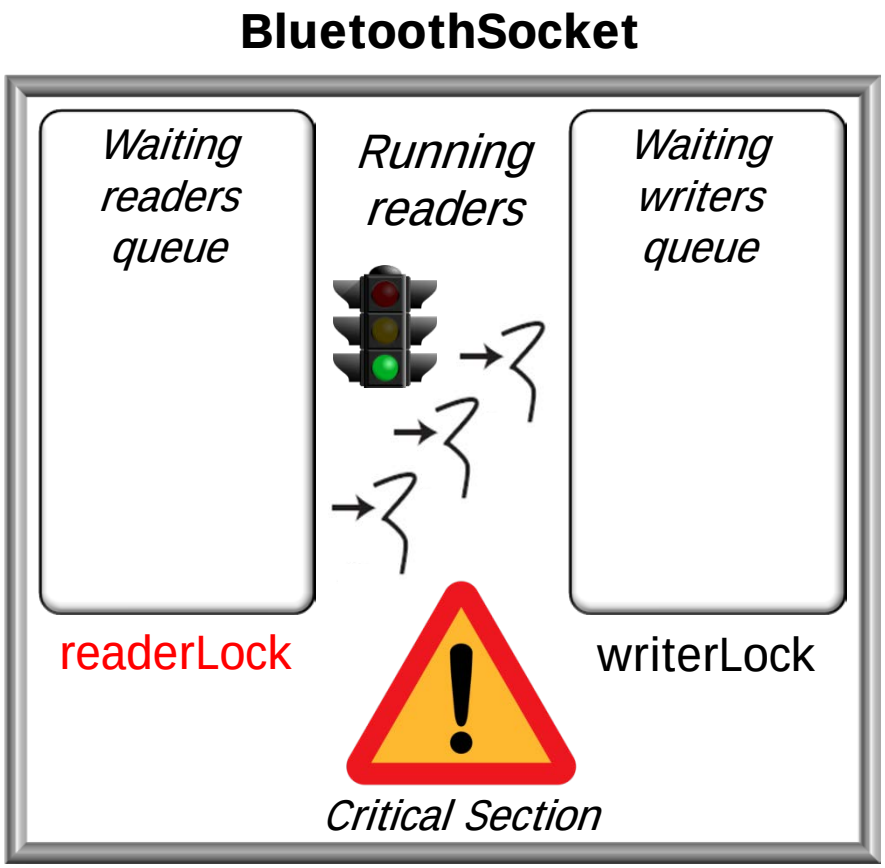
- We show the BluetoothSocket's close() method




```
public final class BluetoothSocket
                implements Closeable {
  ...
  public void close() ... {
    mLock.readLock().lock();
    try { ... }
    finally {
      mLock.readLock().unlock();
    }
    ...
```

