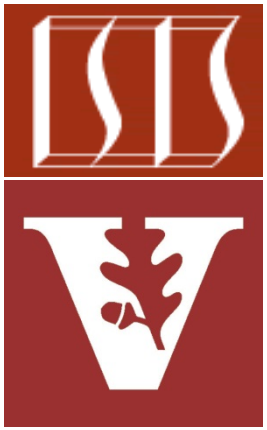
Android Concurrency: Overview of Java Threads (Part 1)

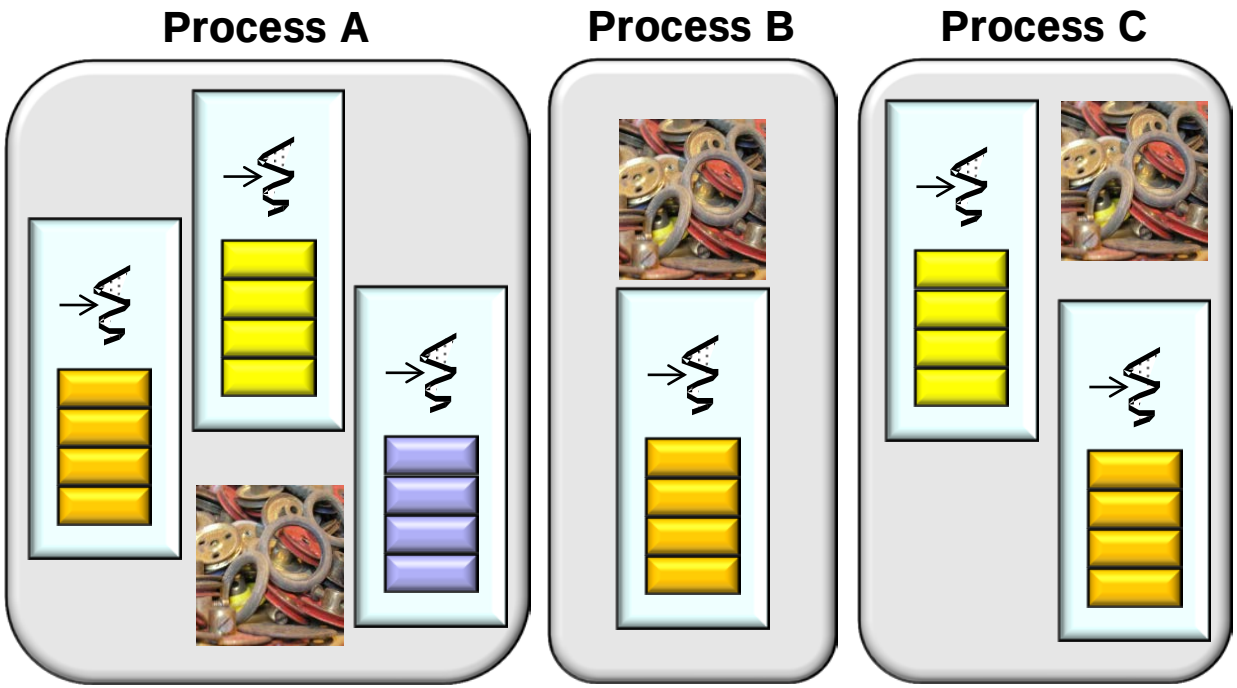


Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

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

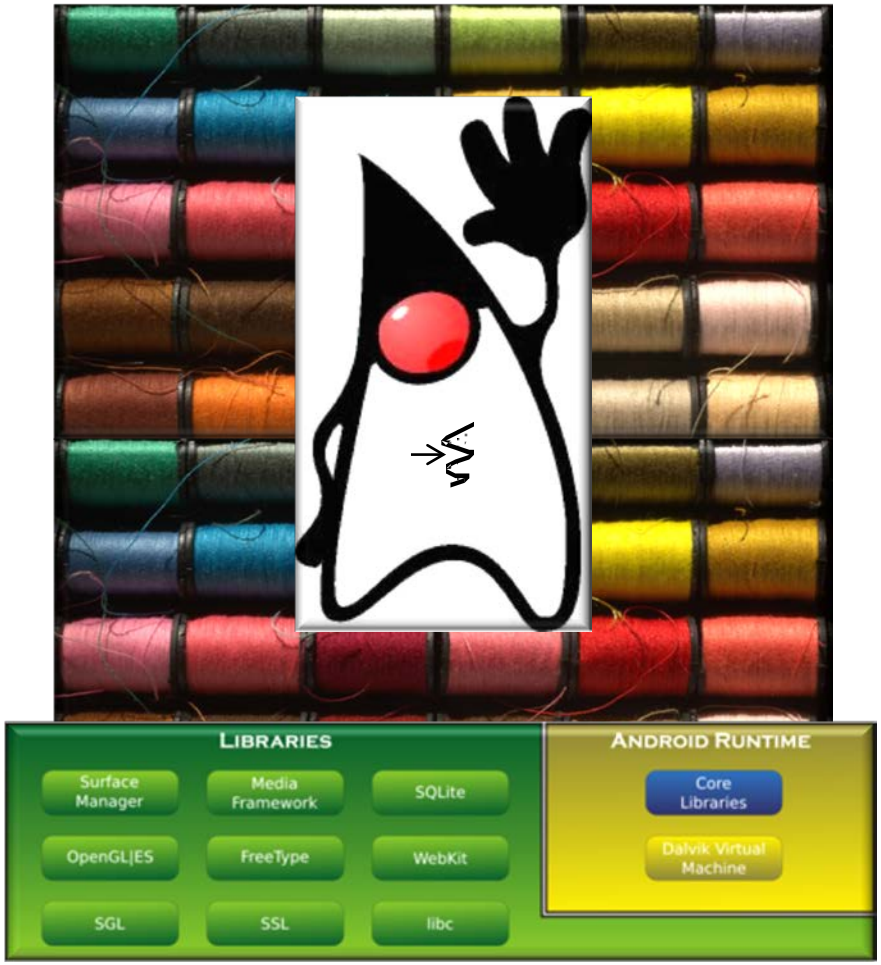
Learning Objectives in this Part of the Module

