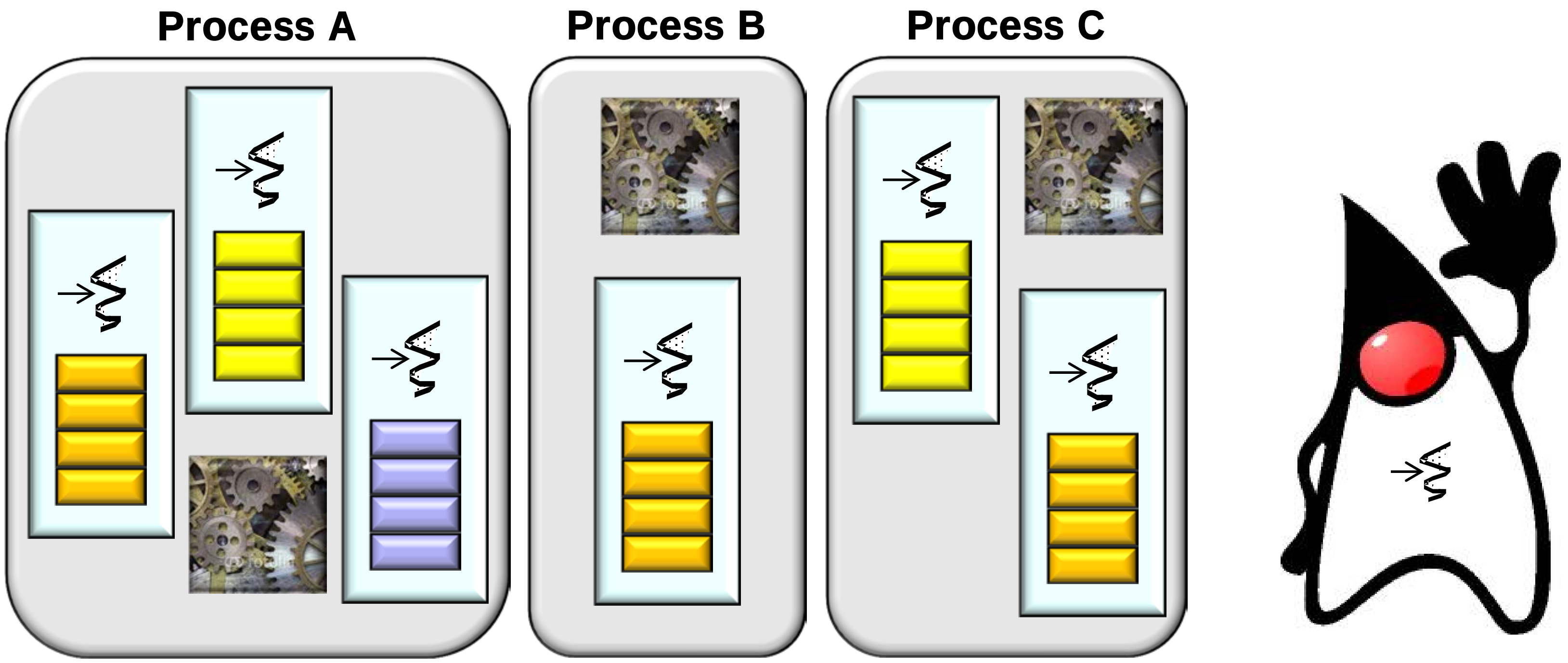
Java Concurrency: Overview of Java Threads

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

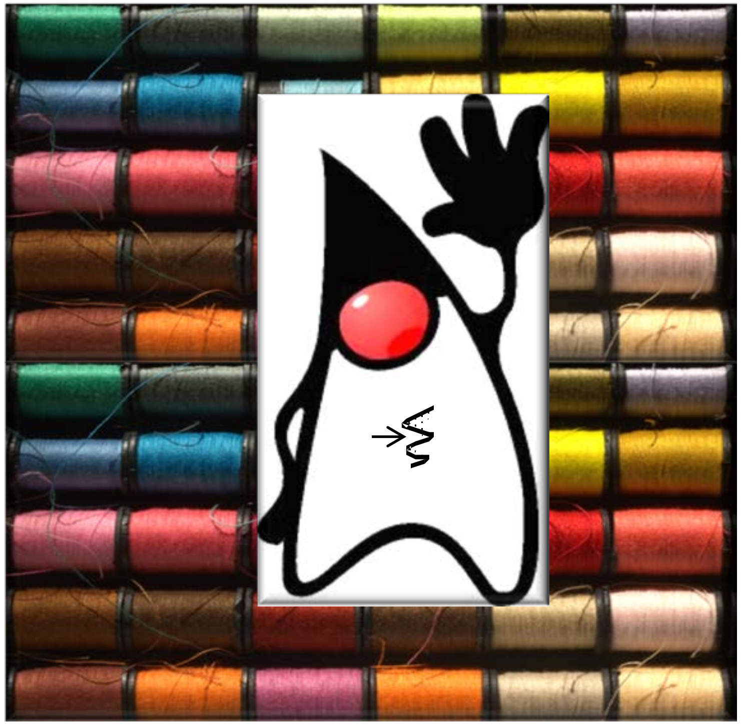
- Recognize the Java threading mechanisms available to program concurrent software



Alternative Ways of Giving Code to Java Threads

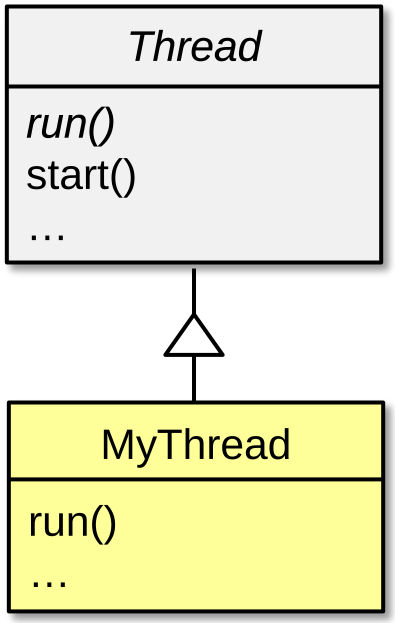
Alternative Ways of Giving Code to Java Threads

- Java Threads must be given code to run



Alternative Ways of Giving Code to Java Threads

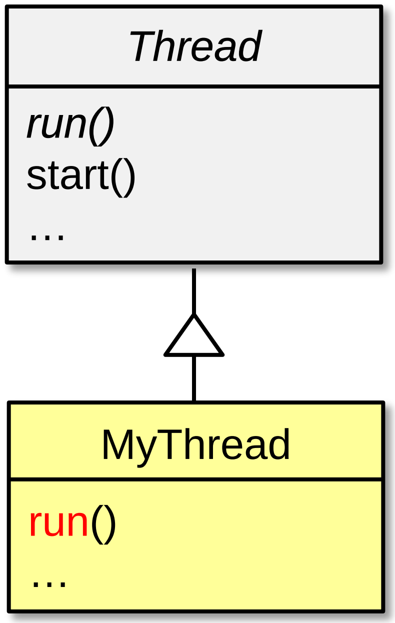
- Java Threads must be given code to run, e.g.
  - Extend the Thread class



See docs.oracle.com/javase/7/docs/api/java/lang/Thread.html

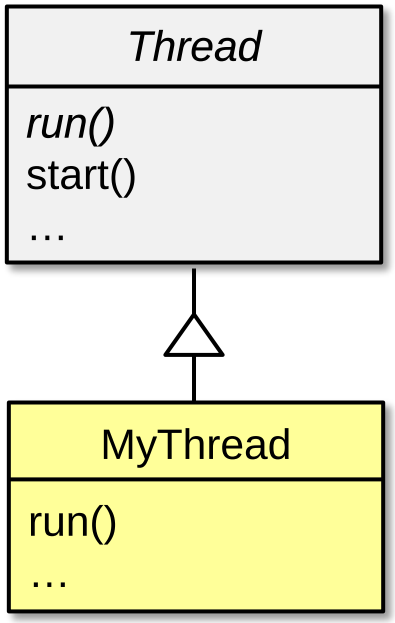
Alternative Ways of Giving Code to Java Threads

- Java Threads must be given code to run, e.g.
  - Extend the Thread class



Alternative Ways of Giving Code to Java Threads

- Java Threads must be given code to run, e.g.
  - Extend the Thread class

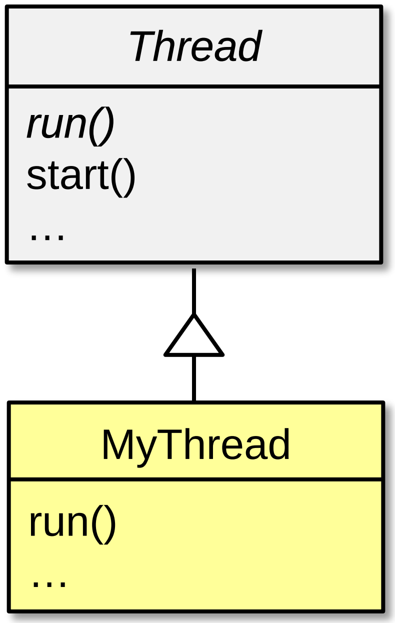


```
public class MyThread
                 extends Thread {
    public void run() {
        // code to run goes here
    }
}
```

Create & start a Thread using a named subclass of Thread

Alternative Ways of Giving Code to Java Threads

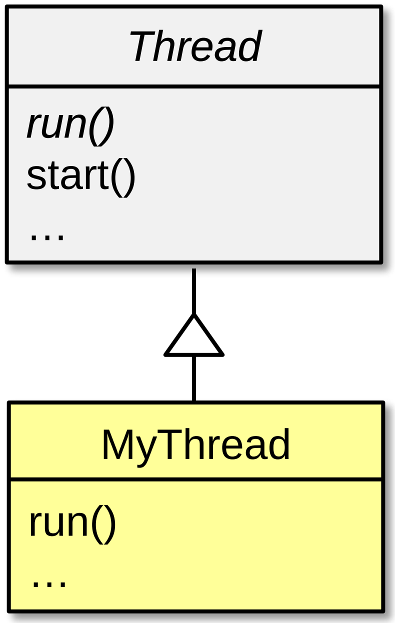
- Java Threads must be given code to run, e.g.
  - Extend the Thread class



```
public class MyThread
                extends Thread {
   public void run() {
      // code to run goes here
   }
}
```

Alternative Ways of Giving Code to Java Threads

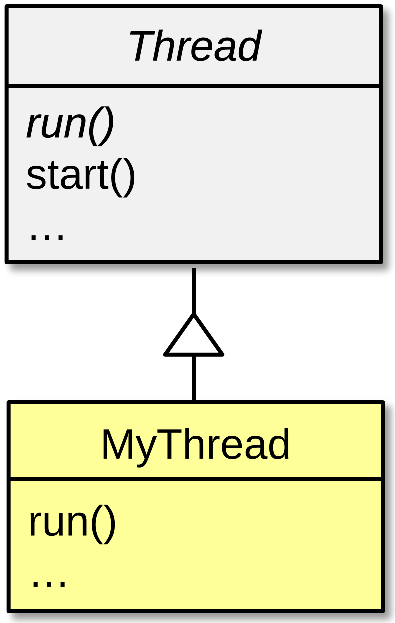
- Java Threads must be given code to run, e.g.
  - Extend the Thread class



```
public class MyThread
                 extends Thread {
    public void run() {
        // code to run goes here
    }
}
```

Alternative Ways of Giving Code to Java Threads

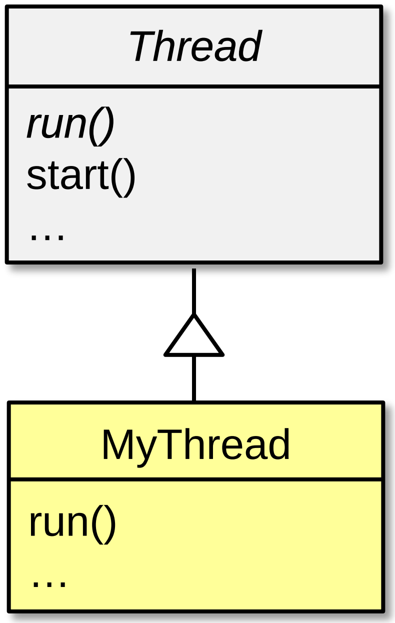
- Java Threads must be given code to run, e.g.
  - Extend the Thread class



```
public class MyThread
                 extends Thread {
    public void run() {
        // code to run goes here
    }
}
```

Alternative Ways of Giving Code to Java Threads

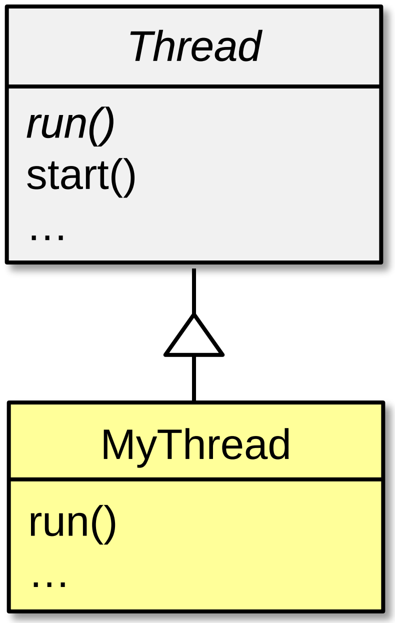
- Java Threads must be given code to run, e.g.
  - Extend the Thread class



```
public class MyThread
                 extends Thread {
    public void run() {
        // code to run goes here
    }
}

final MyThread myThread =
  new MyThread();
```

Alternative Ways of Giving Code to Java Threads

- Java Threads must be given code to run, e.g.
  - Extend the Thread class

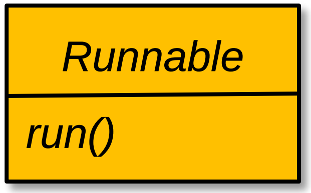


```
public class MyThread
                 extends Thread {
   public void run() {
      // code to run goes here
   }
}

final MyThread myThread =
  new MyThread();
myThread.start();
```

You can also write a one-liner:
`new MyThread().start()`

Alternative Ways of Giving Code to Java Threads

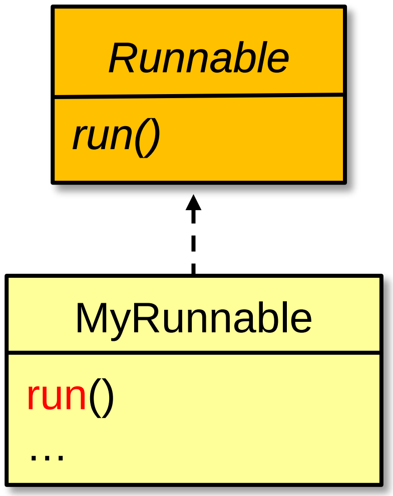
- Java Threads must be given code to run, e.g.
  - Extend the Thread class
  - Implement the Runnable interface



See docs.oracle.com/javase/tutorial/essential/concurrency/runThread.html

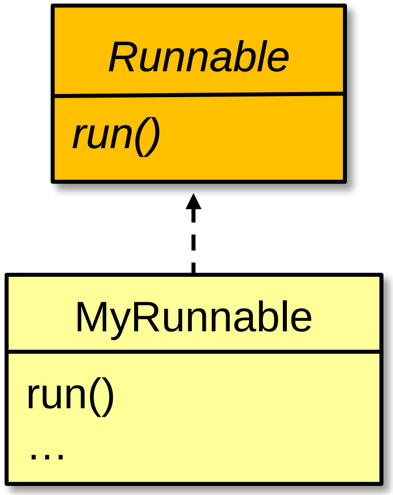
Alternative Ways of Giving Code to Java Threads

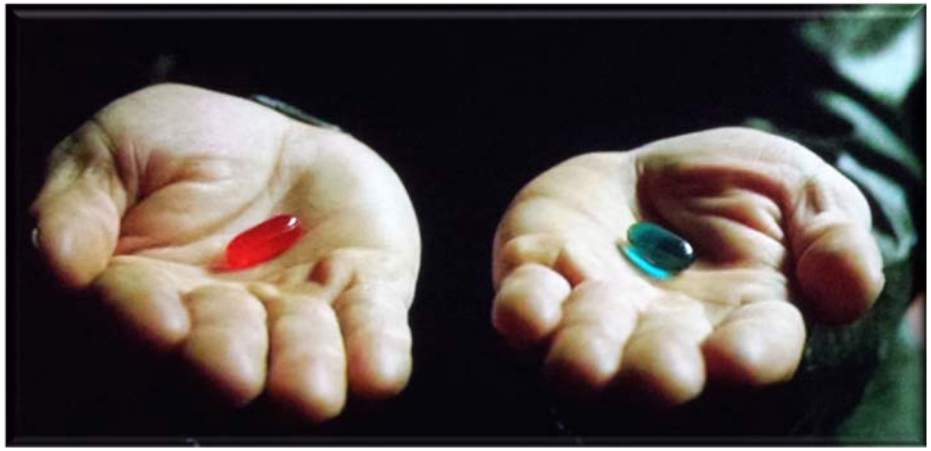
- Java Threads must be given code to run, e.g.
  - Extend the Thread class
  - Implement the Runnable interface



Alternative Ways of Giving Code to Java Threads

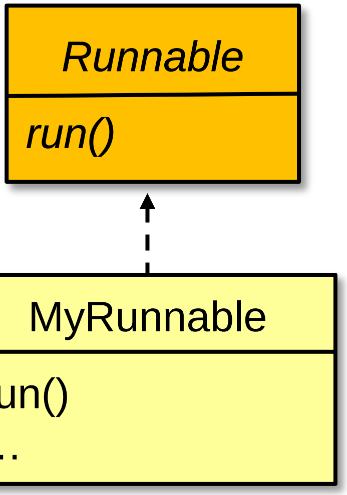
- Java Threads must be given code to run, e.g.
  - Extend the Thread class
  - Implement the Runnable interface





Alternative Ways of Giving Code to Java Threads

- Java Threads must be given code to run, e.g.
  - Extend the Thread class
  - Implement the Runnable interface

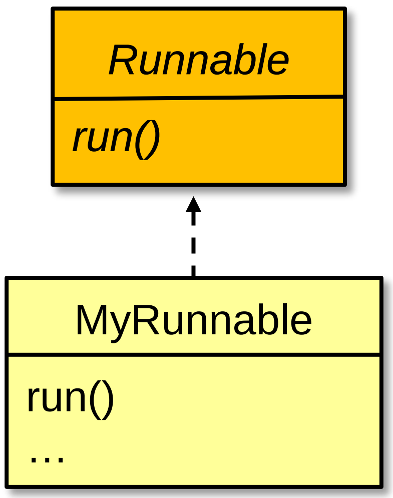


```
public class MyRunnable
       implements Runnable {
  public void run() {
    // code to run goes here
  }
}
```

Create & start a Thread using
a named class as the Runnable

Alternative Ways of Giving Code to Java Threads

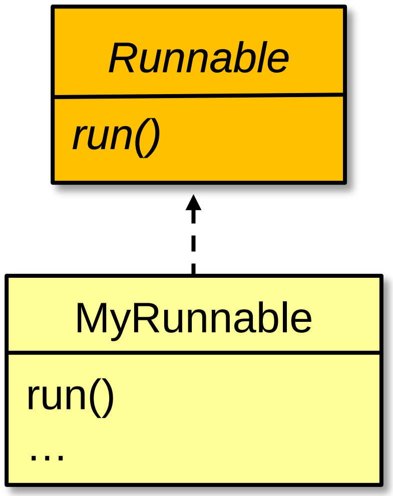
- Java Threads must be given code to run, e.g.
  - Extend the Thread class
  - Implement the Runnable interface



```
public class MyRunnable
       implements Runnable {
  public void run() {
    // code to run goes here
  }
}
```