Using ReentrantReadWriteLock in Android

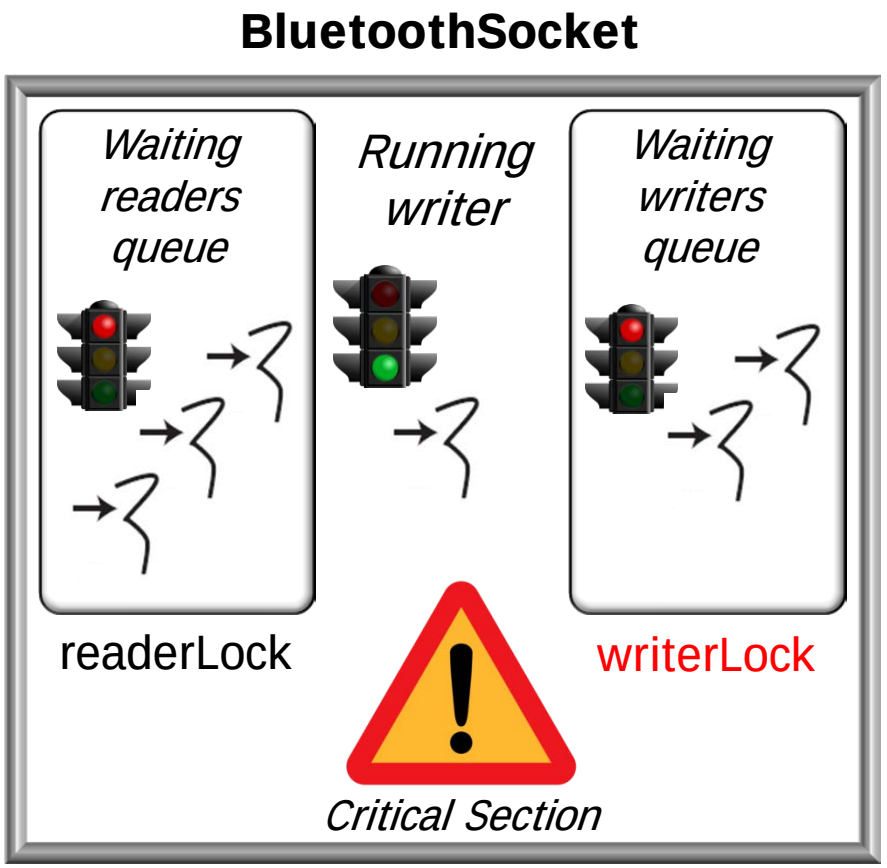
- We show the BluetoothSocket's close() method




```
public final class BluetoothSocket
                implements Closeable {
  ...
  public void close() ... {
    mLock.readLock().lock();
    try { ... }
    finally {
      mLock.readLock().unlock();
    }
    ...
```

Using ReentrantReadWriteLock in Android

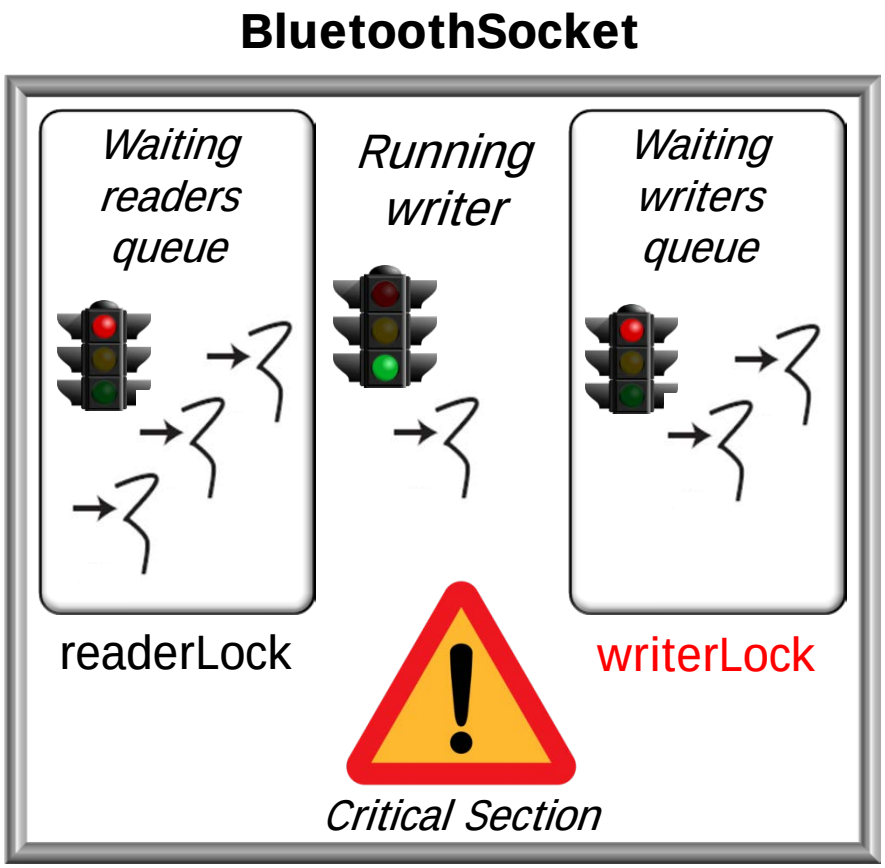
- We show the BluetoothSocket's close() method




```
public final class BluetoothSocket
                implements Closeable {
  ...
  public void close() ... {
    ...
    mLock.writeLock().lock();
    try { ... }
    finally {
      mLock.writeLock().unlock();
    }
    ...
```