- Understand the Java mechanisms available in Android to implement *concurrent* software that processes requests simultaneously via multithreading



Overview of Java Threads in Android

- Android implements many standard Java concurrency & synchronization classes

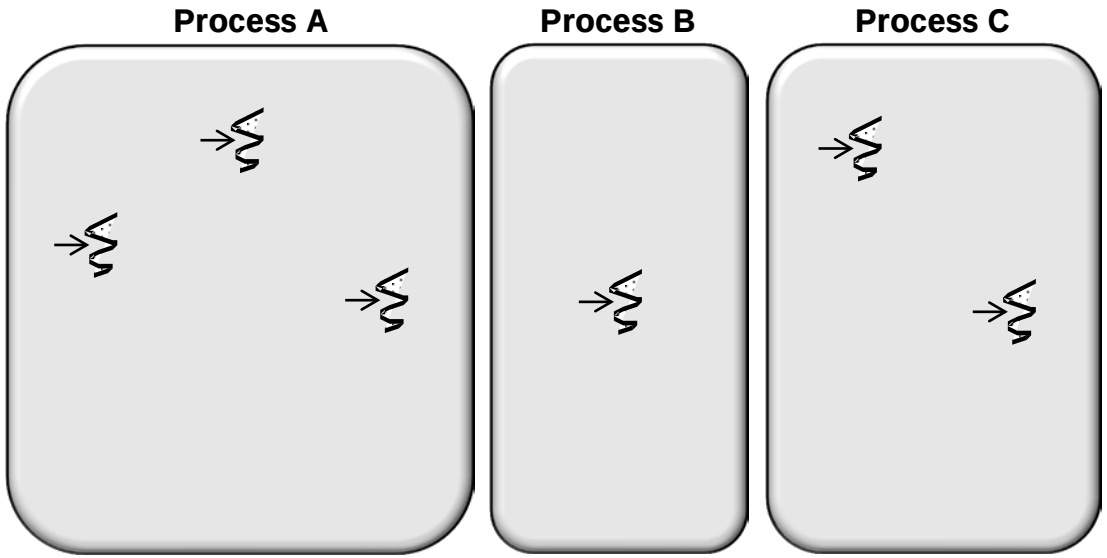


Overview of Java Threads in Android

- Android implements many standard Java concurrency & synchronization classes
- **Conceptual view**
  - A thread is a unit of computation that runs in the context of a process