Alternative Ways of Giving Code to Java Threads

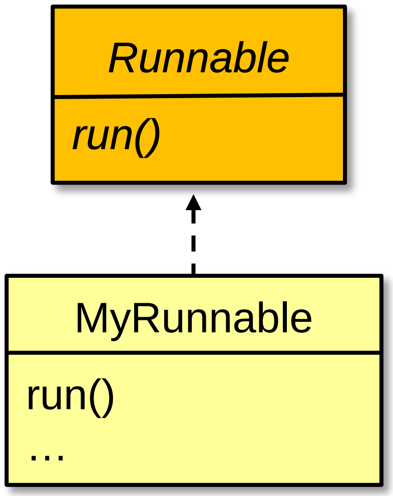
- Java Threads must be given code to run, e.g.
  - Extend the Thread class
  - Implement the Runnable interface



```
public class MyRunnable
       implements Runnable {
  public void run() {
    // code to run goes here
  }
}
```

Alternative Ways of Giving Code to Java Threads

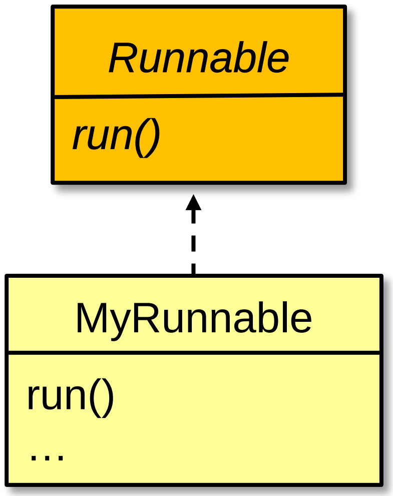
- Java Threads must be given code to run, e.g.
  - Extend the Thread class
  - Implement the Runnable interface



```
public class MyRunnable
       implements Runnable {
  public void run() {
    // code to run goes here
  }
}
```

Alternative Ways of Giving Code to Java Threads

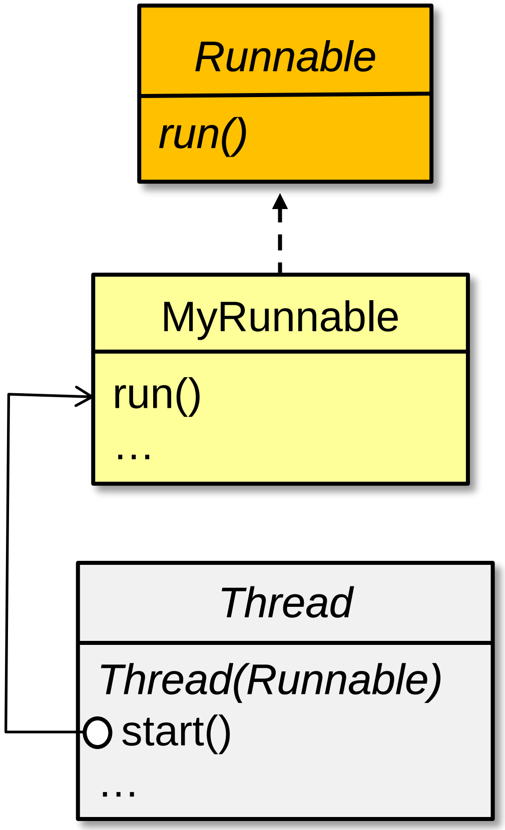
- Java Threads must be given code to run, e.g.
  - Extend the Thread class
  - Implement the Runnable interface



```
public class MyRunnable
       implements Runnable {
  public void run() {
    // code to run goes here
  }
}

final Runnable myRunnable =
  new MyRunnable();
```

Alternative Ways of Giving Code to Java Threads

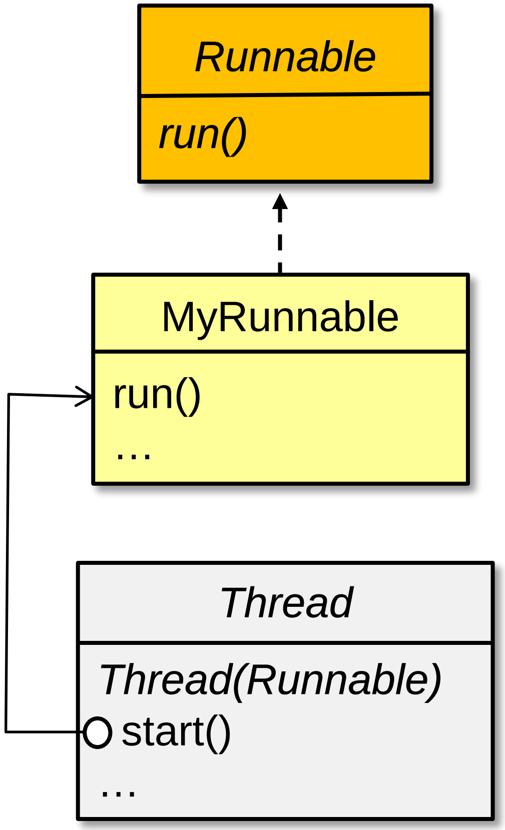
- Java Threads must be given code to run, e.g.
  - Extend the Thread class
  - Implement the Runnable interface



```
public class MyRunnable
       implements Runnable {
  public void run() {
    // code to run goes here
  }
}

final Runnable myRunnable =
  new MyRunnable();
new Thread(myRunnable).start();
```

Alternative Ways of Giving Code to Java Threads

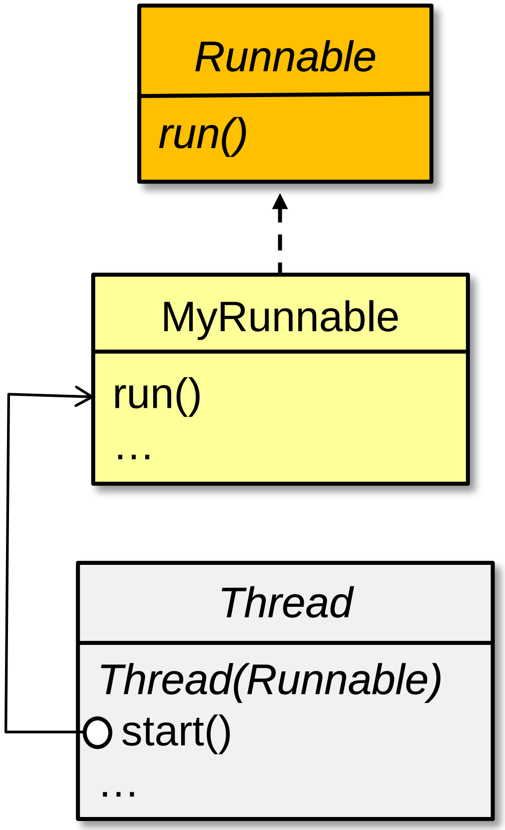
- Java Threads must be given code to run, e.g.
  - Extend the Thread class
  - Implement the Runnable interface



```
public class MyRunnable
       implements Runnable {
  public void run() {
    // code to run goes here
  }
}

final Runnable myRunnable =
  new MyRunnable();
new Thread(myRunnable).start();
```

Alternative Ways of Giving Code to Java Threads

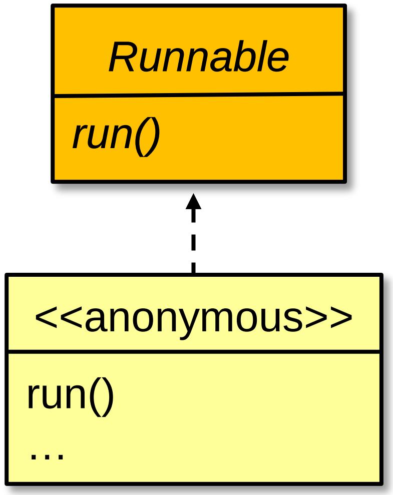
- Java Threads must be given code to run, e.g.
  - Extend the Thread class
  - Implement the Runnable interface



```
public class MyRunnable
       implements Runnable {
  public void run() {
    // code to run goes here
  }
}

final Runnable myRunnable =
  new MyRunnable();
new Thread(myRunnable).start();
```

Alternative Ways of Giving Code to Java Threads

- Java Threads must be given code to run, e.g.
  - Extend the Thread class
  - Implement the Runnable interface

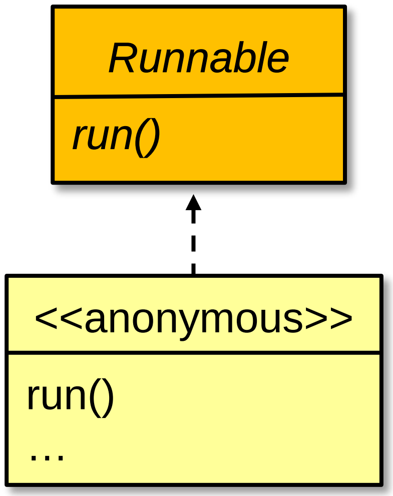


```
public interface Runnable {
   public void run();
}

new Thread(new Runnable() {
    public void run(){
      // code to run goes here
  }
}).start();
```

Create & start a Thread using an anonymous inner class as the Runnable

Alternative Ways of Giving Code to Java Threads

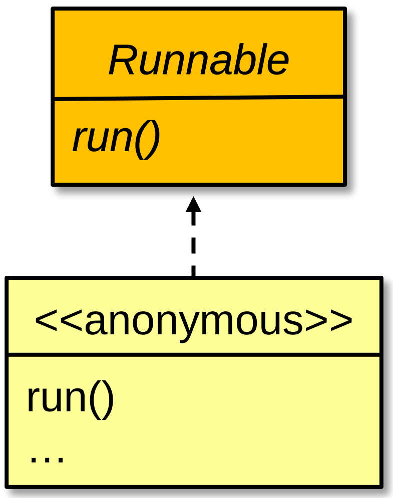
- Java Threads must be given code to run, e.g.
  - Extend the Thread class
  - Implement the Runnable interface



```
public interface Runnable {
   public void run();
}

new Thread(new Runnable() {
    public void run(){
      // code to run goes here
  }
}).start();
```

Alternative Ways of Giving Code to Java Threads

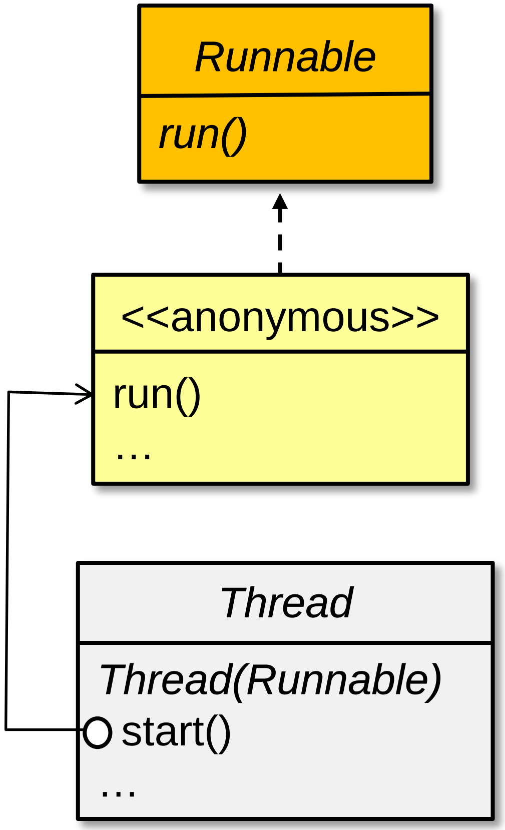
- Java Threads must be given code to run, e.g.
  - Extend the Thread class
  - Implement the Runnable interface



```
public interface Runnable {
   public void run();
}

new Thread(new Runnable() {
    public void run(){
      // code to run goes here
  }
}).start();
```

Alternative Ways of Giving Code to Java Threads

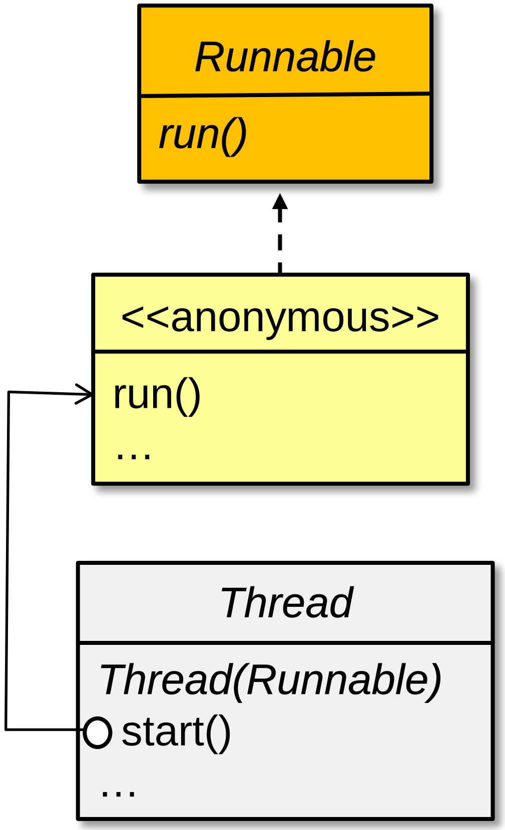
- Java Threads must be given code to run, e.g.
  - Extend the Thread class
  - Implement the Runnable interface



```
public interface Runnable {
   public void run();
}

new Thread(new Runnable() {
    public void run(){
      // code to run goes here
  }
}).start();
```

Alternative Ways of Giving Code to Java Threads

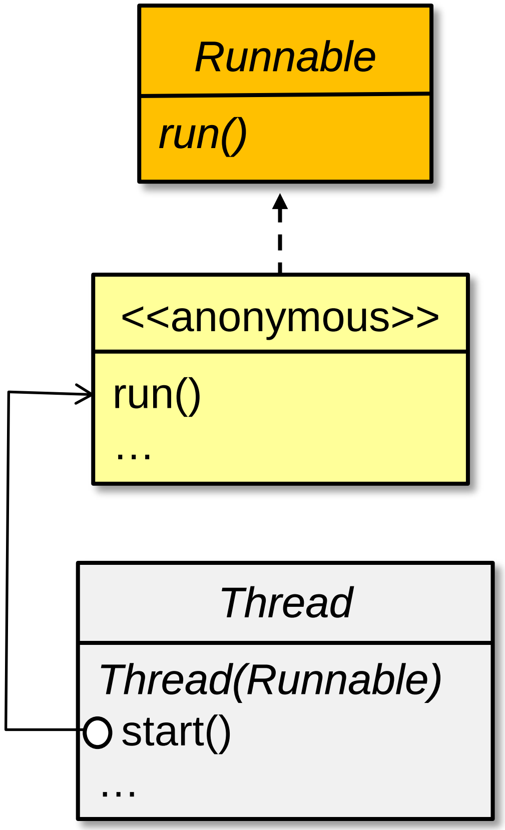
- Java Threads must be given code to run, e.g.
  - Extend the Thread class
  - Implement the Runnable interface



```
public interface Runnable {
   public void run();
}

new Thread(new Runnable() {
    public void run(){
      // code to run goes here
  }
}).start();
```

Alternative Ways of Giving Code to Java Threads

- Java Threads must be given code to run, e.g.
  - Extend the Thread class
  - Implement the Runnable interface

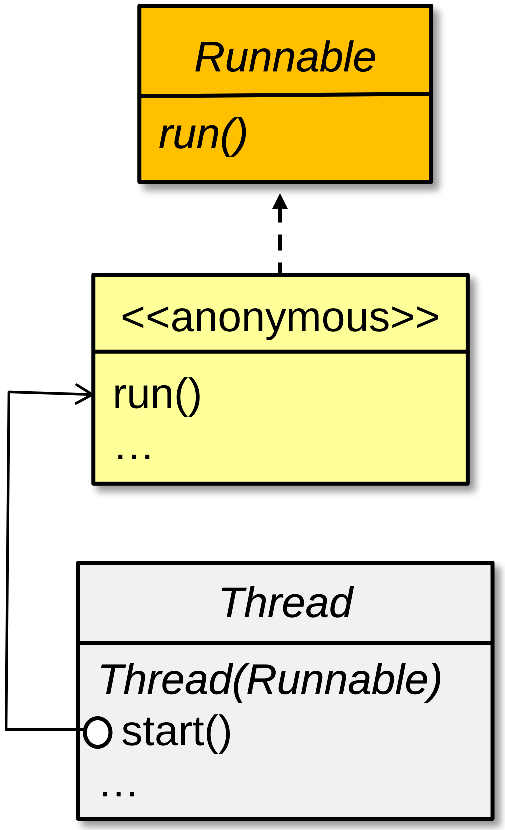


```
public interface Runnable {
   public void run();
}

new Thread(new Runnable() {
    public void run(){
      // code to run goes here
  }
}).start();
```

This anonymous inner class idiom is used extensively in Java code

Alternative Ways of Giving Code to Java Threads

- Java Threads must be given code to run, e.g.
  - Extend the Thread class
  - Implement the Runnable interface
  - Use Java 8 lambda expressions



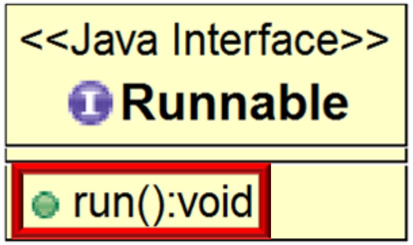
```
public interface Runnable {
   public void run();
}

new Thread(() -> {
      // code to run goes here
}).start();
```

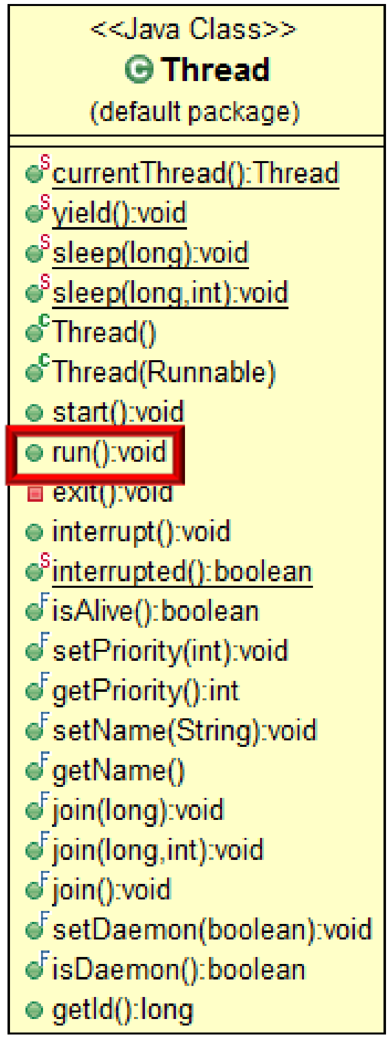
Java 8 lambda expressions are not officially supported in Android (yet)