Using ReentrantReadWriteLock in Android

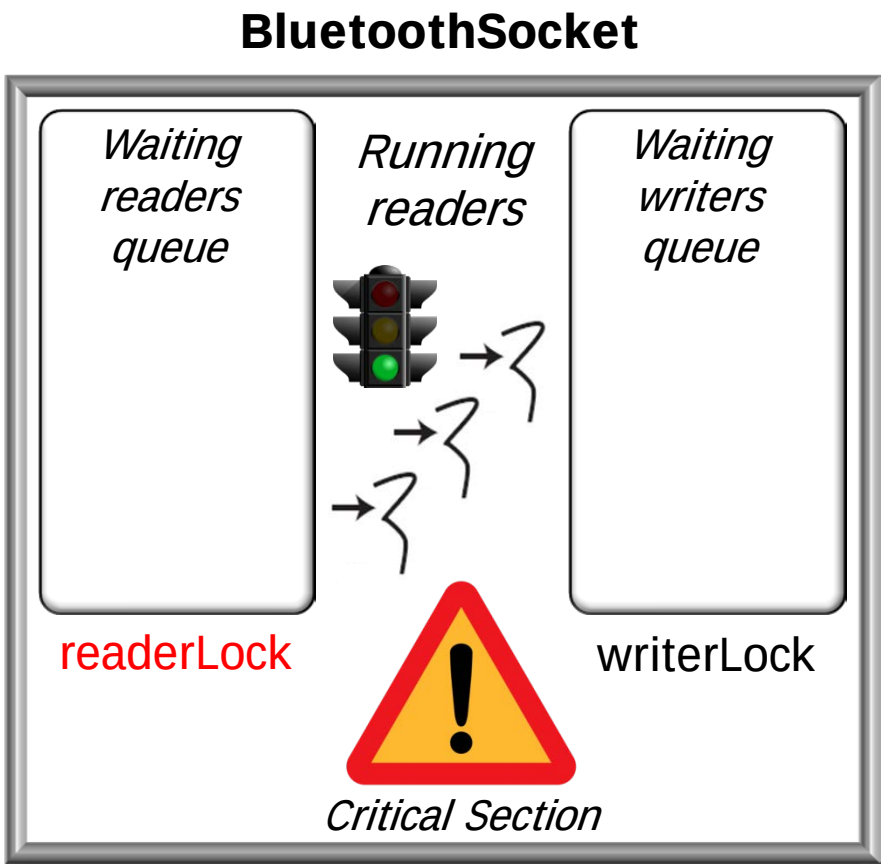
- We show the BluetoothSocket's close() method




```
public final class BluetoothSocket
                implements Closeable {
  ...
  public void close() ... {
    ...
    mLock.writeLock().lock();
    try { ... }
    finally {
      mLock.writeLock().unlock();
    }
    ...
```