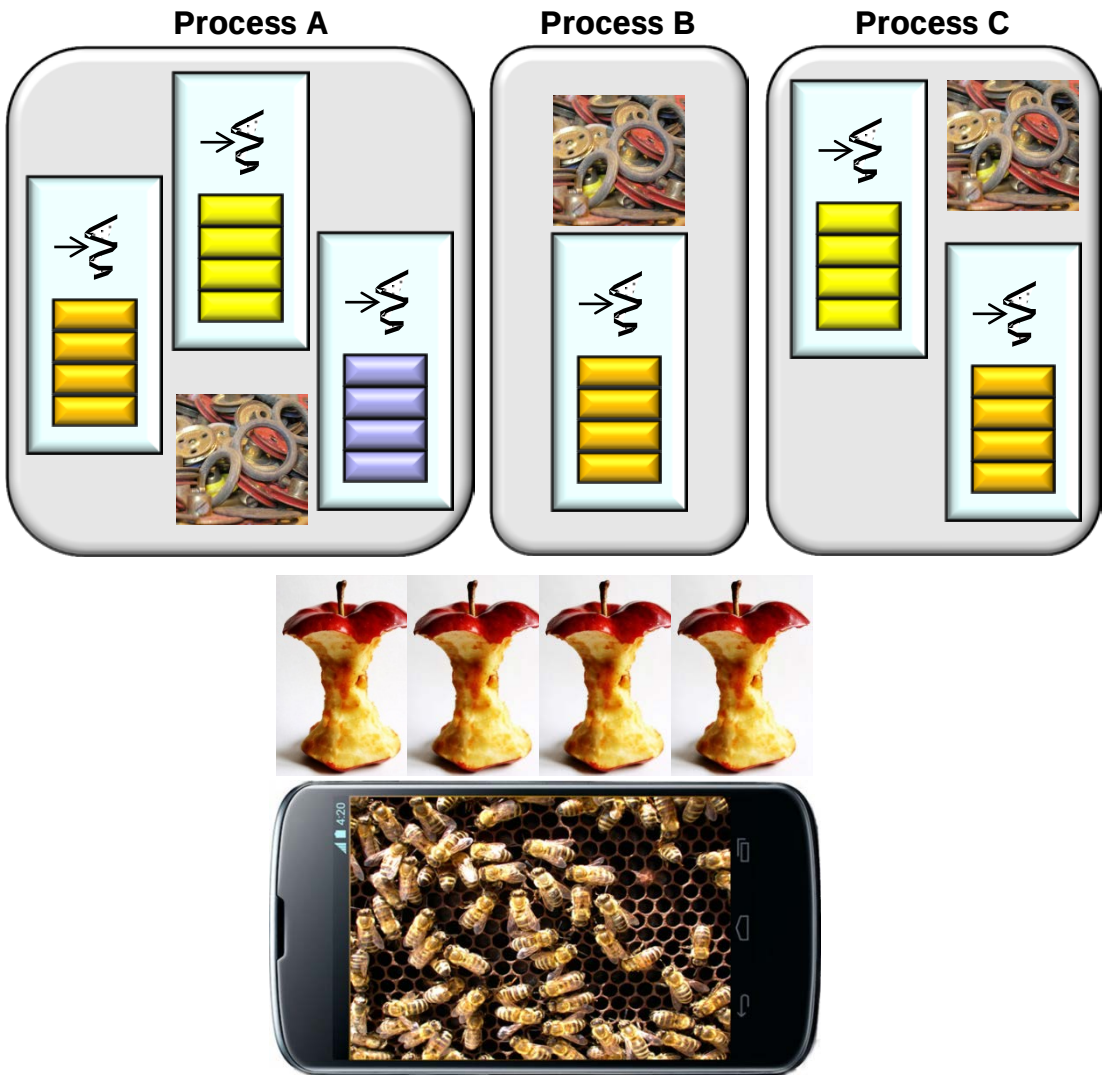
Overview of Java Threads in Android

- Android implements many standard Java concurrency & synchronization classes
- **Conceptual view**
  - A thread is a unit of computation that runs in the context of a process
  - Threads running in a process can communicate with each other via shared objects or message passing



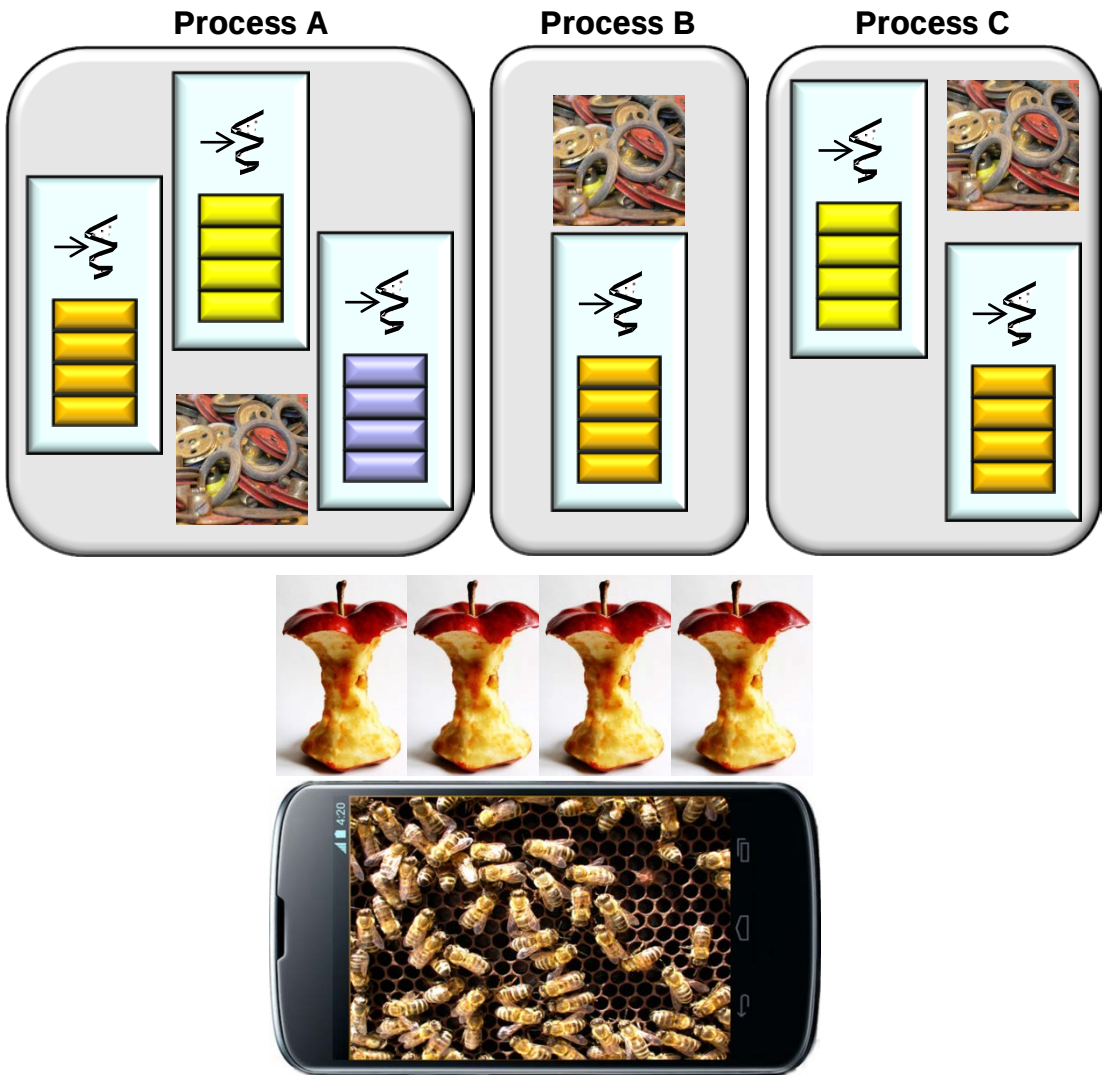
Overview of Java Threads in Android

- Android implements many standard Java concurrency & synchronization classes
- **Conceptual view**
- **Implementation view**
  - Each Java thread has a a stack, a program counter, & other registers (unique "state")



Overview of Java Threads in Android

- Android implements many standard Java concurrency & synchronization classes
- **Conceptual view**
- **Implementation view**
  - Each Java thread has a a stack, a program counter, & other registers (unique "state")
  - The heap & static areas are shared across threads (common "state")