Passing Parameters to a Started Thread

Passing Parameters to a Started Java Thread

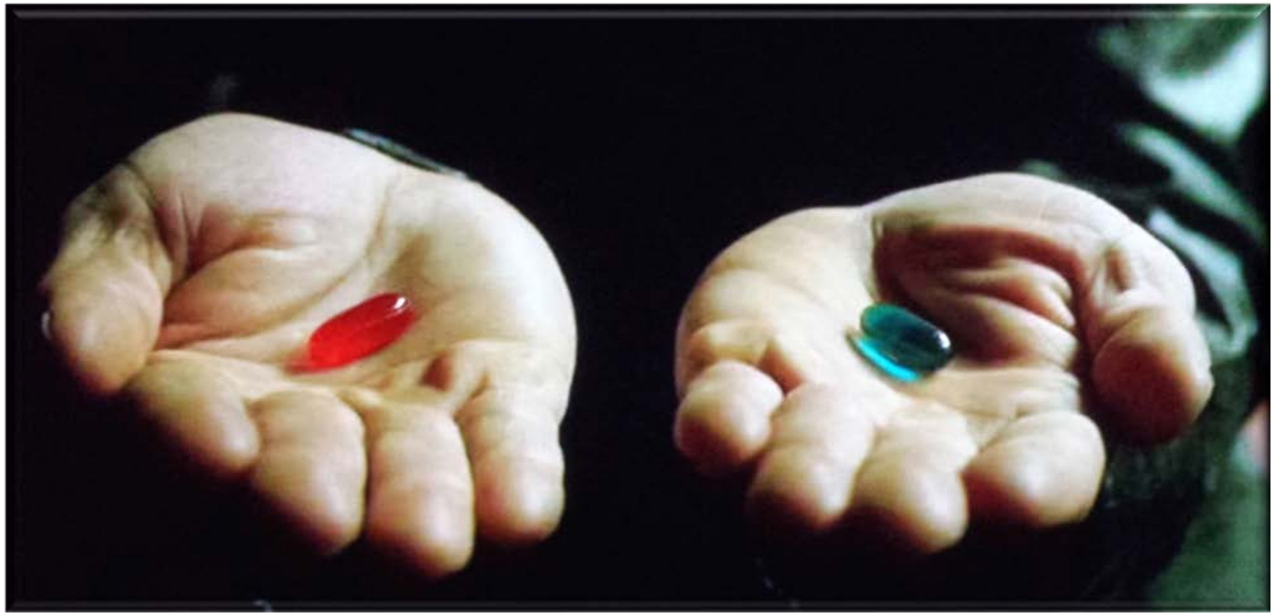
- The run() methods defined in Java Thread & Runnable take no parameters





Passing Parameters to a Started Java Thread

- The run() methods defined in Java Thread & Runnable take no parameters
- Parameters passed to run() can be supplied via one of two other means



Passing Parameters to a Started Java Thread

- The run() methods defined in Java Thread & Runnable take no parameters
- Parameters passed to run() can be supplied via one of two other means, e.g.
    - As parameters to a class constructor

```
public class GCDRunnable extends Random implements Runnable {
  final private String mThreadType;

  public GCDRunnable(String threadType) {
      mThreadType = threadType;
  }

  public void run() {
    final String threadString =
      " with " + mThreadType + " thread id " +
      Thread.currentThread();
    ...
```

Passing Parameters to a Started Java Thread

- The run() methods defined in Java Thread & Runnable take no parameters
- Parameters passed to run() can be supplied via one of two other means, e.g.
  - As parameters to a class constructor

```
public class GCDRunnable extends Random implements Runnable {
  final private String mThreadType;

  public GCDRunnable(String threadType) {
      mThreadType = threadType;
  }

  public void run() {
    final String threadString =
      " with " + mThreadType + " thread id " +
      Thread.currentThread();
    ...
```

Passing Parameters to a Started Java Thread

- The run() methods defined in Java Thread & Runnable take no parameters
- Parameters passed to run() can be supplied via one of two other means, e.g.
  - As parameters to a class constructor
  - As parameters to “setter” methods

```
public class GCDRunnable extends Random implements Runnable {
  final private String mThreadType;

  public void setThreadType(String threadType) {
      mThreadType = threadType;
  }

  public void run() {
    final String threadString =
      " with " + mThreadType + " thread id " +
      Thread.currentThread();
    ...
```

Passing Parameters to a Started Java Thread

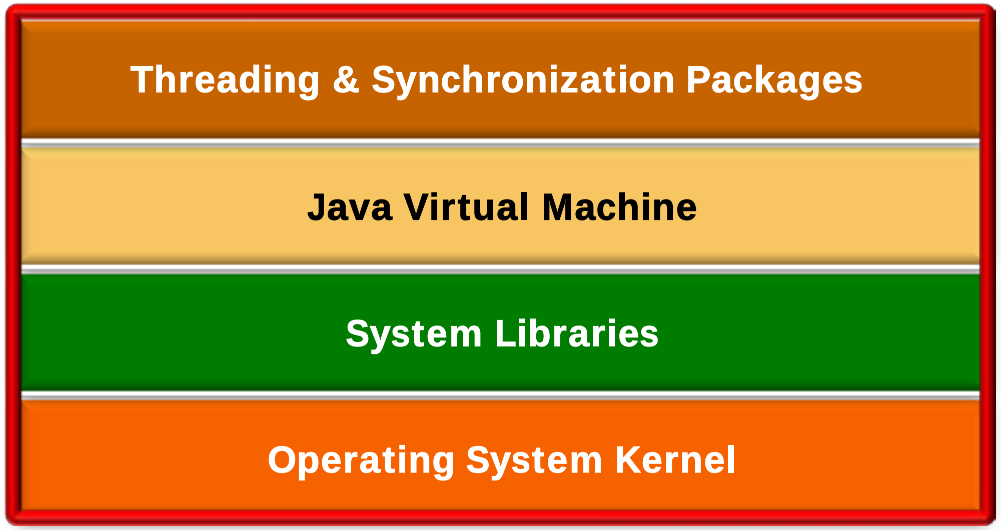
- The run() methods defined in Java Thread & Runnable take no parameters
- Parameters passed to run() can be supplied via one of two other means, e.g.
  - As parameters to a class constructor
  - As parameters to “setter” methods

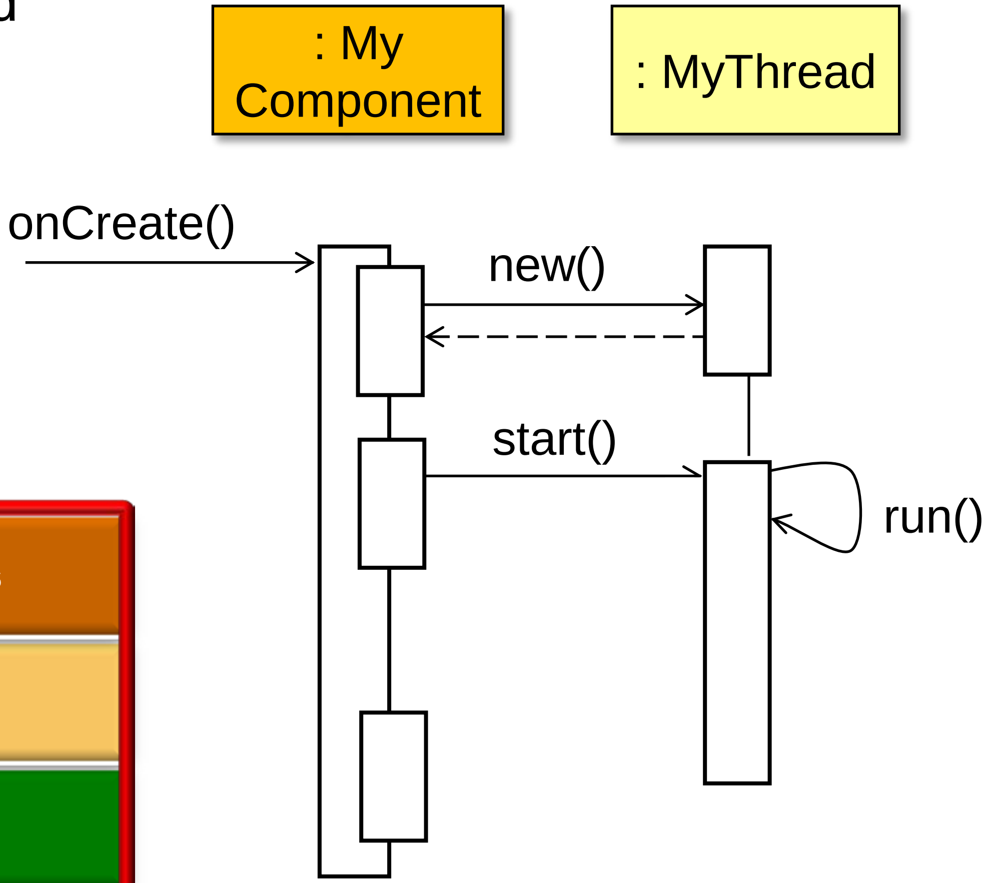
```
public class GCDRunnable extends Random implements Runnable {
  final private String mThreadType;

  public void setThreadType(String threadType) {
      mThreadType = threadType;
  }

  public void run() {
    final String threadString =
      " with " + mThreadType + " thread id " +
      Thread.currentThread();
    ...
```

Running
Java Threads

Running Java Threads

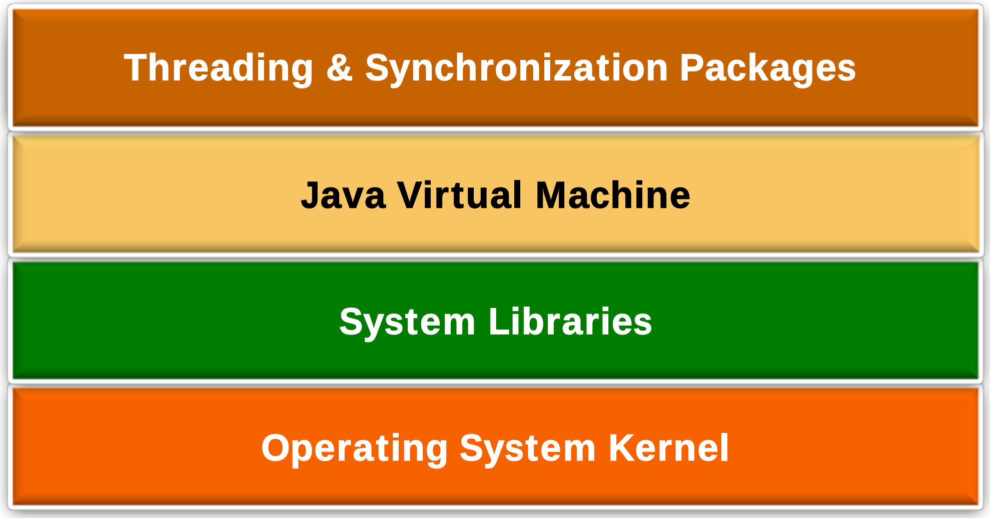
- After start() creates the Thread's resources, the JVM calls its run() hook method

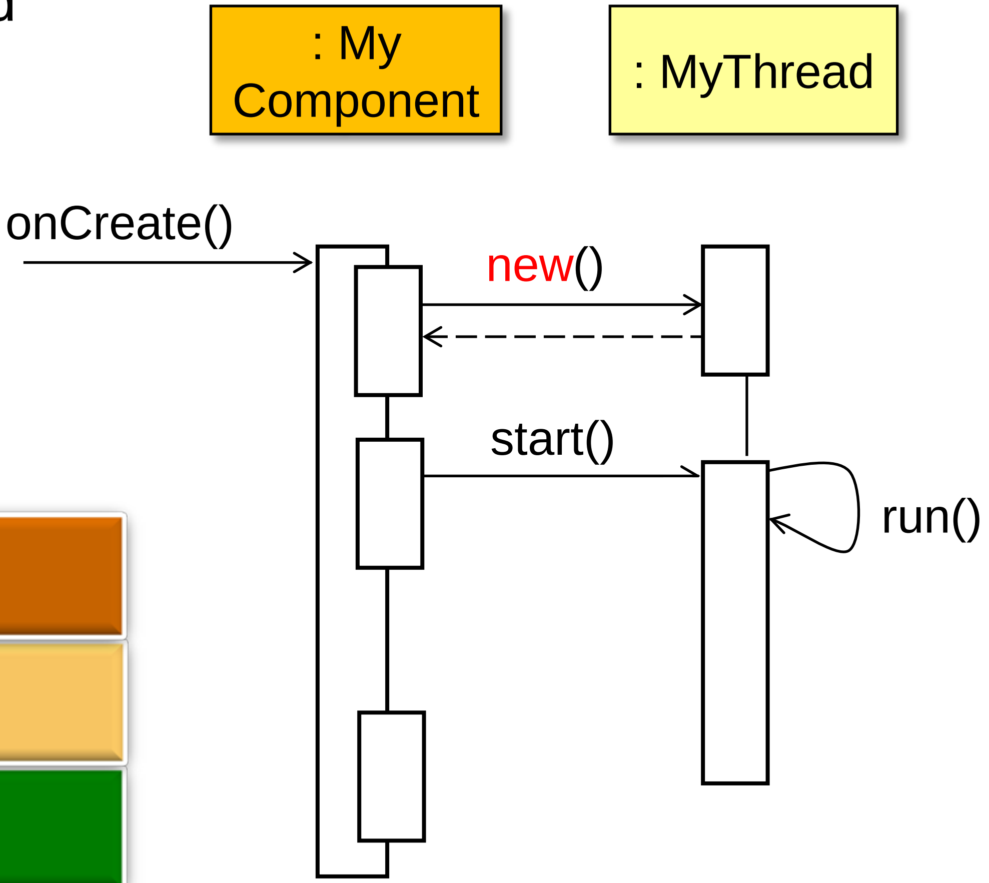




Running Java Threads

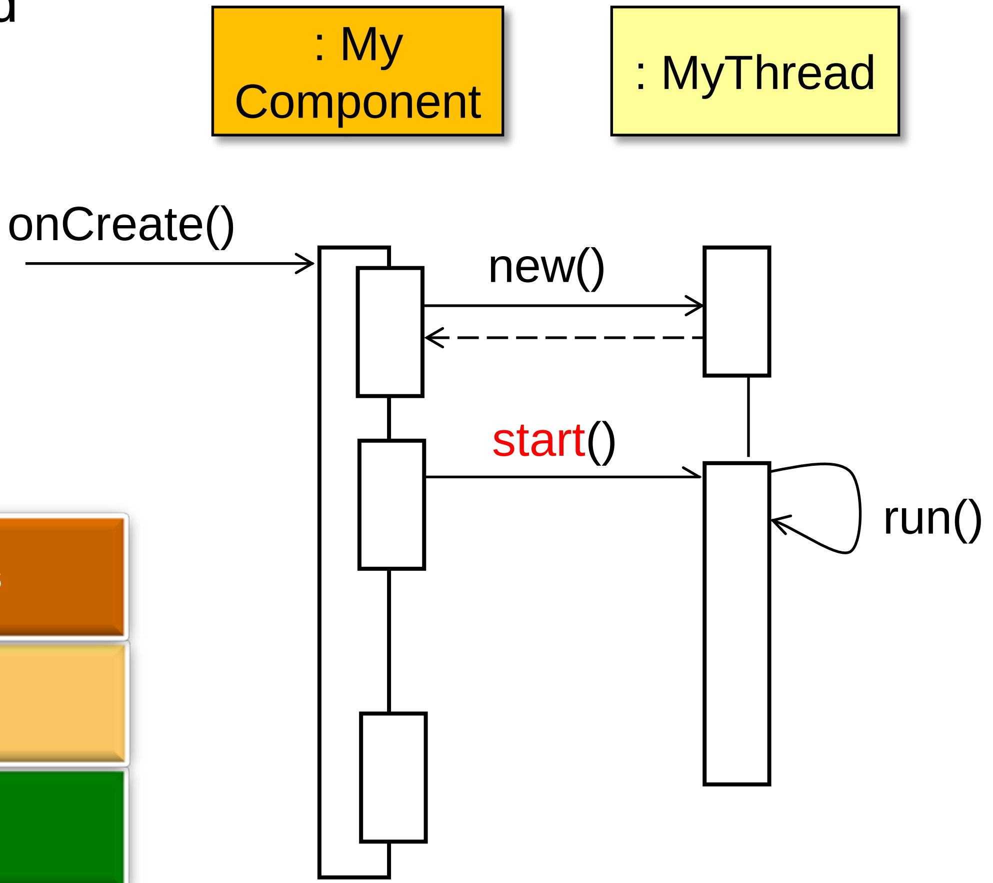
- After start() creates the Thread's resources, the JVM calls its run() hook method

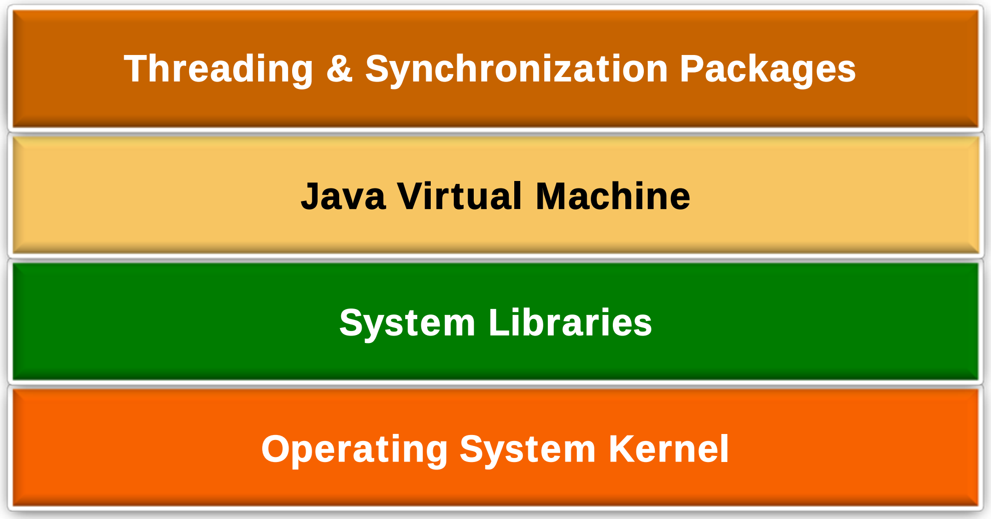




Running Java Threads

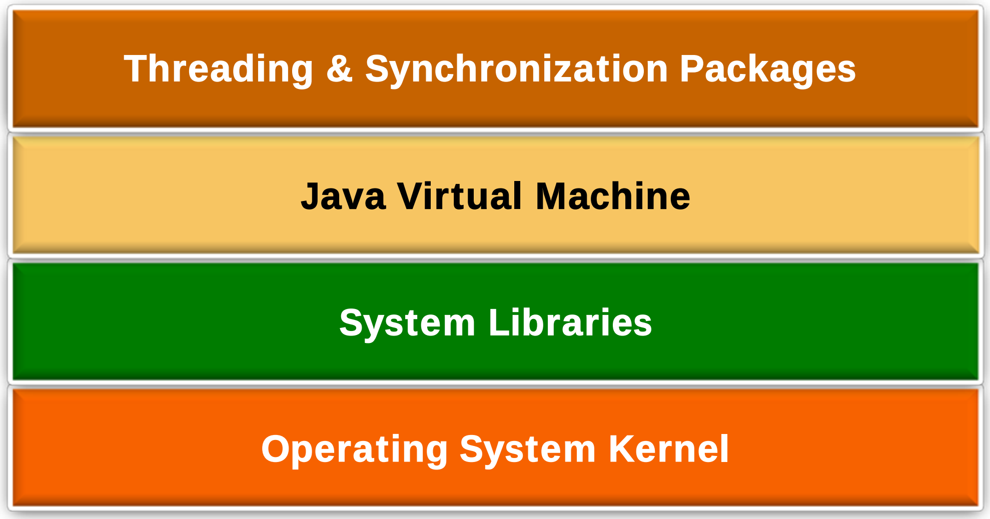
- After start() creates the Thread's resources, the JVM calls its run() hook method