Using ReentrantReadWriteLock in Android

- We show the BluetoothSocket's close() method

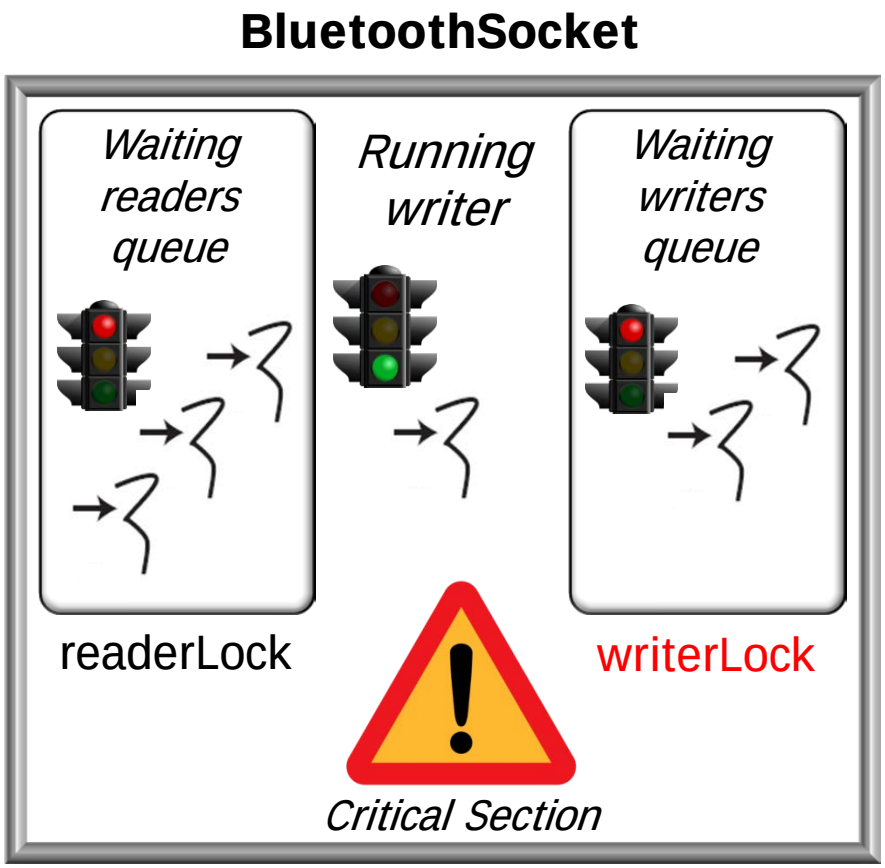



```
public final class BluetoothSocket
                   implements Closeable {
  ...
  ReentrantReadWriteLock mLock;
```

Multiple Threads can acquire mLock for reading

Using ReentrantReadWriteLock in Android

- We show the BluetoothSocket's close() method




```
public final class BluetoothSocket
                   implements Closeable {
  ...
  ReentrantReadWriteLock mLock;
```