Programming Java Threads in Android

Programming Java Threads in Android

Programming Java Threads in Android

- All threads must be given code to run by either
  - Extending the Thread class

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

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

Programming Java Threads in Android

- All threads must be given code to run by either
  - Extending the Thread class

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

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

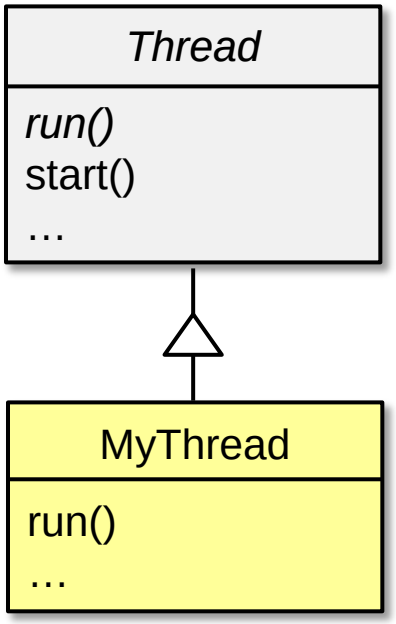
Programming Java Threads in Android

- All threads must be given code to run by either
  - Extending the Thread class

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

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

Programming Java Threads in Android

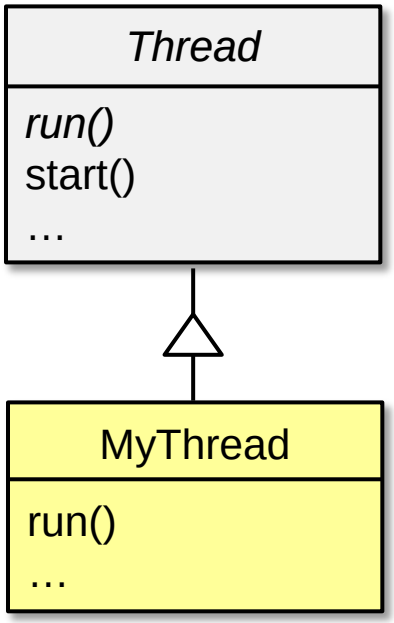
- All threads must be given code to run by either
  - Extending the Thread class




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

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

Programming Java Threads in Android

- All threads must be given code to run by either
  - Extending the Thread class




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

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

Starting a thread using a named class

Programming Java Threads in Android

- All threads must be given code to run by either
  - Extending the Thread class
  - Implementing the Runnable interface

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

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

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

Programming Java Threads in Android

- All threads must be given code to run by either
  - Extending the Thread class
  - Implementing the Runnable interface

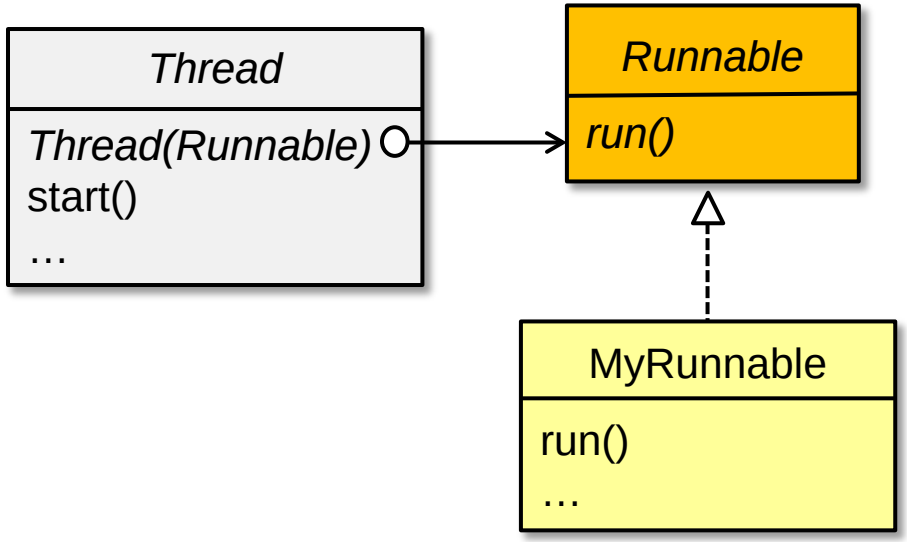
```
public interface Runnable {
   public void run();
}
```

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

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

Programming Java Threads in Android

- All threads must be given code to run by either
  - Extending the Thread class
  - Implementing the Runnable interface