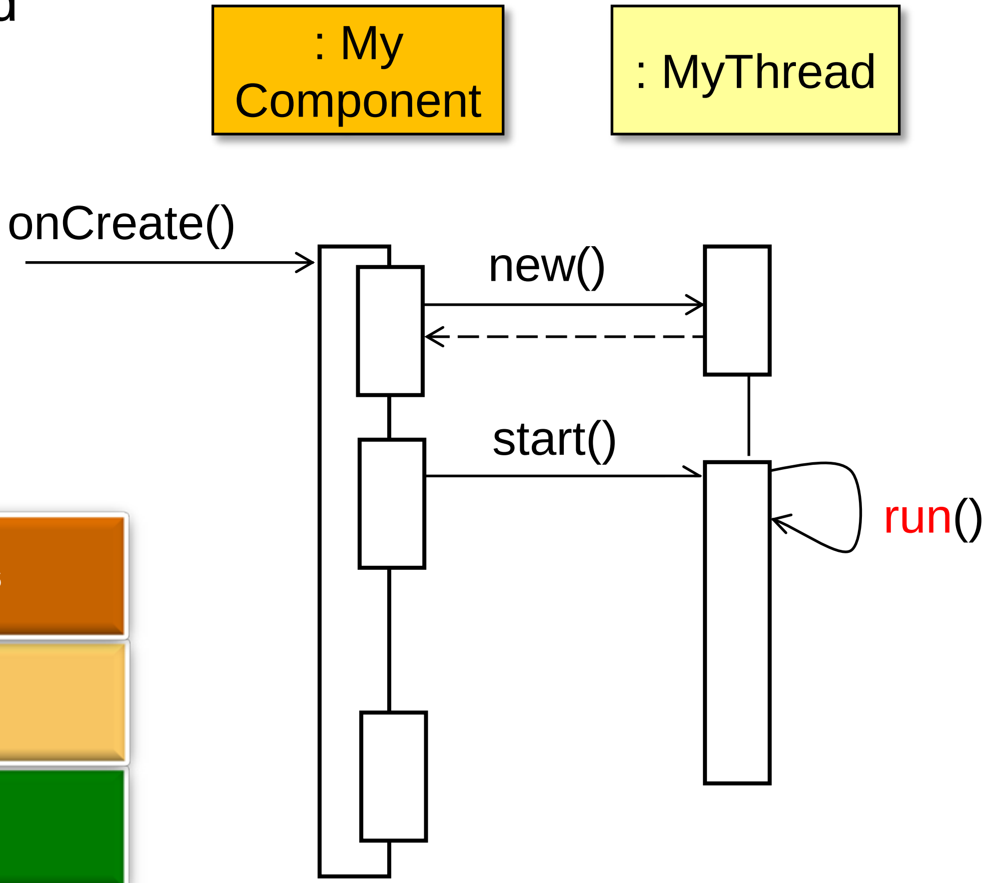




Running Java Threads

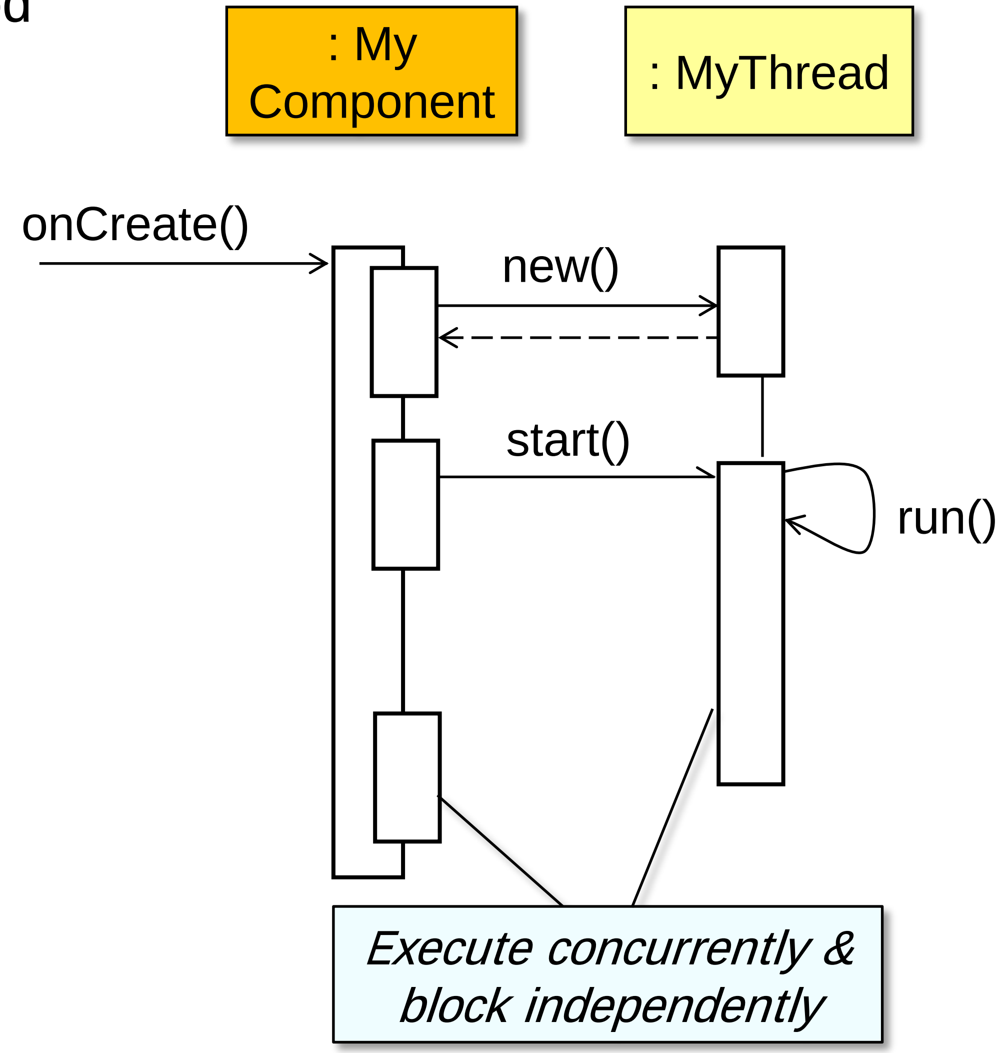
- After start() creates the Thread's resources, the JVM calls its run() hook method





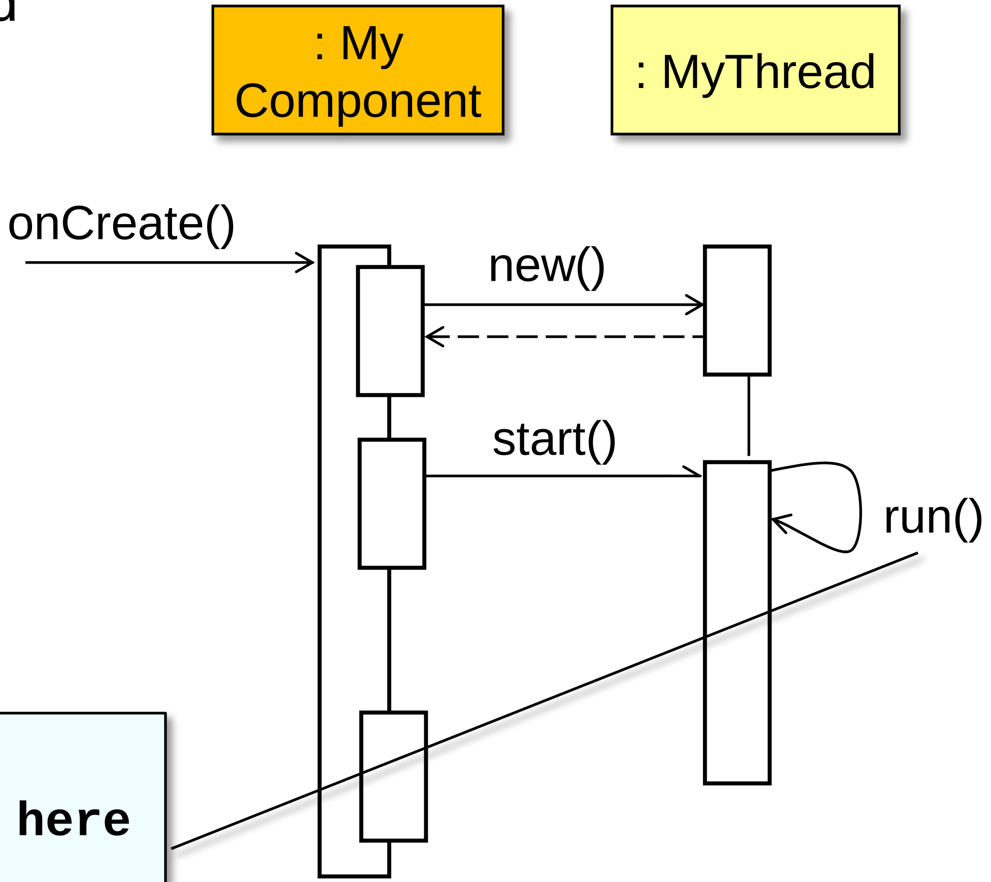
Running Java Threads

- After start() creates the Thread's resources, the JVM calls its run() hook method



Running Java Threads

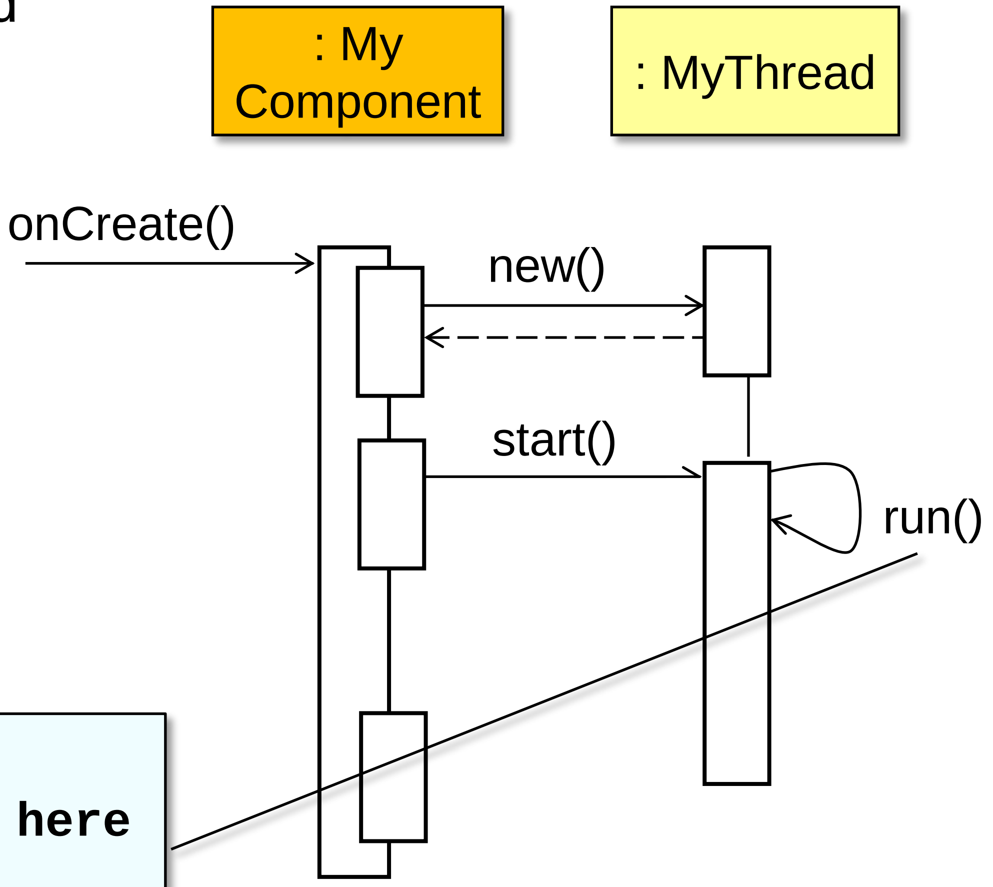
- After start() creates the Thread's resources, the JVM calls its run() hook method
  - Generally any code can run in a Thread



```
public void run(){
  // code to run goes here
}
```

Running Java Threads

- After start() creates the Thread's resources, the JVM calls its run() hook method
  - Generally any code can run in a Thread
  - Windowing toolkits often restrict which thread can access GUI components

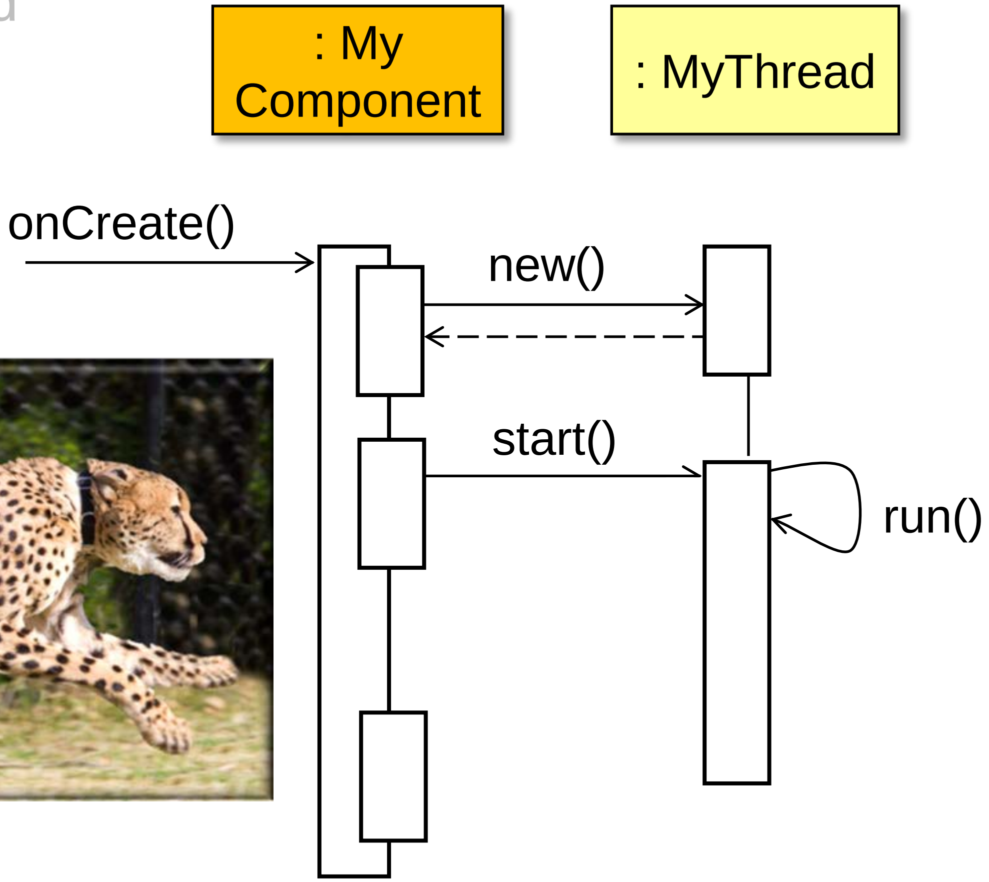


```
public void run(){
  // code to run goes here
}
```

See developer.android.com/training/multiple-threads/communicate-ui.html

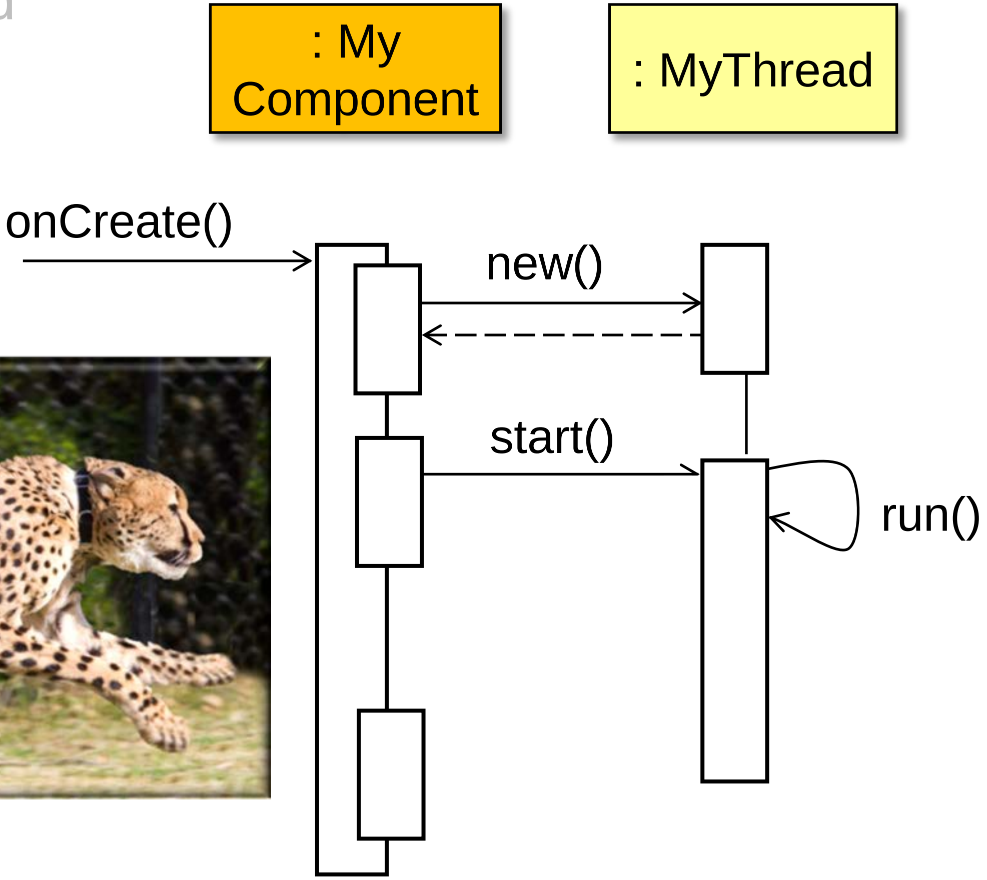
Running Java Threads

- After start() creates the Thread's resources, the JVM calls its run() hook method
- A Thread can live as long as its run() method hasn't returned