Only on Thread at a time can acquire mLock for writing

Summary

Summary

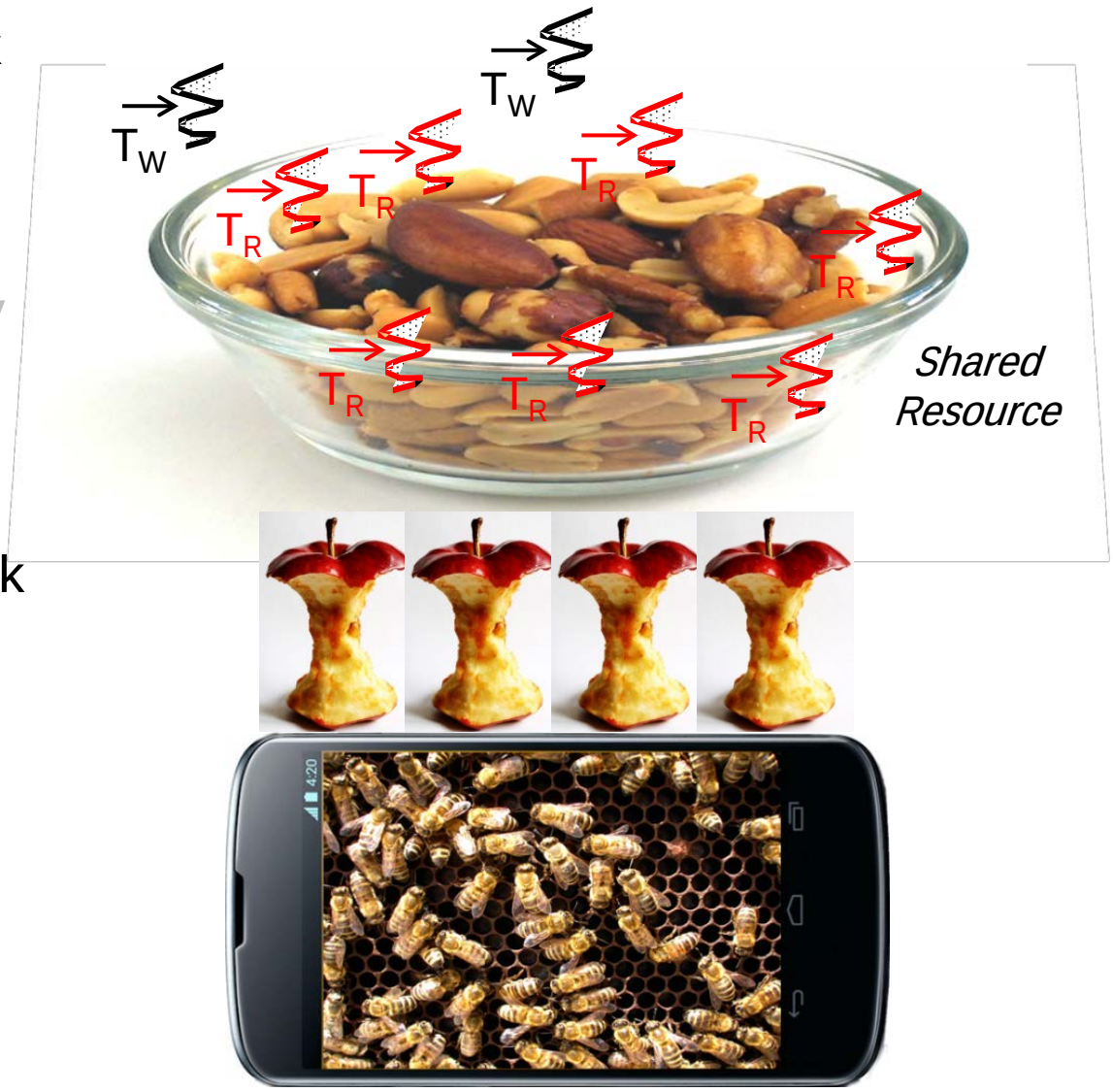
- Java ReentrantReadWriteLock allows greater concurrency when accessing shared data than a ReentrantLock

Summary

- Java ReentrantReadWriteLock allows greater concurrency when accessing shared data than a ReentrantLock
  - Speedups from concurrency will only be fully realized under certain conditions

Summary

- Java ReentrantReadWriteLock allows greater concurrency when accessing shared data than a ReentrantLock
  - Speedups from concurrency will only be fully realized under certain conditions
  - Profiling helps establish whether a readers-write lock is suitable for a particular use-case



www2.rdrop.com/~paulmck/scalability/paper/mutexprimpat.pdf has more

Summary

- Java ReentrantReadWriteLock allows greater concurrency when accessing shared data than a ReentrantLock
- ReentrantReadWriteLock supports a number of properties

- **Reentrancy**

  This lock allows both readers and writers to reacquire read or write locks in the style of a `ReentrantLock`. Non-reentrant readers are not allowed until all write locks held by the writing thread have been released.