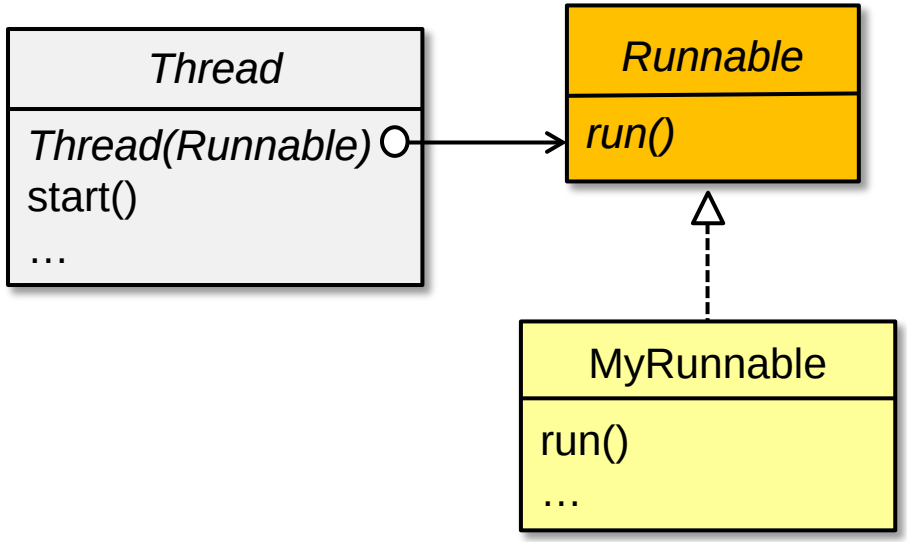
```
public interface Runnable {
   public void run();
}

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

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



Programming Java Threads in Android

- All threads must be given code to run by either
  - Extending the Thread class
  - Implementing the Runnable interface

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

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

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

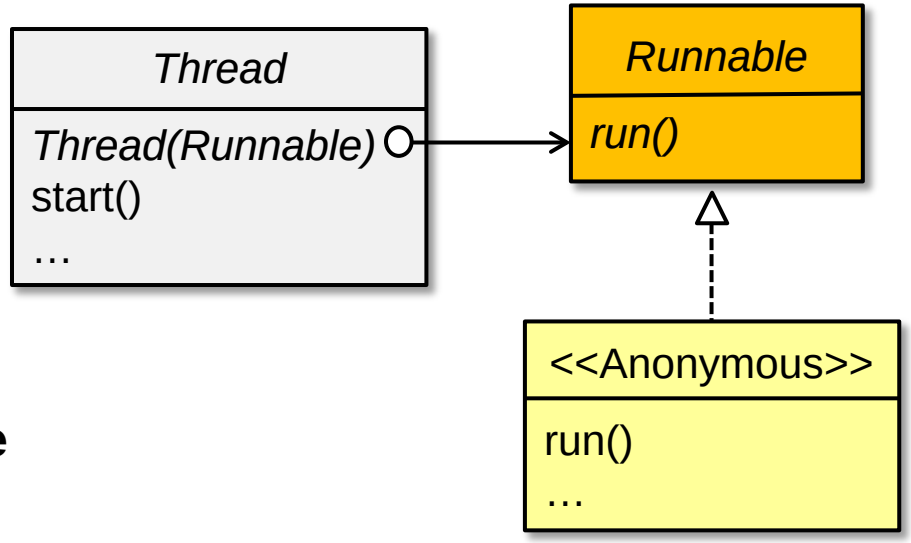


Define/start a thread using a named Runnable implementation

Programming Java Threads in Android

- All threads must be given code to run by either
  - Extending the Thread class
  - Implementing the Runnable interface

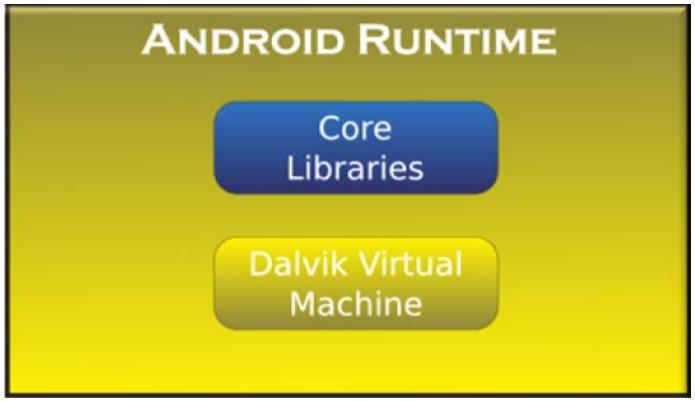
```
public interface Runnable {
   public void run();
}