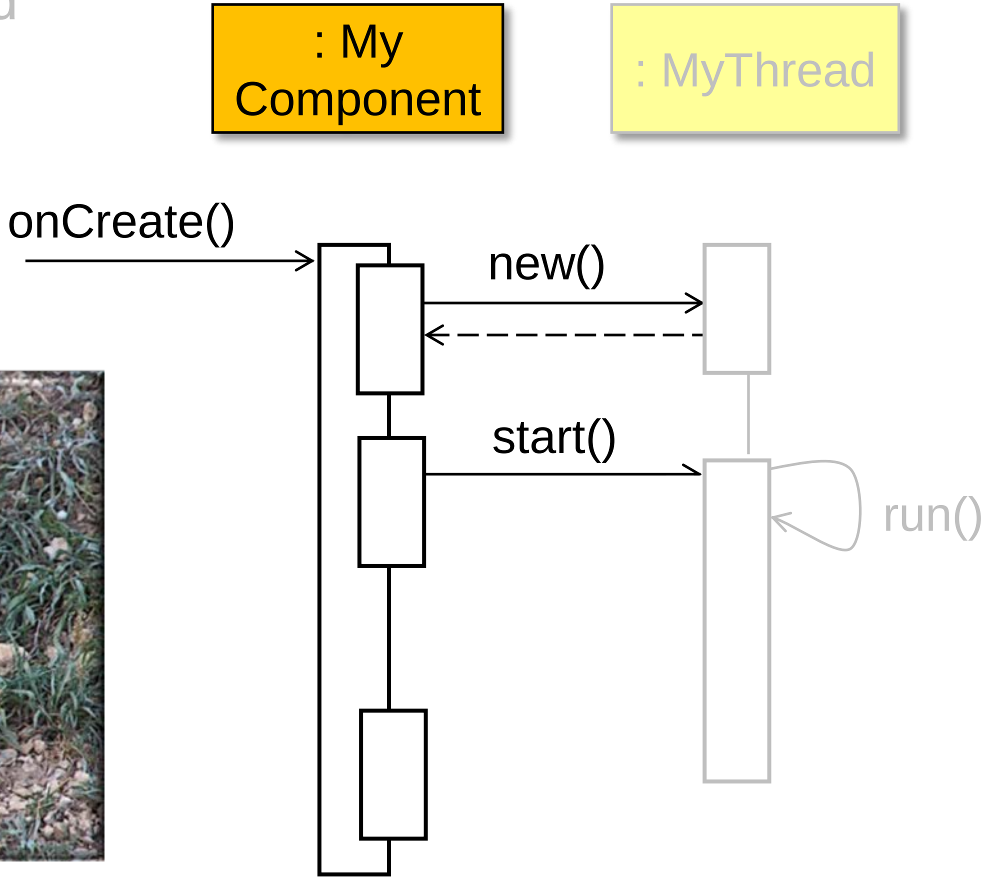
Running Java Threads

- After start() creates the Thread's resources, the JVM calls its run() hook method
- A Thread can live as long as its run() method hasn't returned
  - i.e., either normally or via an exception



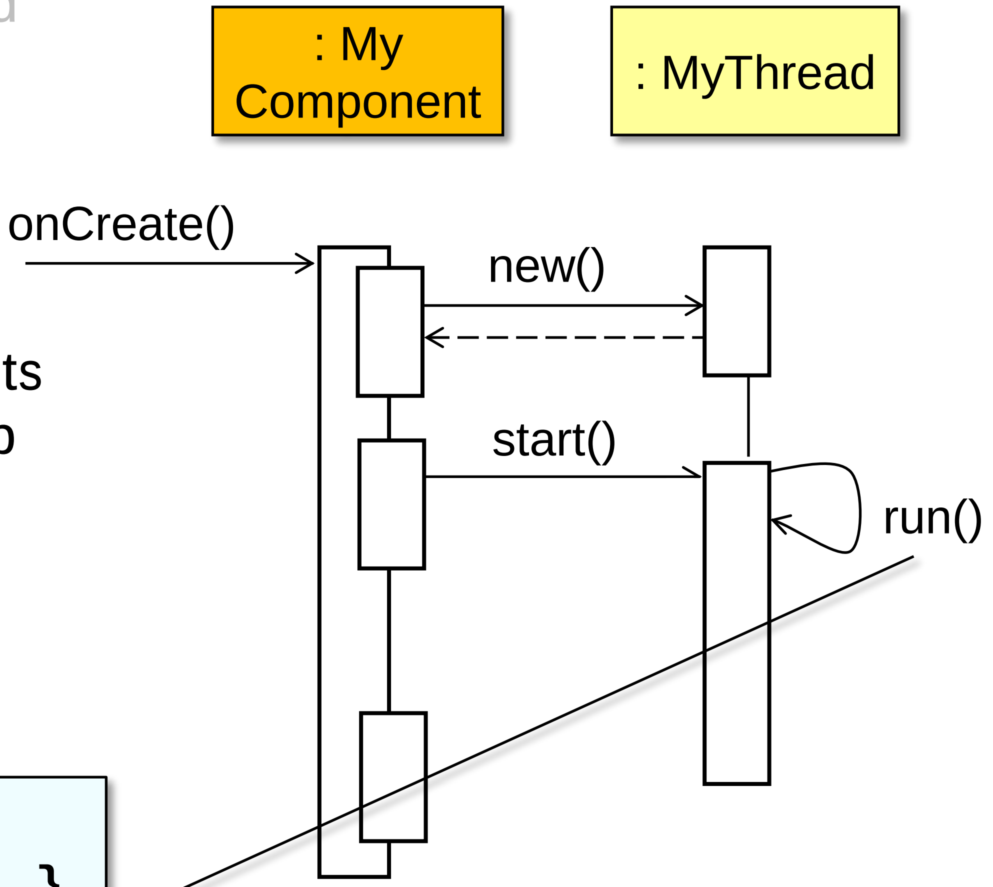
Running Java Threads

- After start() creates the Thread's resources, the JVM calls its run() hook method
- A Thread can live as long as its run() method hasn't returned
- The scheduler can suspend & resume a Thread many times



Running Java Threads

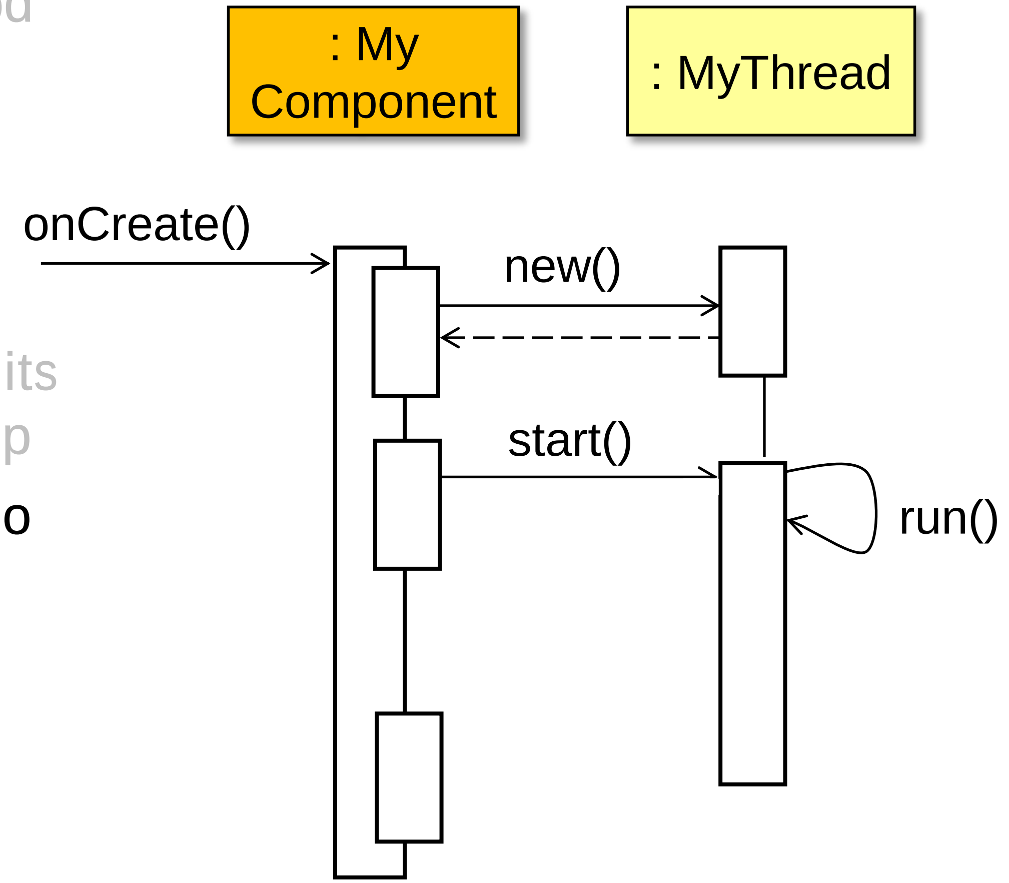
- After start() creates the Thread's resources, the JVM calls its run() hook method
- A Thread can live as long as its run() method hasn't returned
- The scheduler can suspend & resume a Thread many times
- For a Thread to execute "forever," its run() method needs an infinite loop



```
public void run(){
  while (true) { ... }
}
```

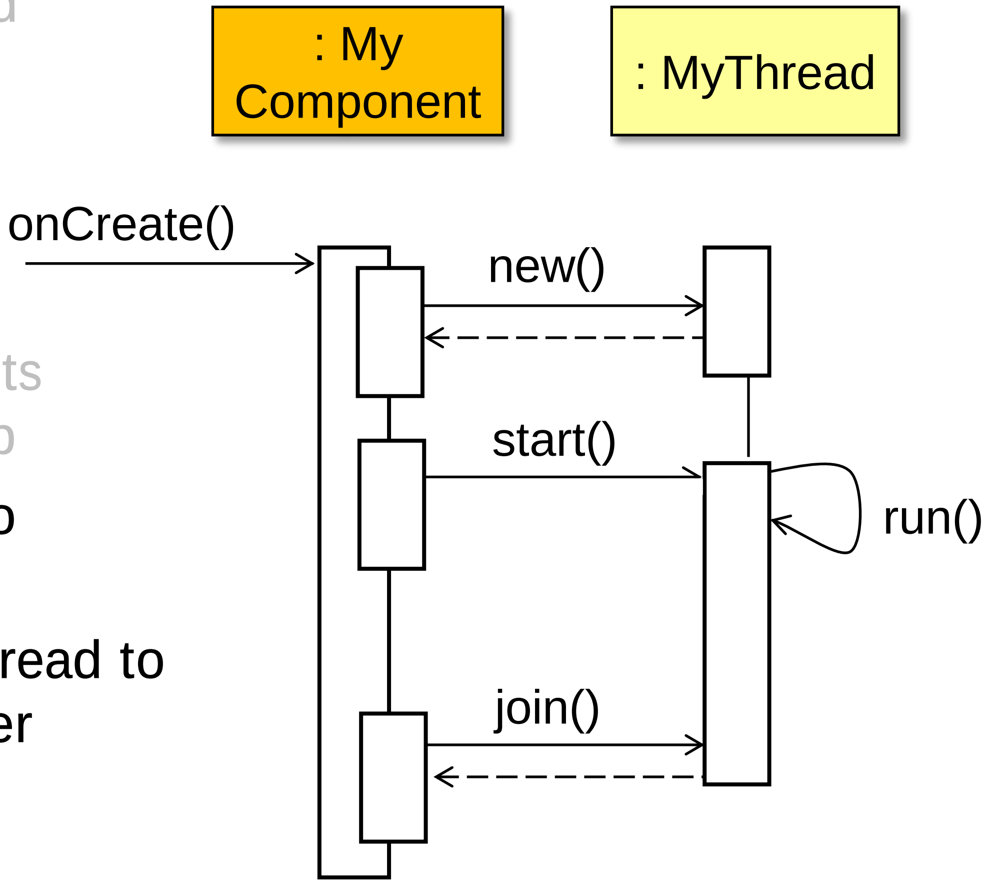
Running Java Threads

- After start() creates the Thread's resources, the JVM calls its run() hook method
- A Thread can live as long as its run() method hasn't returned
- The scheduler can suspend & resume a Thread many times
- For a Thread to execute "forever," its run() method needs an infinite loop
- After run() returns the Thread is no longer alive



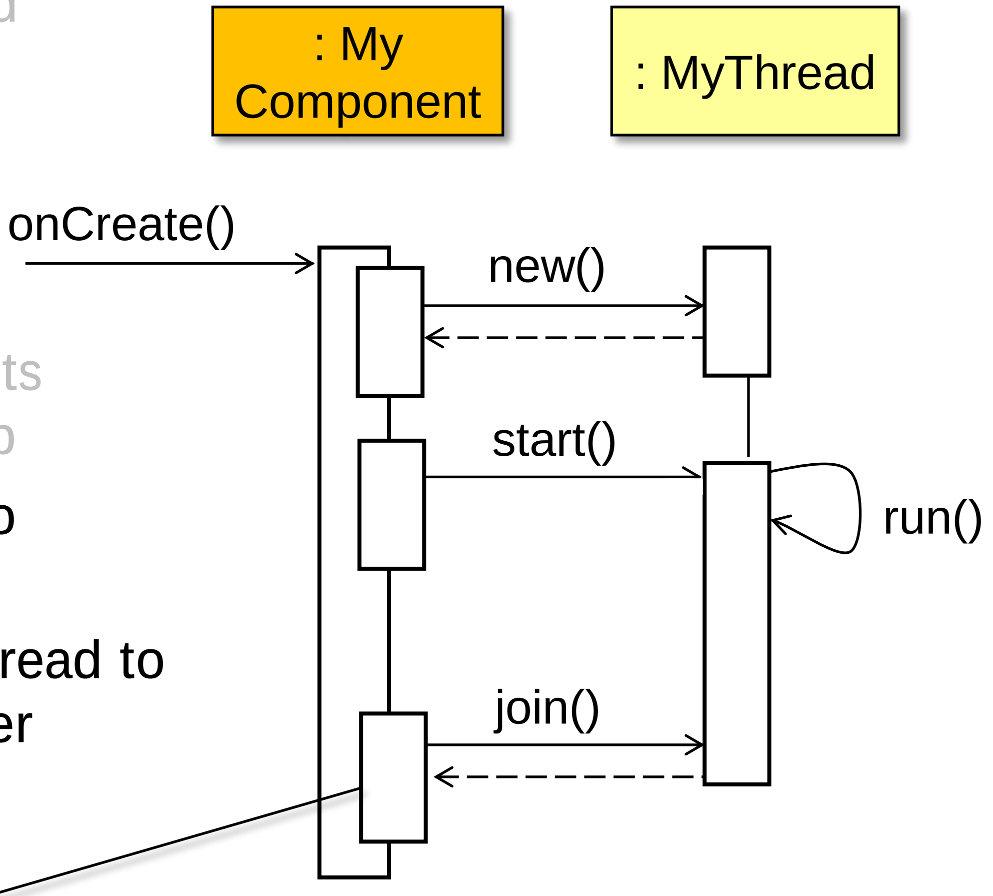
Running Java Threads

- After start() creates the Thread's resources, the JVM calls its run() hook method
- A Thread can live as long as its run() method hasn't returned
- The scheduler can suspend & resume a Thread many times
- For a Thread to execute "forever," its run() method needs an infinite loop
- After run() returns the Thread is no longer alive
  - The join() method allows one Thread to wait for the completion of another



Running Java Threads

- After start() creates the Thread's resources, the JVM calls its run() hook method
- A Thread can live as long as its run() method hasn't returned
- The scheduler can suspend & resume a Thread many times
- For a Thread to execute "forever," its run() method needs an infinite loop
- After run() returns the Thread is no longer alive
  - The join() method allows one Thread to wait for the completion of another

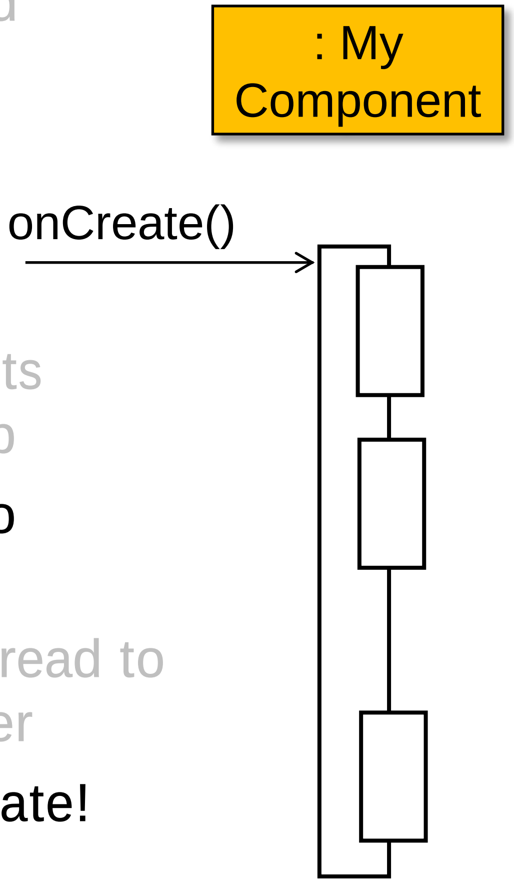


Simple form of "barrier synchronization"

See upcoming parts on "Java Barrier Synchronizers"

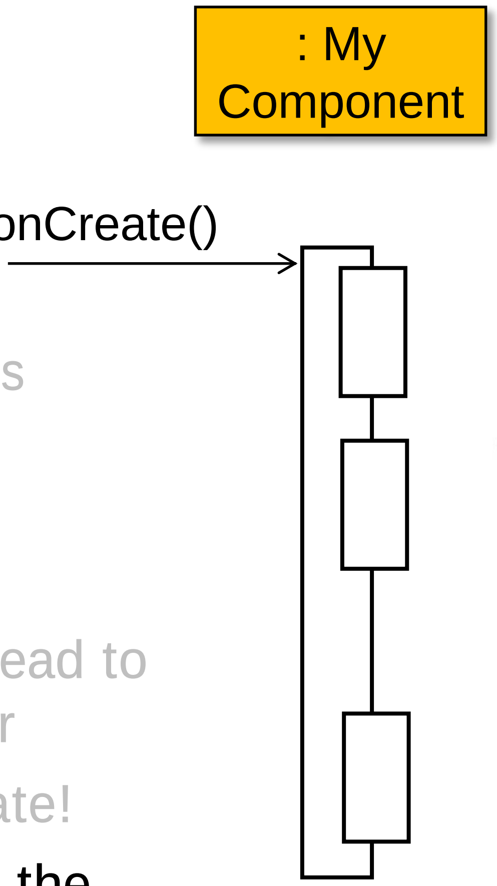
Running Java Threads

- After start() creates the Thread's resources, the JVM calls its run() hook method
- A Thread can live as long as its run() method hasn't returned
- The scheduler can suspend & resume a Thread many times
- For a Thread to execute "forever," its run() method needs an infinite loop
- After run() returns the Thread is no longer alive
    - The join() method allows one Thread to wait for the completion of another
    - Or the Thread can simple evaporate!