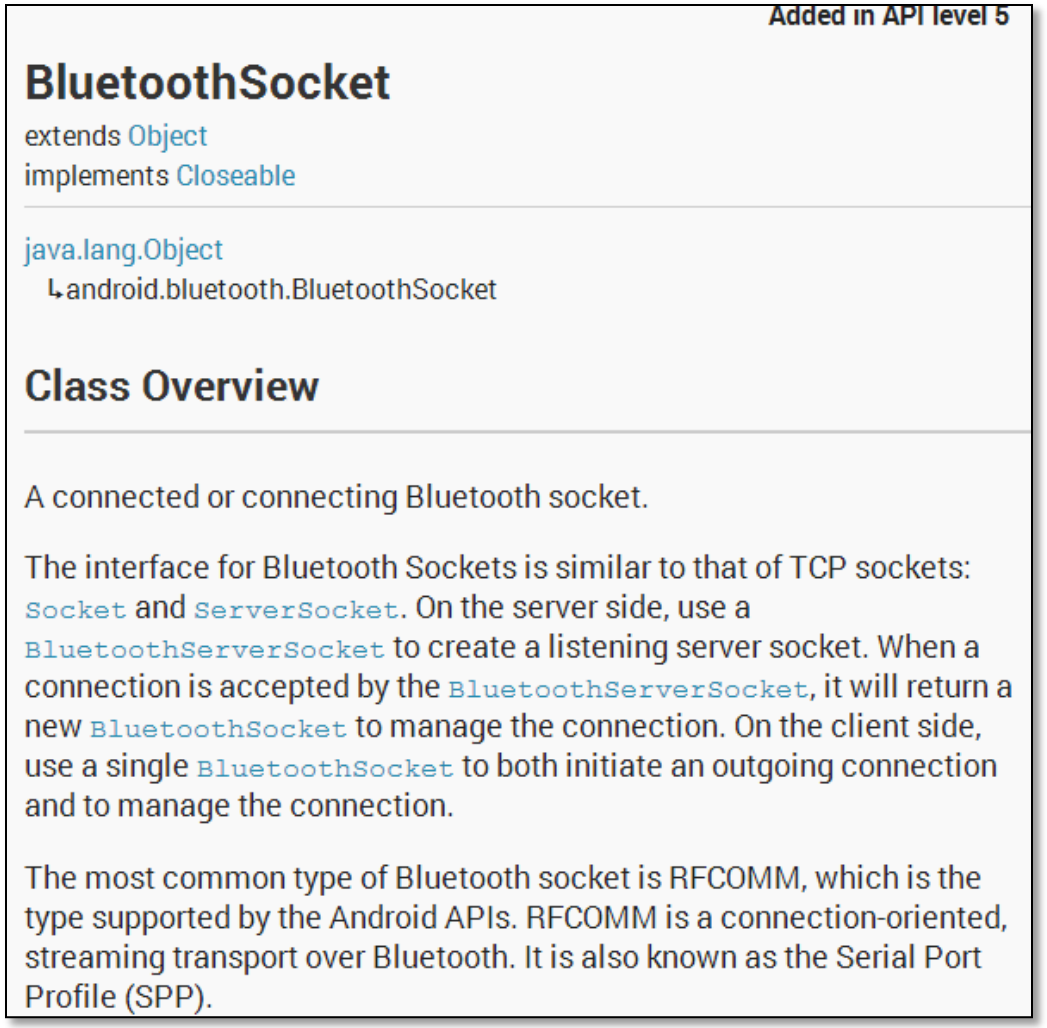
  Additionally, a writer can acquire the read lock, but not vice-versa. Among other applications, reentrancy can be useful when write locks are held during calls or callbacks to methods that perform reads under read locks. If a reader tries to acquire the write lock it will never succeed.
- **Lock downgrading**

  Reentrancy also allows downgrading from the write lock to a read lock, by acquiring the write lock, then the read lock and then releasing the write lock. However, upgrading from a read lock to the write lock is **not** possible.
- **Interruption of lock acquisition**

  The read lock and write lock both support interruption during lock acquisition.

Summary

- Java ReentrantReadWriteLock allows greater concurrency when accessing shared data than a ReentrantLock
- ReentrantReadWriteLocks supports a number of properties
- ReentrantReadWriteLock is typically used for more interesting applications that closing a BluetoothSocket!



Added in API level 5

BluetoothSocket

extends Object
implements Closeable

java.lang.Object
↳android.bluetooth.BluetoothSocket

Class Overview

A connected or connecting Bluetooth socket.

The interface for Bluetooth Sockets is similar to that of TCP sockets: `Socket` and `ServerSocket`. On the server side, use a `BluetoothServerSocket` to create a listening server socket. When a connection is accepted by the `BluetoothServerSocket`, it will return a new `BluetoothSocket` to manage the connection. On the client side, use a single `BluetoothSocket` to both initiate an outgoing connection and to manage the connection.

The most common type of Bluetooth socket is RFCOMM, which is the type supported by the Android APIs. RFCOMM is a connection-oriented, streaming transport over Bluetooth. It is also known as the Serial Port Profile (SPP).