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

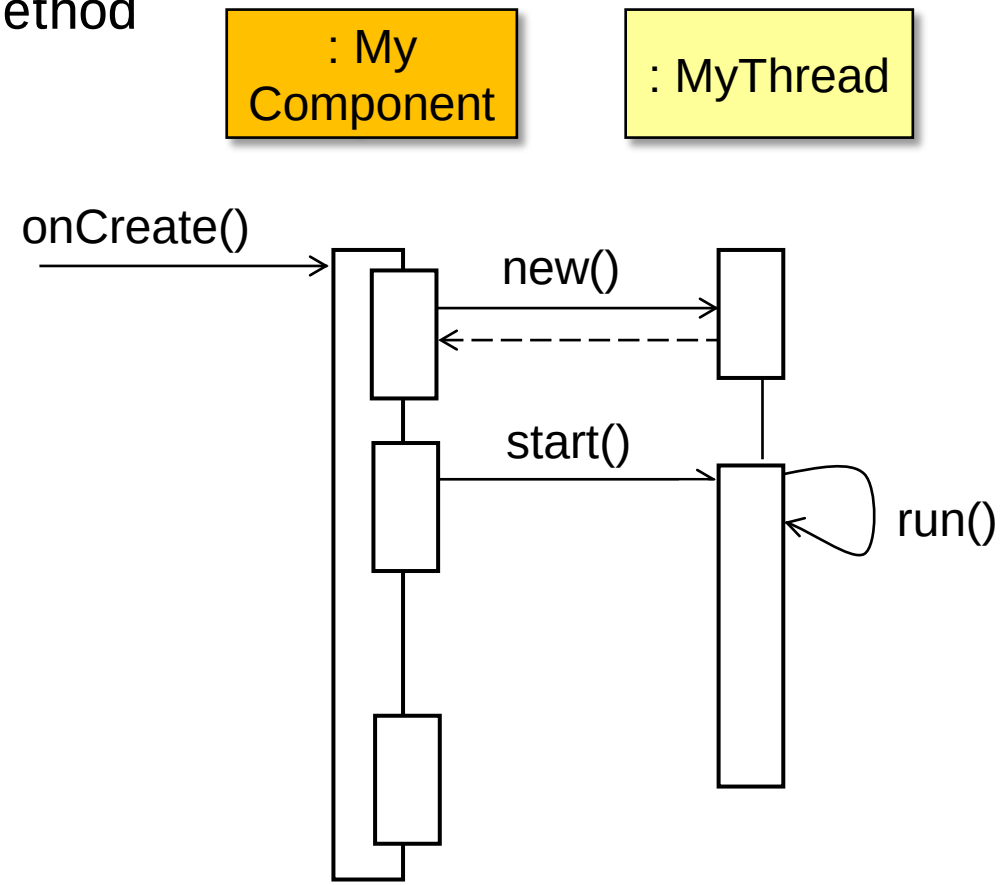


Define/start a thread using an anonymous inner class as the Runnable

Programming Java Threads in Android

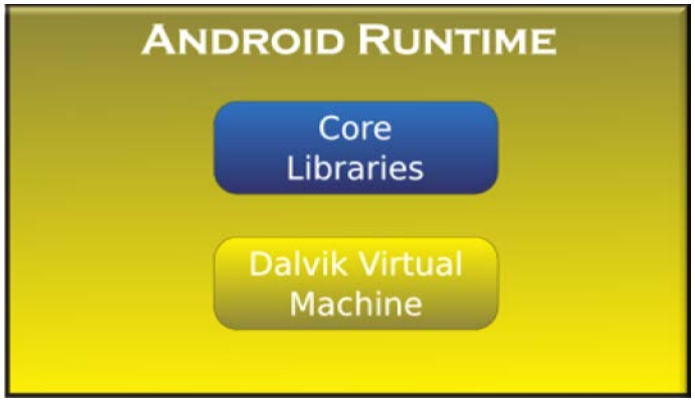
- All threads must be given code to run
- After creating the thread's resources, the Android JVM calls its run() hook method

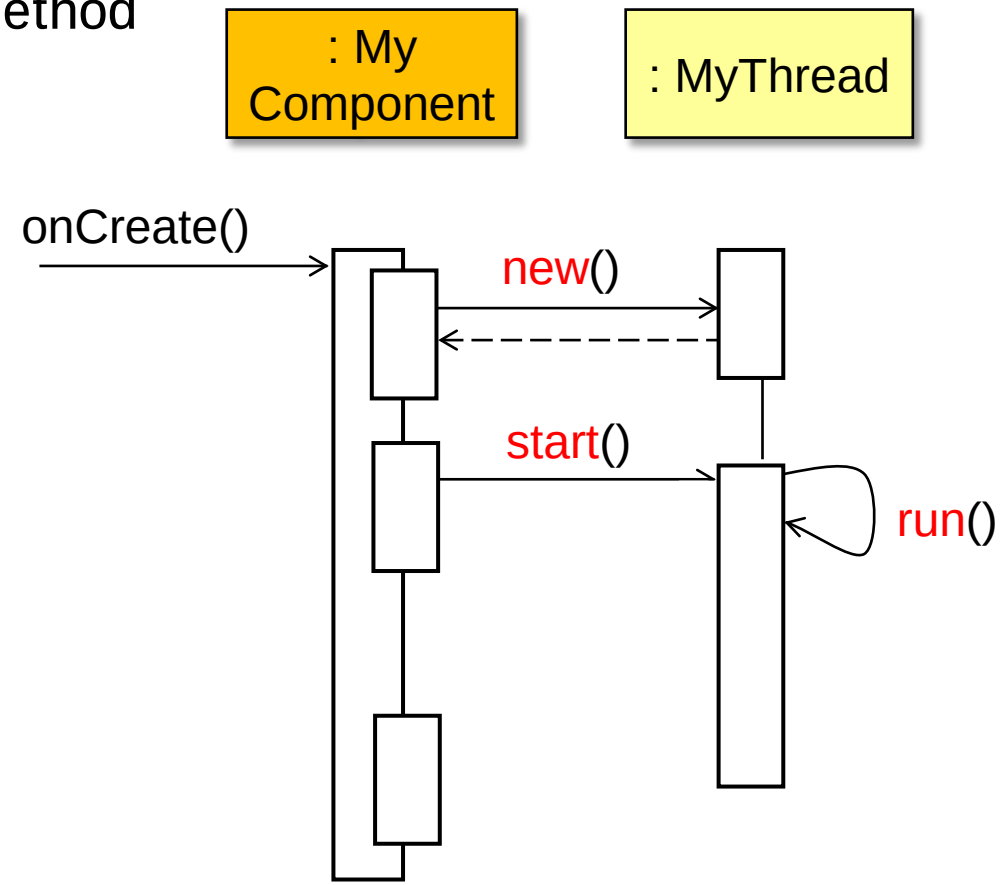




Programming Java Threads in Android

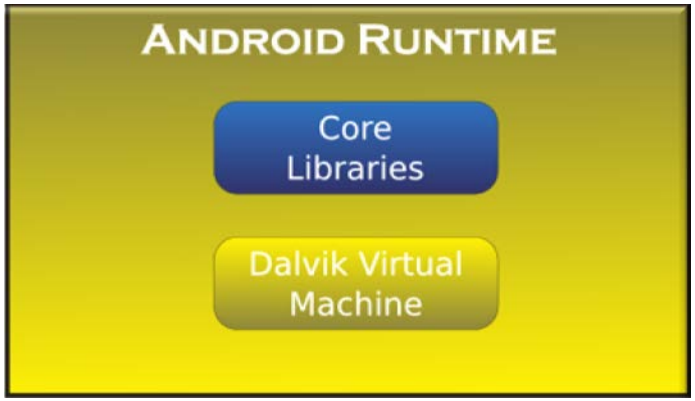
- All threads must be given code to run
- After creating the thread's resources, the Android JVM calls its run() hook method