Running Java Threads

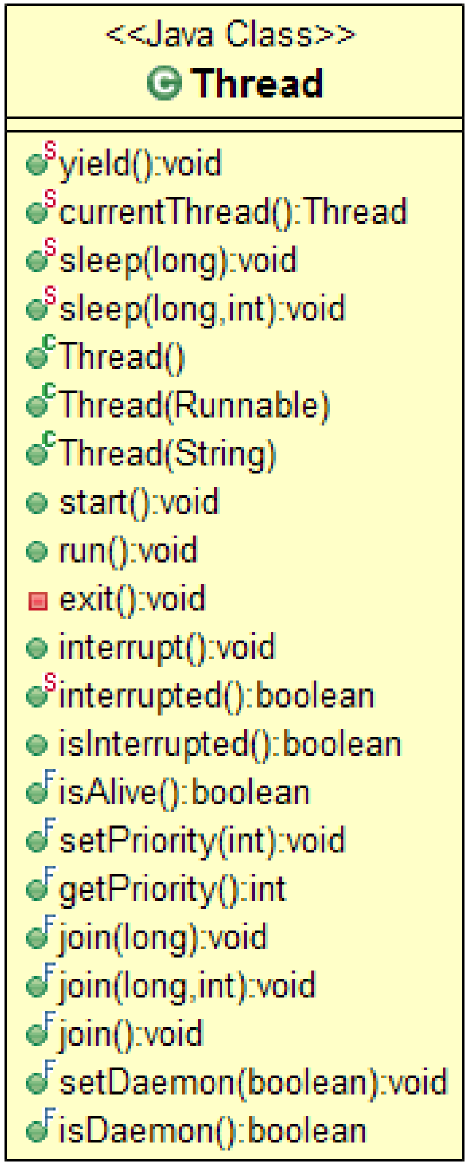
- After start() creates the Thread's resources, the JVM calls its run() hook method
- A Thread can live as long as its run() method hasn't returned
- The scheduler can suspend & resume a Thread many times
- For a Thread to execute "forever," its run() method needs an infinite loop
- After run() returns the Thread is no longer alive
  - The join() method allows one Thread to wait for the completion of another
  - Or the Thread can simple evaporate!
  - The Java virtual machine recycles the resources associated with the Thread



Some Common Java Thread Methods

Some Common Java Thread Methods

- There are a number of commonly used methods in the Java Thread class



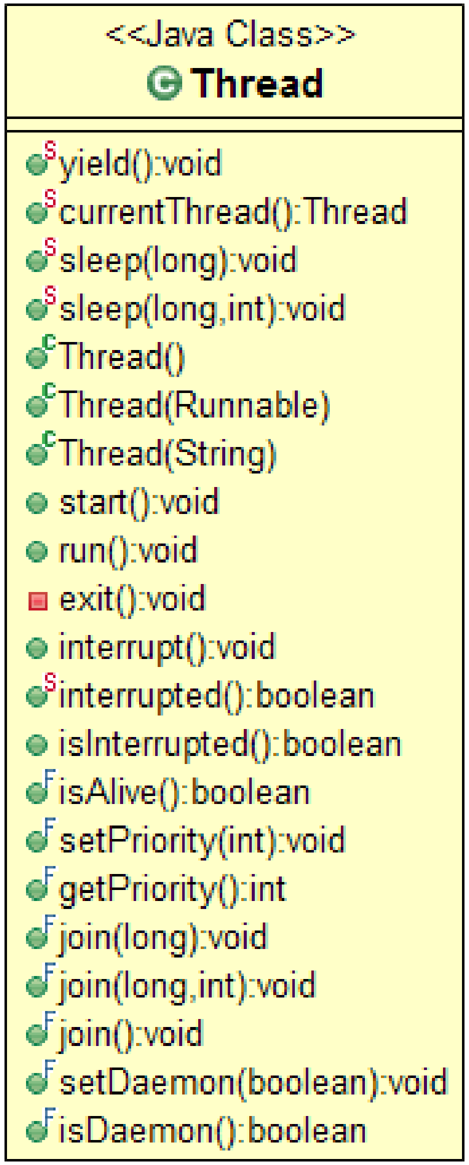
<<Java Class>>
Thread

- 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/7/docs/api/java/lang/Thread.html

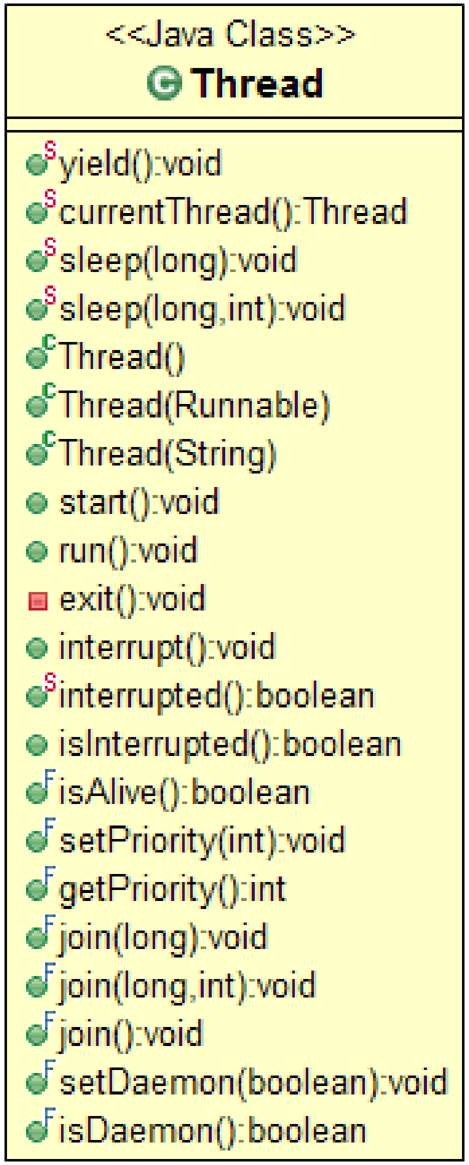
Some Common Java Thread Methods

- There are a number of commonly used methods in the Java Thread class, e.g.,
  - **`void setDaemon()`**
    - Marks Thread as a Daemon



Some Common Java Thread Methods

- There are a number of commonly used methods in the Java Thread class, e.g.,
  - `void setDaemon()`
  - **`void start()`**
    - Initiates Thread execution

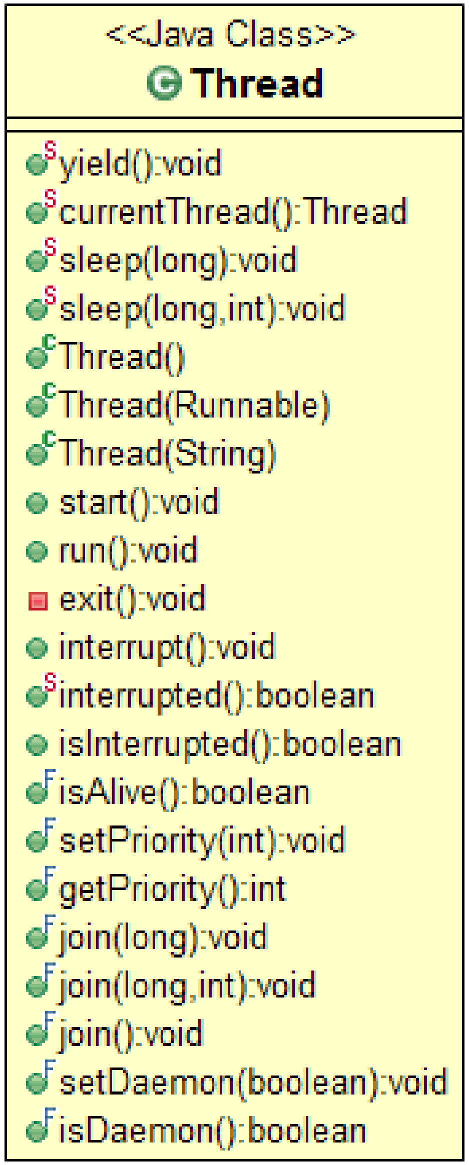


<<Java Class>>
Thread

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

Some Common Java Thread Methods

- There are a number of commonly used methods in the Java Thread class, e.g.,
  - `void setDaemon()`
  - `void start()`
  - **`abstract void run()`**
    - Hook method for user code

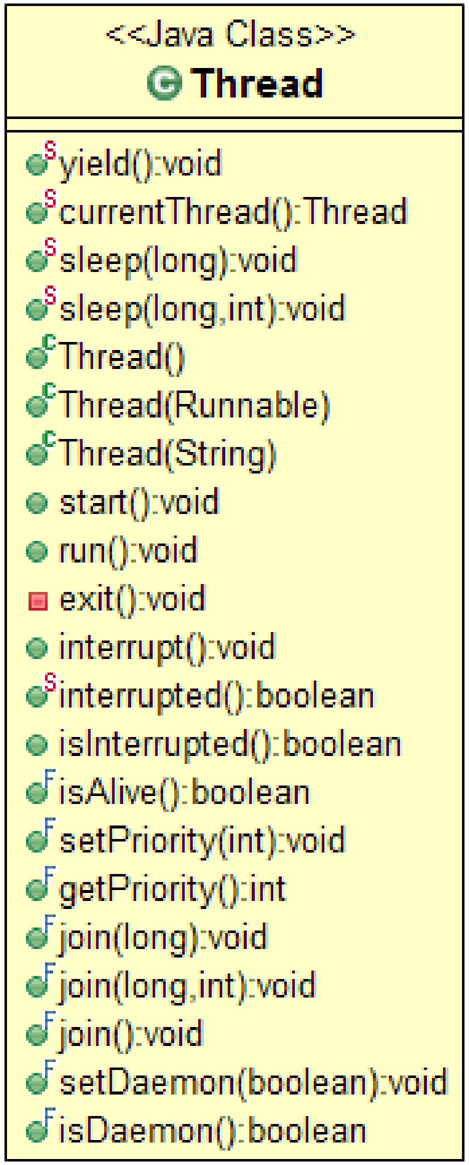


<<Java Class>>
Thread

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

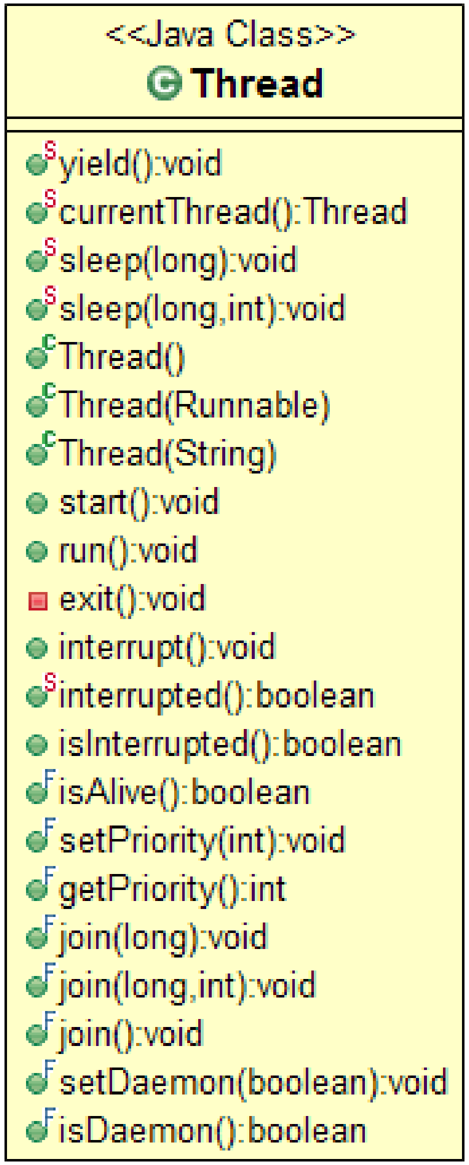
Some Common Java Thread Methods

- There are a number of commonly used methods in the Java Thread class, e.g.,
  - `void setDaemon()`
  - `void start()`
  - `abstract void run()`
  - **`void join()`**
    - Waits for a Thread to finish



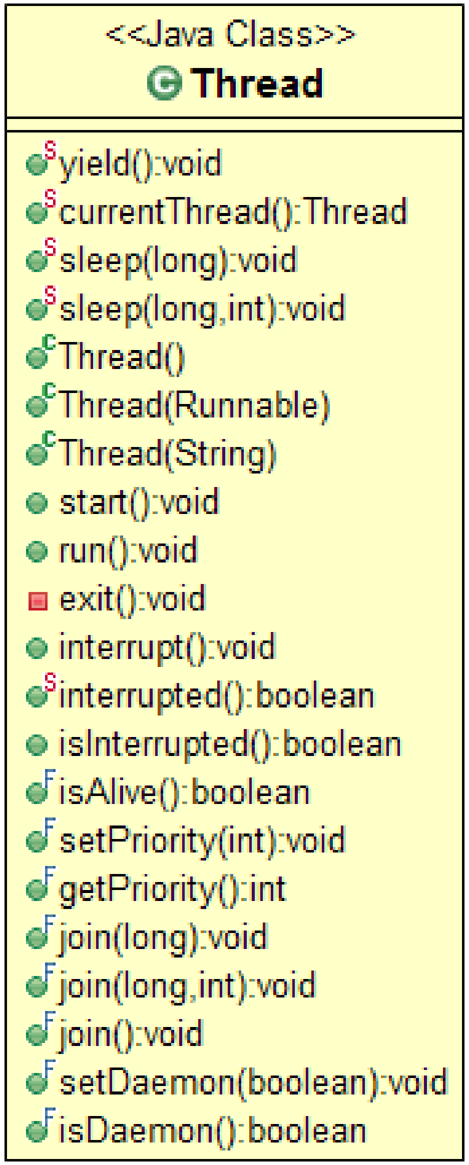
Some Common Java Thread Methods

- There are a number of commonly used methods in the Java Thread class, e.g.,
  - `void setDaemon()`
  - `void start()`
  - `abstract void run()`
  - `void join()`
  - **`void sleep(long time)`**
    - Sleeps for given time in ms



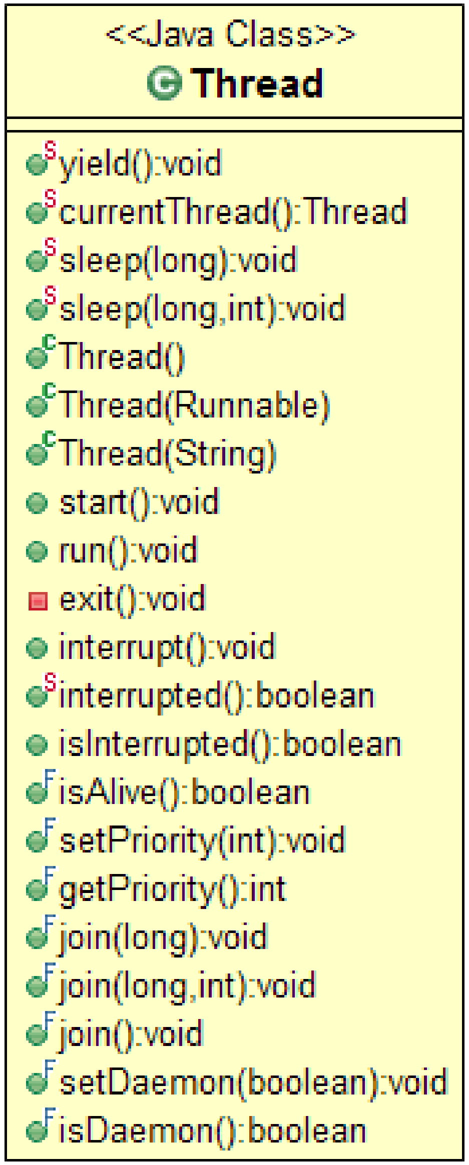
Some Common Java Thread Methods

- There are a number of commonly used methods in the Java Thread class, e.g.,
  - `void setDaemon()`
  - `void start()`
  - `abstract void run()`
  - `void join()`
  - `void sleep(long time)`
  - **`Thread currentThread()`**
    - Object for current Thread



Some Common Java Thread Methods

- There are a number of commonly used methods in the Java Thread class, e.g.,
  - `void setDaemon()`
  - `void start()`
  - `abstract void run()`
  - `void join()`
  - `void sleep(long time)`
  - `Thread currentThread()`
  - **`void interrupt()`**
    - Post an interrupt request to a Thread