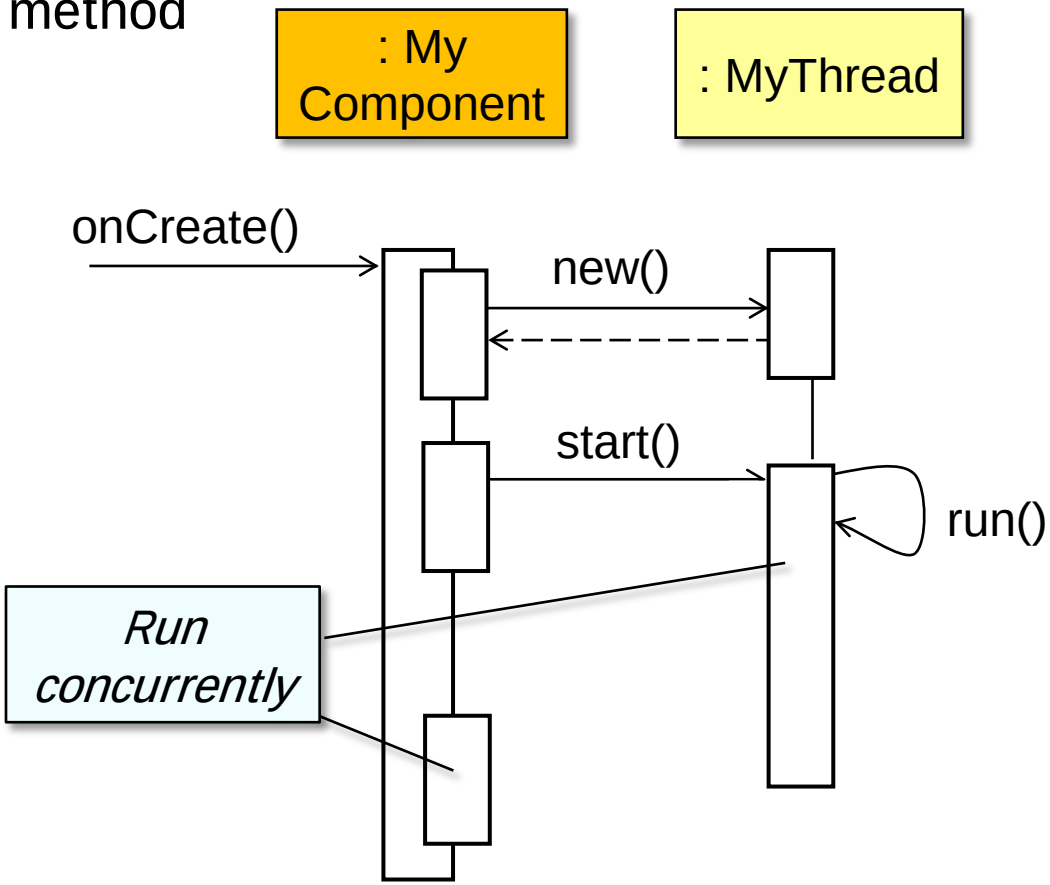




Programming Java Threads in Android

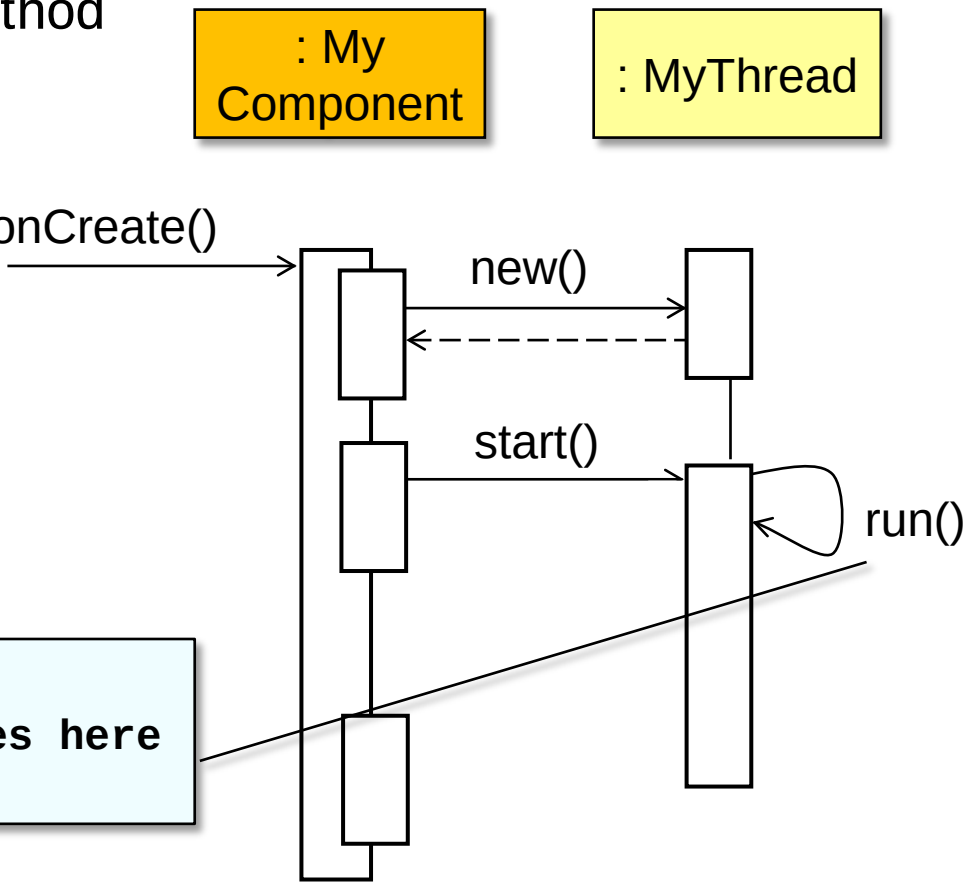
- All threads must be given code to run
- After creating the thread's resources, the Android JVM calls its run() hook method