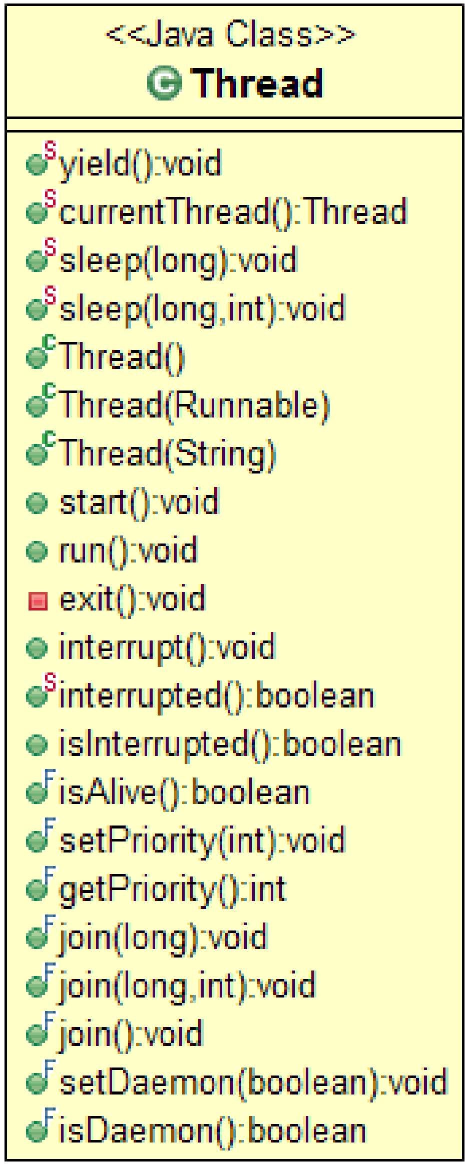


```
<<Java Class>>
Thread

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 upcoming part on "Managing the Java Thread Lifecycle"

Some Common Java Thread Methods

- There are a number of commonly used methods in the Java Thread class, e.g.,
  - `void setDaemon()`
  - `void start()`
  - `abstract void run()`
  - `void join()`
  - `void sleep(long time)`
  - `Thread currentThread()`
  - `void interrupt()`
  - **`boolean isInterrupted()`**
    - Tests whether a thread has been interrupted

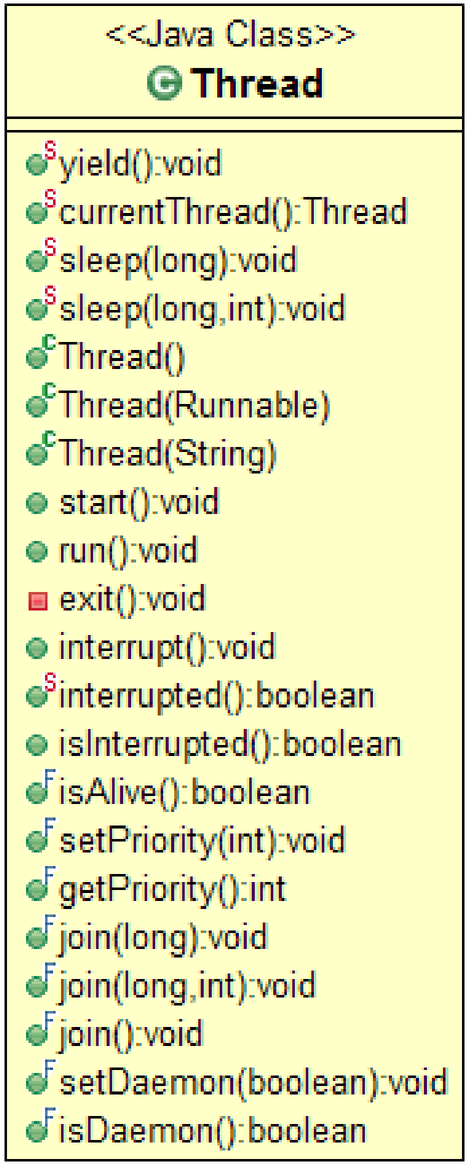

<<Java Class>>
Thread

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

isInterrupted() can be called multiple times w/out affecting the *interrupted status*

Some Common Java Thread Methods

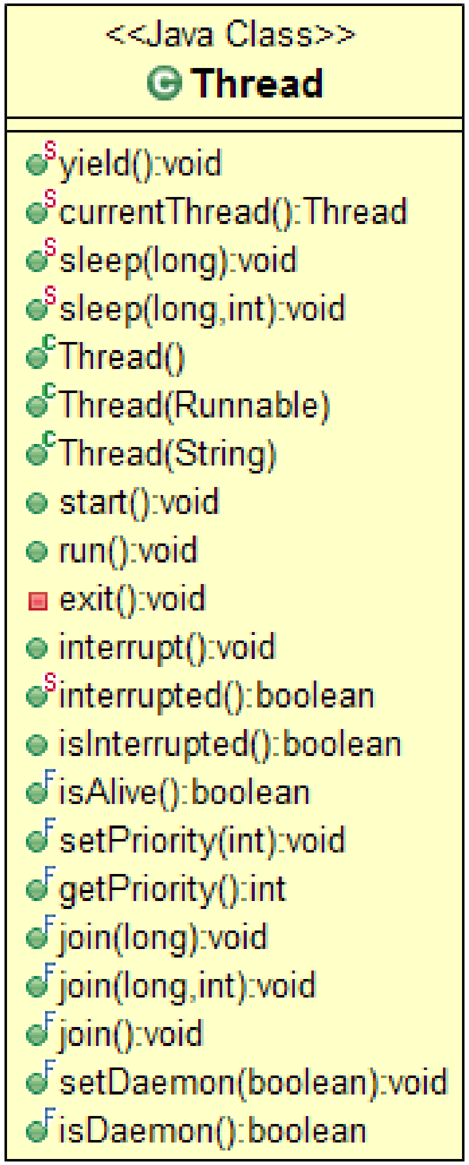
- There are a number of commonly used methods in the Java Thread class, e.g.,
  - `void setDaemon()`
  - `void start()`
  - `abstract void run()`
  - `void join()`
  - `void sleep(long time)`
  - `Thread currentThread()`
  - `void interrupt()`
  - `boolean isInterrupted()`
  - **`boolean interrupted()`**
    - Tests whether current thread has been interrupted



interrupted() clears the *interrupted status* the first time it's called

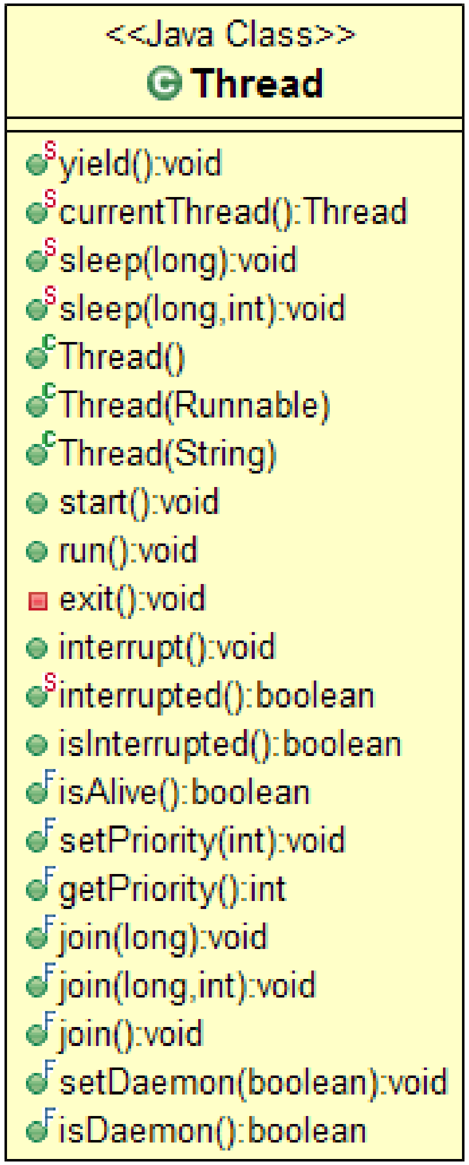
Some Common Java Thread Methods

- There are a number of commonly used methods in the Java Thread class, e.g.,
  - `void setDaemon()`
  - `void start()`
  - `abstract void run()`
  - `void join()`
  - `void sleep(long time)`
  - `Thread currentThread()`
  - `void interrupt()`
  - `boolean isInterrupted()`
  - `boolean interrupted()`
  - **`void setPriority(int newPriority)` & `int getPriority()`**
    - Set & get the priority of a Thread



Some Common Java Thread Methods

- There are a number of commonly used methods in the Java Thread class
- These methods establish various "Happens-Before" orderings

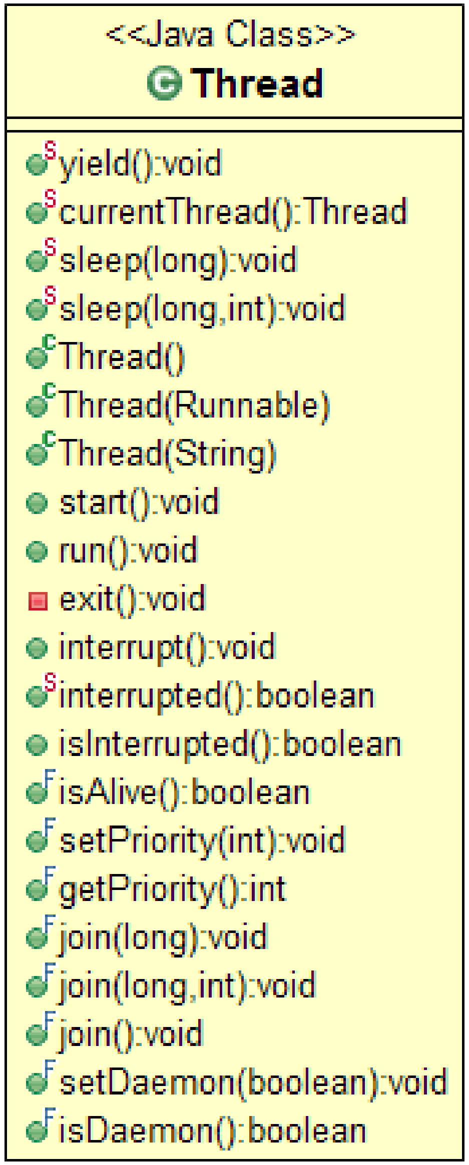

<<Java Class>>
Thread

```
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 en.wikipedia.org/wiki/Happened-before

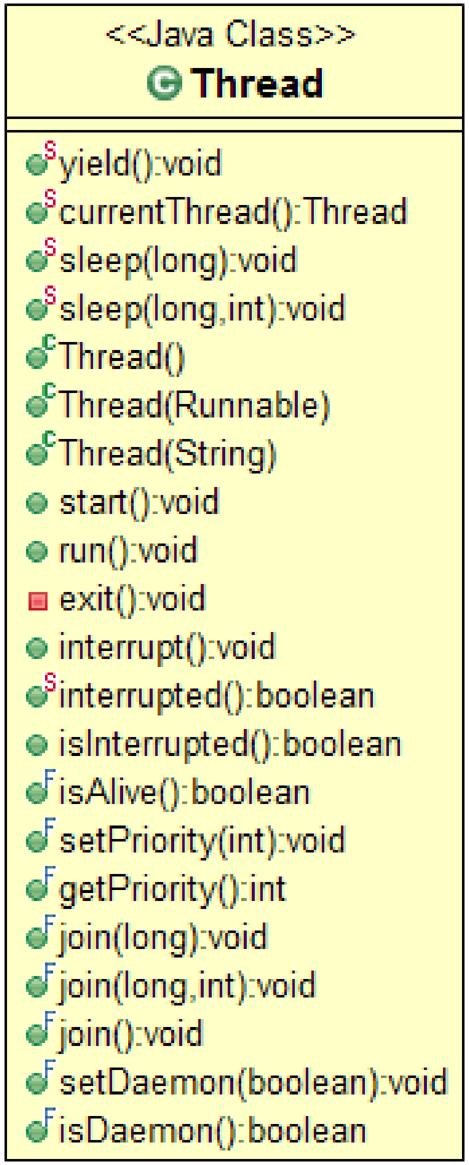
Some Common Java Thread Methods

- There are a number of commonly used methods in the Java Thread class
- These methods establish various “Happens-Before” orderings, e.g.,
  - Starting a thread happens-before the run method of the thread



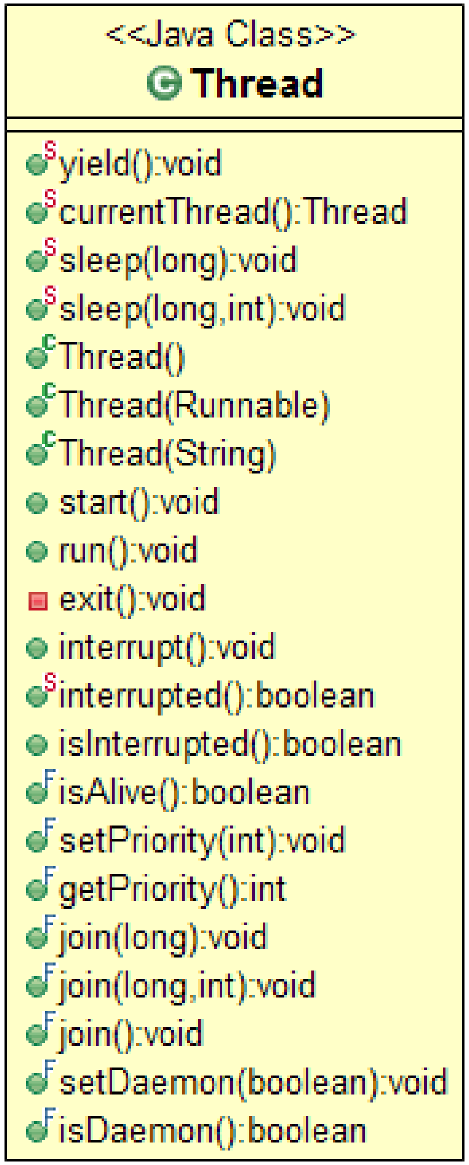
Some Common Java Thread Methods

- There are a number of commonly used methods in the Java Thread class
- These methods establish various "Happens-Before" orderings, e.g.,
  - Starting a thread happens-before the run method of the thread
  - The termination of a thread happens-before a join with the terminated thread



Some Common Java Thread Methods

- There are a number of commonly used methods in the Java Thread class
- These methods establish various "Happens-Before" orderings, e.g.,
  - Starting a thread happens-before the run method of the thread
  - The termination of a thread happens-before a join with the terminated thread
  - Many java.util.concurrent methods also establish happen-before orderings
    - e.g., placing an object into a concurrent collection happens-before the access or removal of the element from the collection

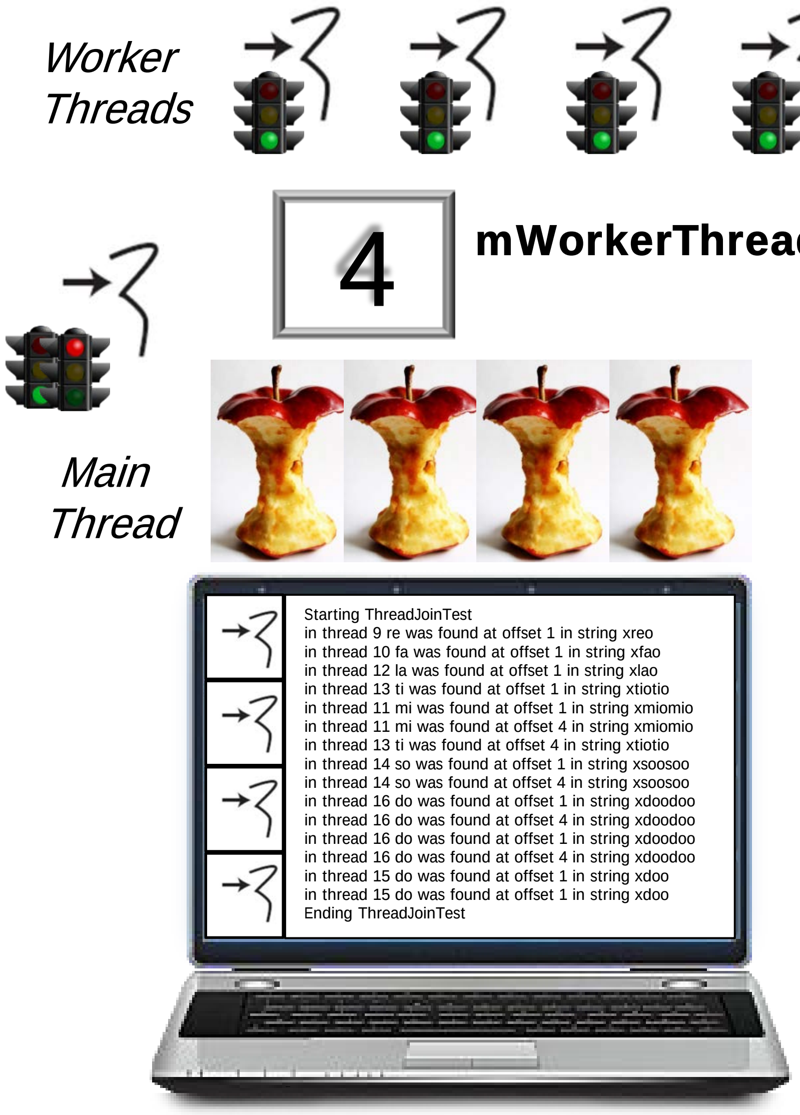


```
<<Java Class>>
Thread

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
```

Example of Starting & Joining with Java Threads

Example of Starting & Joining with Java Threads

- Demonstrates the use of various Thread methods to implement an "embarrassingly parallel" application

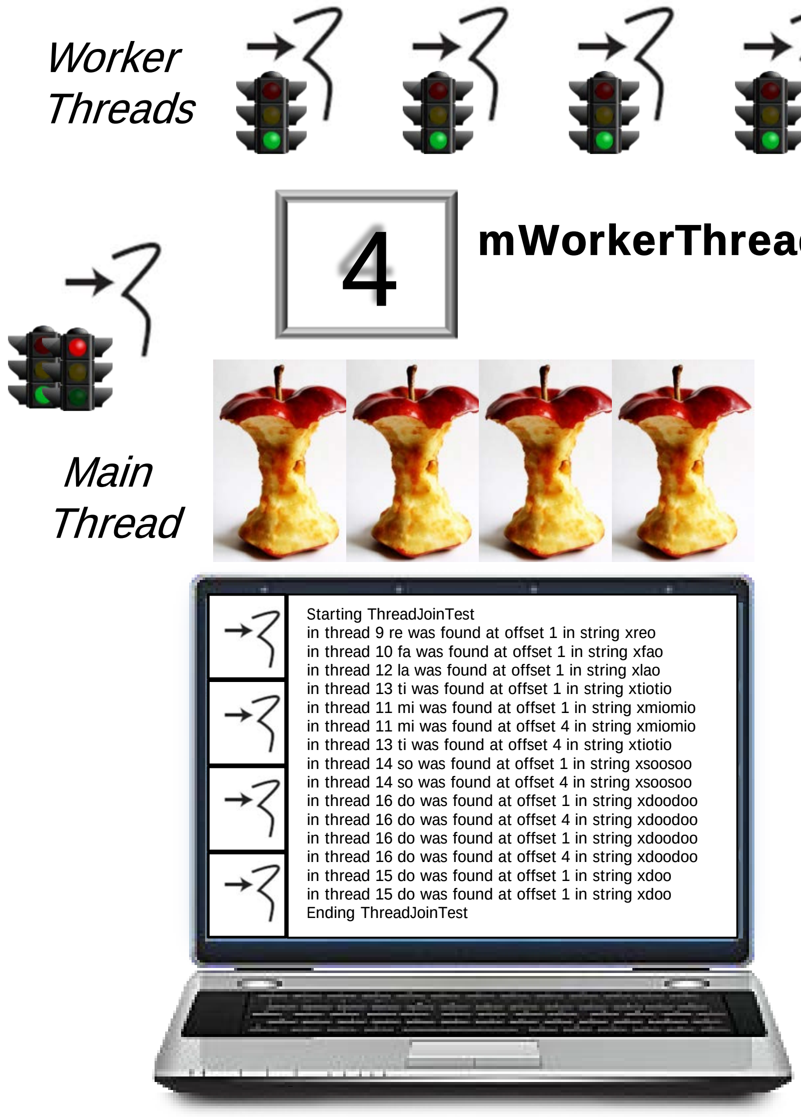


```
Starting ThreadJoinTest
in thread 9 re was found at offset 1 in string xreo
in thread 10 fa was found at offset 1 in string xfao
in thread 12 la was found at offset 1 in string xlao
in thread 13 ti was found at offset 1 in string xtiotio
in thread 11 mi was found at offset 1 in string xmiomio
in thread 11 mi was found at offset 4 in string xmiomio
in thread 13 ti was found at offset 4 in string xtiotio
in thread 14 so was found at offset 1 in string xsoosoo
in thread 14 so was found at offset 4 in string xsoosoo
in thread 16 do was found at offset 1 in string xdoodoo
in thread 16 do was found at offset 4 in string xdoodoo
in thread 16 do was found at offset 1 in string xdoodoo
in thread 16 do was found at offset 4 in string xdoodoo
in thread 15 do was found at offset 1 in string xdoo
in thread 15 do was found at offset 1 in string xdoo
Ending ThreadJoinTest
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/ThreadJoinTest

Example of Starting & Joining with Java Threads

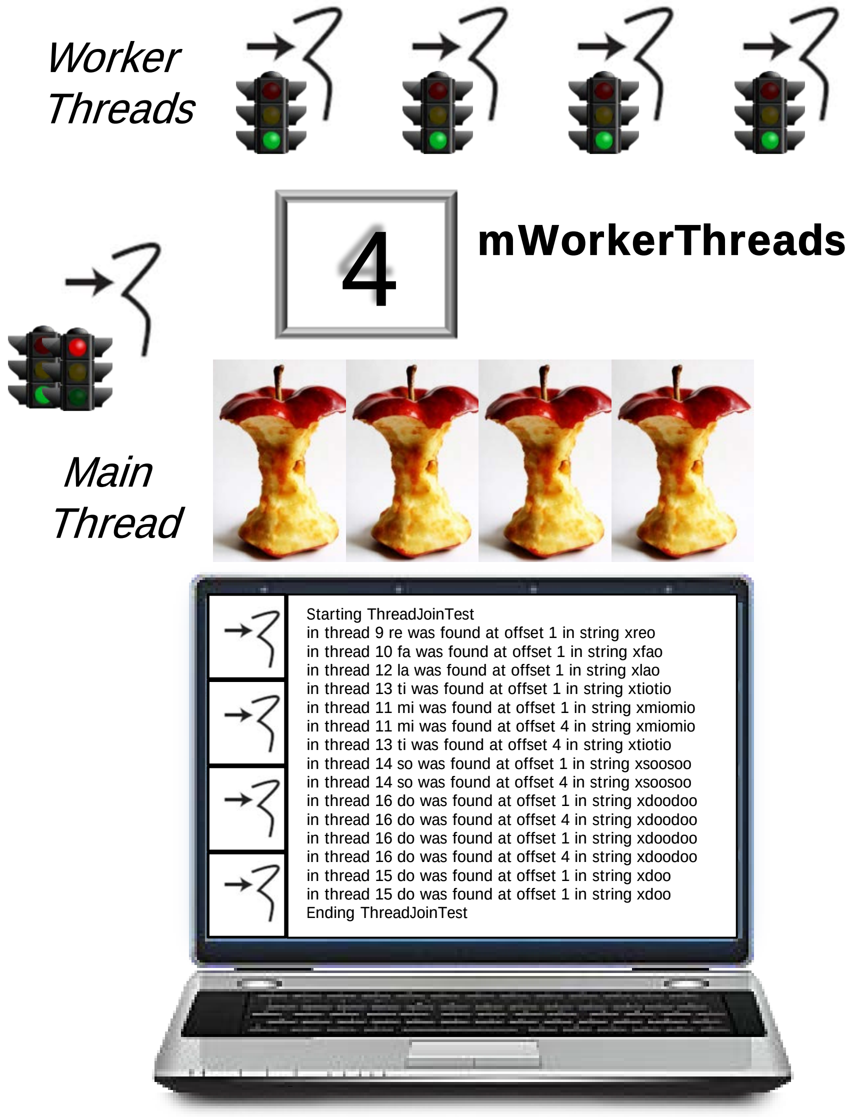
- Demonstrates the use of various Thread methods to implement an "embarrassingly parallel" application
  - Concurrently searches for words in a List of Strings



```
Starting ThreadJoinTest
in thread 9 re was found at offset 1 in string xreo
in thread 10 fa was found at offset 1 in string xfao
in thread 12 la was found at offset 1 in string xlao
in thread 13 ti was found at offset 1 in string xtiotio
in thread 11 mi was found at offset 1 in string xmiomio
in thread 11 mi was found at offset 4 in string xmiomio
in thread 13 ti was found at offset 4 in string xtiotio
in thread 14 so was found at offset 1 in string xsoosoo
in thread 14 so was found at offset 4 in string xsoosoo
in thread 16 do was found at offset 1 in string xdoodoo
in thread 16 do was found at offset 4 in string xdoodoo
in thread 16 do was found at offset 1 in string xdoodoo
in thread 16 do was found at offset 4 in string xdoodoo
in thread 15 do was found at offset 1 in string xdoo
in thread 15 do was found at offset 1 in string xdoo
Ending ThreadJoinTest
```

Example of Starting & Joining with Java Threads

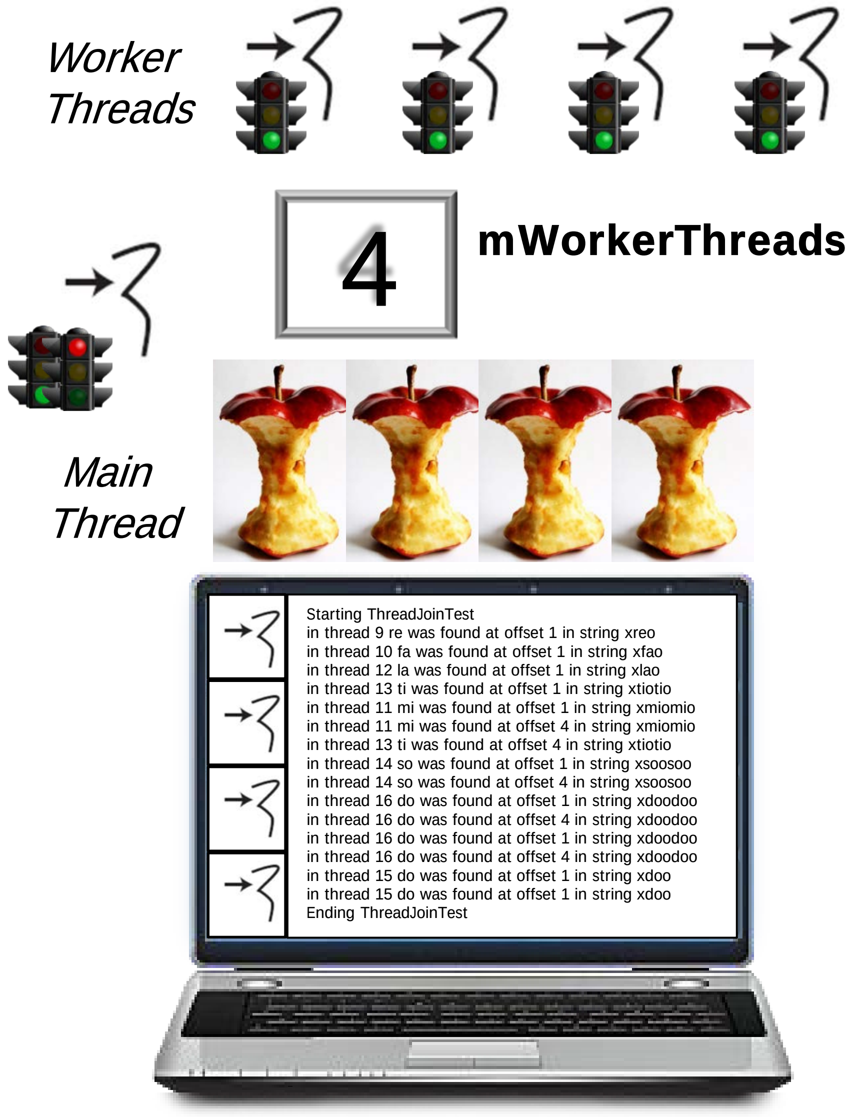
- Demonstrates the use of various Thread methods to implement an "embarrassingly parallel" application
  - Concurrently searches for words in a List of Strings
  - Calls Thread.start() to spawn a worker Thread for each element in the List of Strings



```
Starting ThreadJoinTest
in thread 9 re was found at offset 1 in string xreo
in thread 10 fa was found at offset 1 in string xfao
in thread 12 la was found at offset 1 in string xlao
in thread 13 ti was found at offset 1 in string xtiotio
in thread 11 mi was found at offset 1 in string xmiomio
in thread 11 mi was found at offset 4 in string xmiomio
in thread 13 ti was found at offset 4 in string xtiotio
in thread 14 so was found at offset 1 in string xsoosoo
in thread 14 so was found at offset 4 in string xsoosoo
in thread 16 do was found at offset 1 in string xdoodoo
in thread 16 do was found at offset 4 in string xdoodoo
in thread 16 do was found at offset 1 in string xdoodoo
in thread 16 do was found at offset 4 in string xdoodoo
in thread 15 do was found at offset 1 in string xdoo
in thread 15 do was found at offset 1 in string xdoo
Ending ThreadJoinTest
```

Example of Starting & Joining with Java Threads

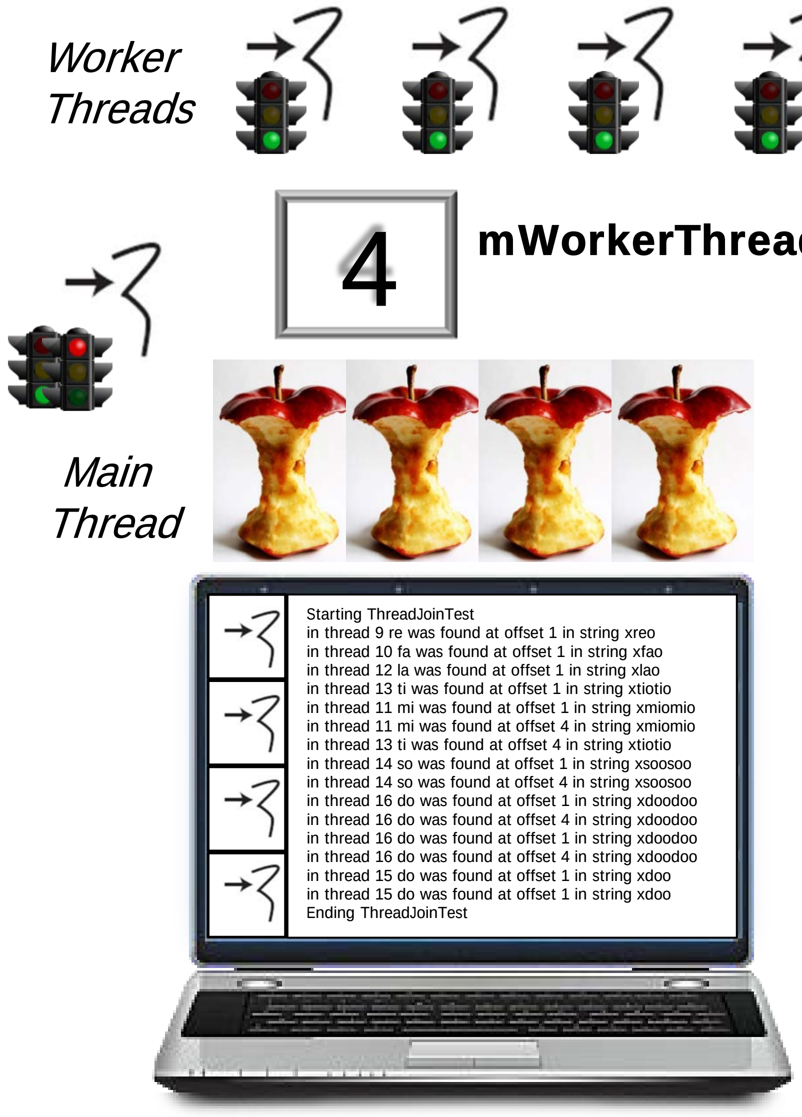
- Demonstrates the use of various Thread methods to implement an "embarrassingly parallel" application
  - Concurrently searches for words in a List of Strings
  - Calls Thread.start() to spawn a worker Thread for each element in the List of Strings
    - Each Thread concurrently searches for words in its associated String



```
Starting ThreadJoinTest
in thread 9 re was found at offset 1 in string xreo
in thread 10 fa was found at offset 1 in string xfao
in thread 12 la was found at offset 1 in string xlao
in thread 13 ti was found at offset 1 in string xtiotio
in thread 11 mi was found at offset 1 in string xmiomio
in thread 11 mi was found at offset 4 in string xmiomio
in thread 13 ti was found at offset 4 in string xtiotio
in thread 14 so was found at offset 1 in string xsoosoo
in thread 14 so was found at offset 4 in string xsoosoo
in thread 16 do was found at offset 1 in string xdoodoo
in thread 16 do was found at offset 4 in string xdoodoo
in thread 16 do was found at offset 1 in string xdoodoo
in thread 16 do was found at offset 4 in string xdoodoo
in thread 15 do was found at offset 1 in string xdoo
in thread 15 do was found at offset 1 in string xdoo
Ending ThreadJoinTest
```

Example of Starting & Joining with Java Threads

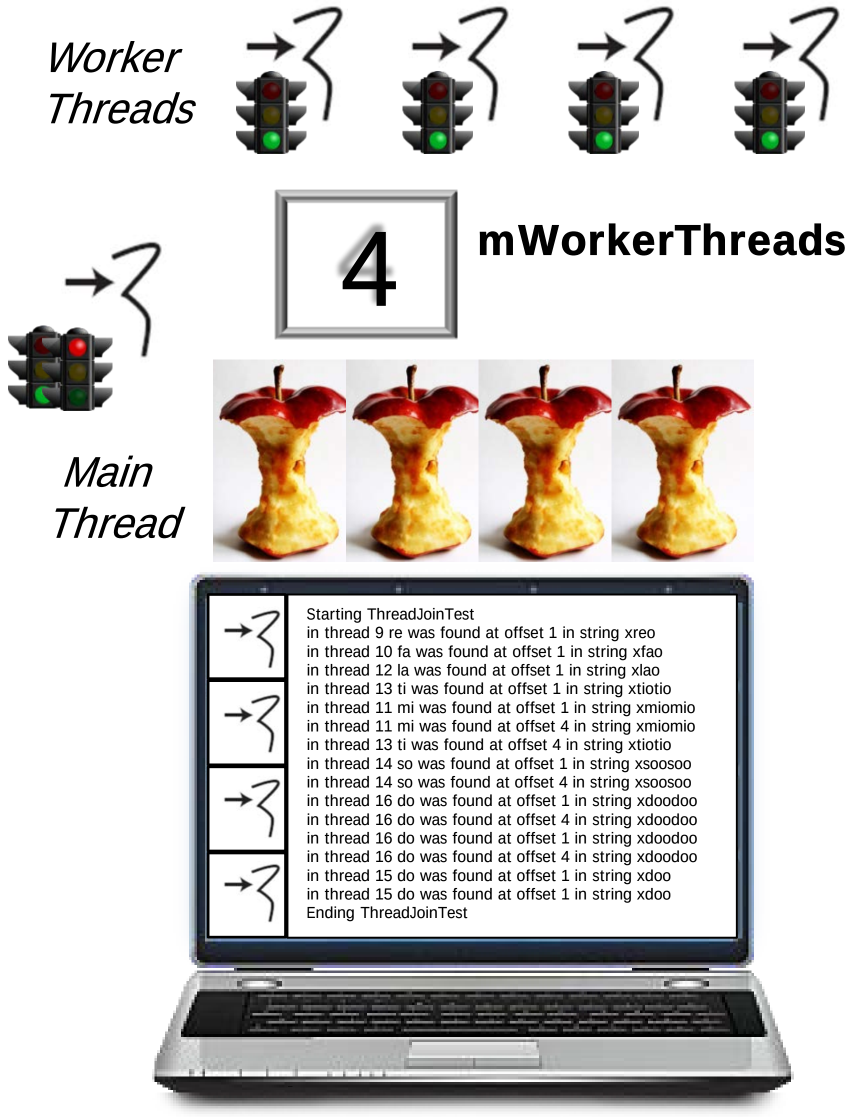
- Demonstrates the use of various Thread methods to implement an "embarrassingly parallel" application
  - Concurrently searches for words in a List of Strings
  - Calls Thread.start() to spawn a worker Thread for each element in the List of Strings
  - The main thread uses Thread.join() to wait for the worker Threads to finish



```
Starting ThreadJoinTest
in thread 9 re was found at offset 1 in string xreo
in thread 10 fa was found at offset 1 in string xfao
in thread 12 la was found at offset 1 in string xlao
in thread 13 ti was found at offset 1 in string xtiotio
in thread 11 mi was found at offset 1 in string xmiomio
in thread 11 mi was found at offset 4 in string xmiomio
in thread 13 ti was found at offset 4 in string xtiotio
in thread 14 so was found at offset 1 in string xsoosoo
in thread 14 so was found at offset 4 in string xsoosoo
in thread 16 do was found at offset 1 in string xdoodoo
in thread 16 do was found at offset 4 in string xdoodoo
in thread 16 do was found at offset 1 in string xdoodoo
in thread 16 do was found at offset 4 in string xdoodoo
in thread 15 do was found at offset 1 in string xdoo
in thread 15 do was found at offset 1 in string xdoo
Ending ThreadJoinTest
```

Example of Starting & Joining with Java Threads

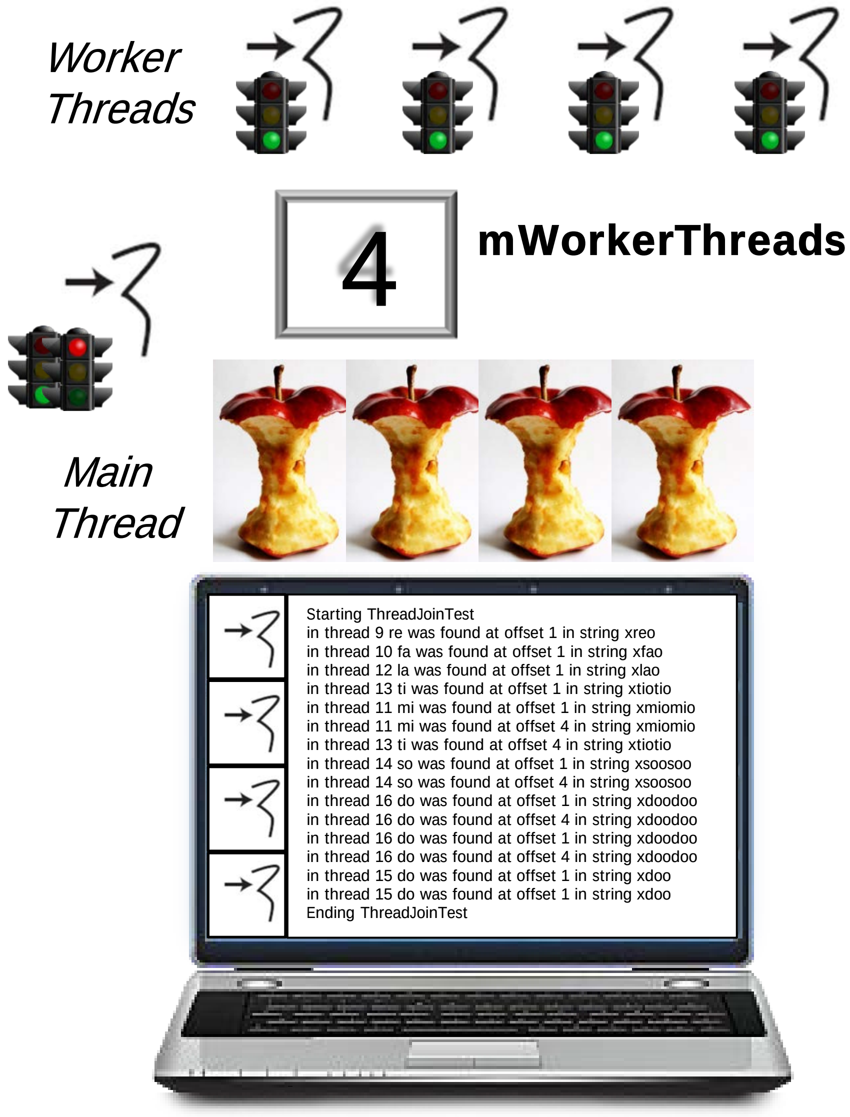
- Demonstrates the use of various Thread methods to implement an "embarrassingly parallel" application
  - Concurrently searches for words in a List of Strings
  - Calls Thread.start() to spawn a worker Thread for each element in the List of Strings
  - The main thread uses Thread.join() to wait for the worker Threads to finish
    - Thread.join() is a simple form of barrier synchronization



```
Starting ThreadJoinTest
in thread 9 re was found at offset 1 in string xreo
in thread 10 fa was found at offset 1 in string xfao
in thread 12 la was found at offset 1 in string xlao
in thread 13 ti was found at offset 1 in string xtiotio
in thread 11 mi was found at offset 1 in string xmiomio
in thread 11 mi was found at offset 4 in string xmiomio
in thread 13 ti was found at offset 4 in string xtiotio
in thread 14 so was found at offset 1 in string xsoosoo
in thread 14 so was found at offset 4 in string xsoosoo
in thread 16 do was found at offset 1 in string xdoodoo
in thread 16 do was found at offset 4 in string xdoodoo
in thread 16 do was found at offset 1 in string xdoodoo
in thread 16 do was found at offset 4 in string xdoodoo
in thread 15 do was found at offset 1 in string xdoo
in thread 15 do was found at offset 1 in string xdoo
Ending ThreadJoinTest
```

Example of Starting & Joining with Java Threads

- Demonstrates the use of various Thread methods to implement an "embarrassingly parallel" application
  - Concurrently searches for words in a List of Strings
  - Calls Thread.start() to spawn a worker Thread for each element in the List of Strings
  - The main thread uses Thread.join() to wait for the worker Threads to finish
    - Thread.join() is a simple form of barrier synchronization
    - No other Java synchronization mechanisms are needed



```
Starting ThreadJoinTest
in thread 9 re was found at offset 1 in string xreo
in thread 10 fa was found at offset 1 in string xfao
in thread 12 la was found at offset 1 in string xlao
in thread 13 ti was found at offset 1 in string xtiotio
in thread 11 mi was found at offset 1 in string xmiomio
in thread 11 mi was found at offset 4 in string xmiomio
in thread 13 ti was found at offset 4 in string xtiotio
in thread 14 so was found at offset 1 in string xsoosoo
in thread 14 so was found at offset 4 in string xsoosoo
in thread 16 do was found at offset 1 in string xdoodoo
in thread 16 do was found at offset 4 in string xdoodoo
in thread 16 do was found at offset 1 in string xdoodoo
in thread 16 do was found at offset 4 in string xdoodoo
in thread 15 do was found at offset 1 in string xdoo
in thread 15 do was found at offset 1 in string xdoo
Ending ThreadJoinTest
```

Upcoming parts show other examples of Java synchronization mechanisms