Programming Java Threads in Android

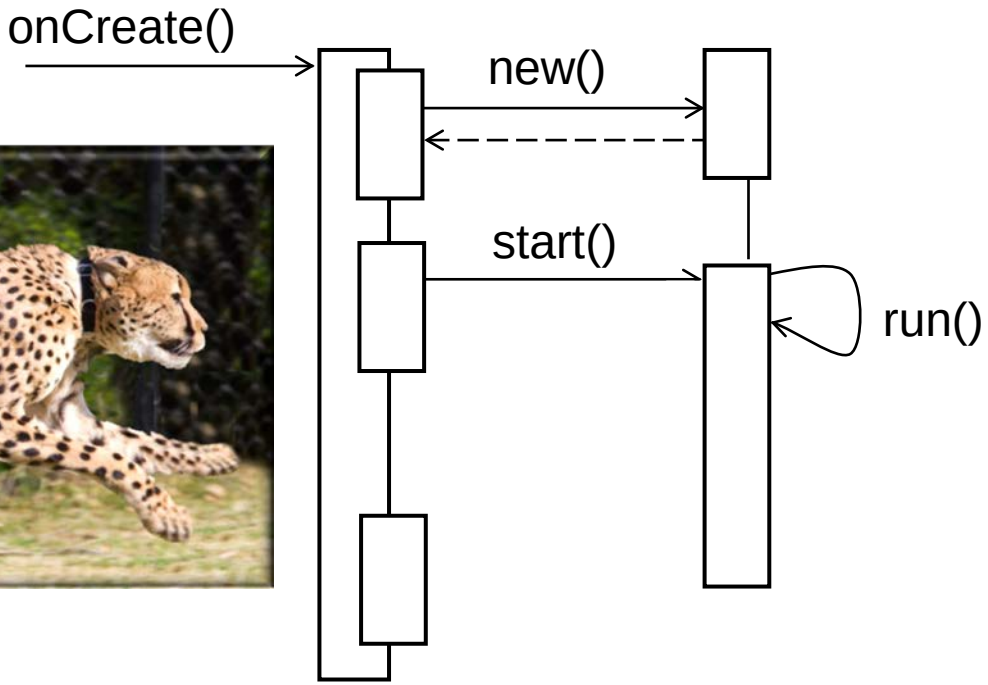
- All threads must be given code to run
- After creating the thread's resources, the Android JVM calls its run() hook method
  - You can generally run any code in a thread, though Android imposes certain restrictions on GUI operations




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

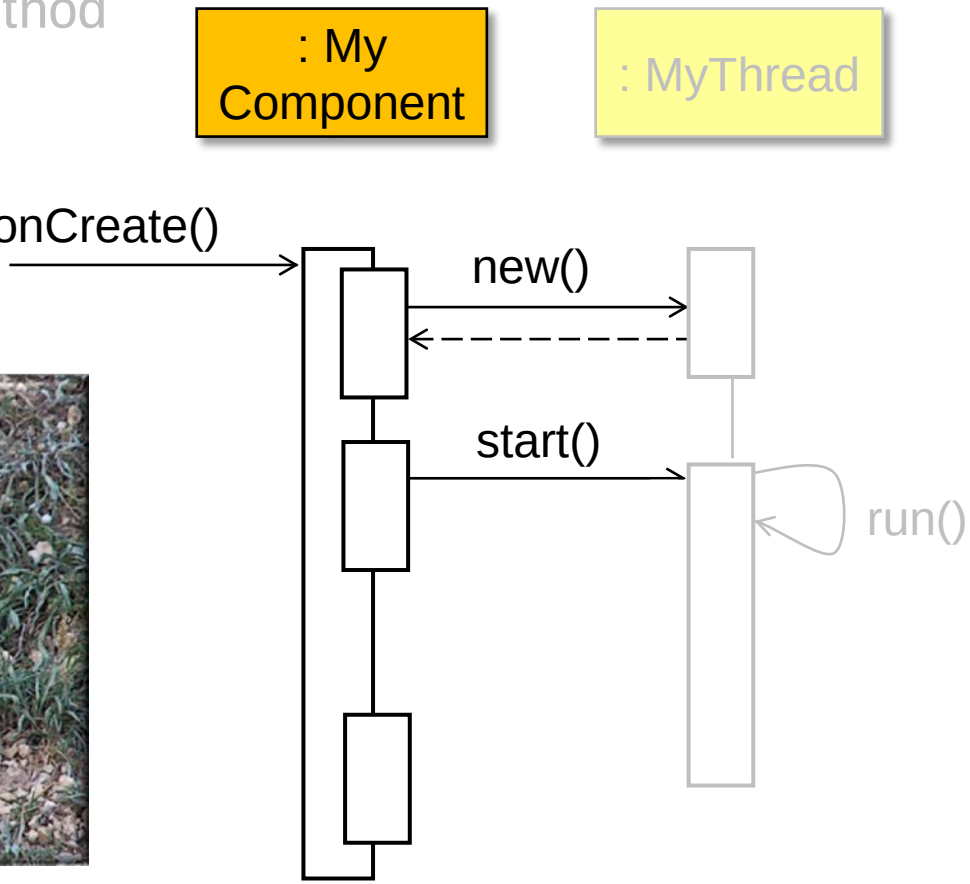
Programming Java Threads in Android

- All threads must be given code to run
- After creating the thread's resources, the Android JVM calls its run() hook method
- A thread can run as long as its run() method hasn't returned



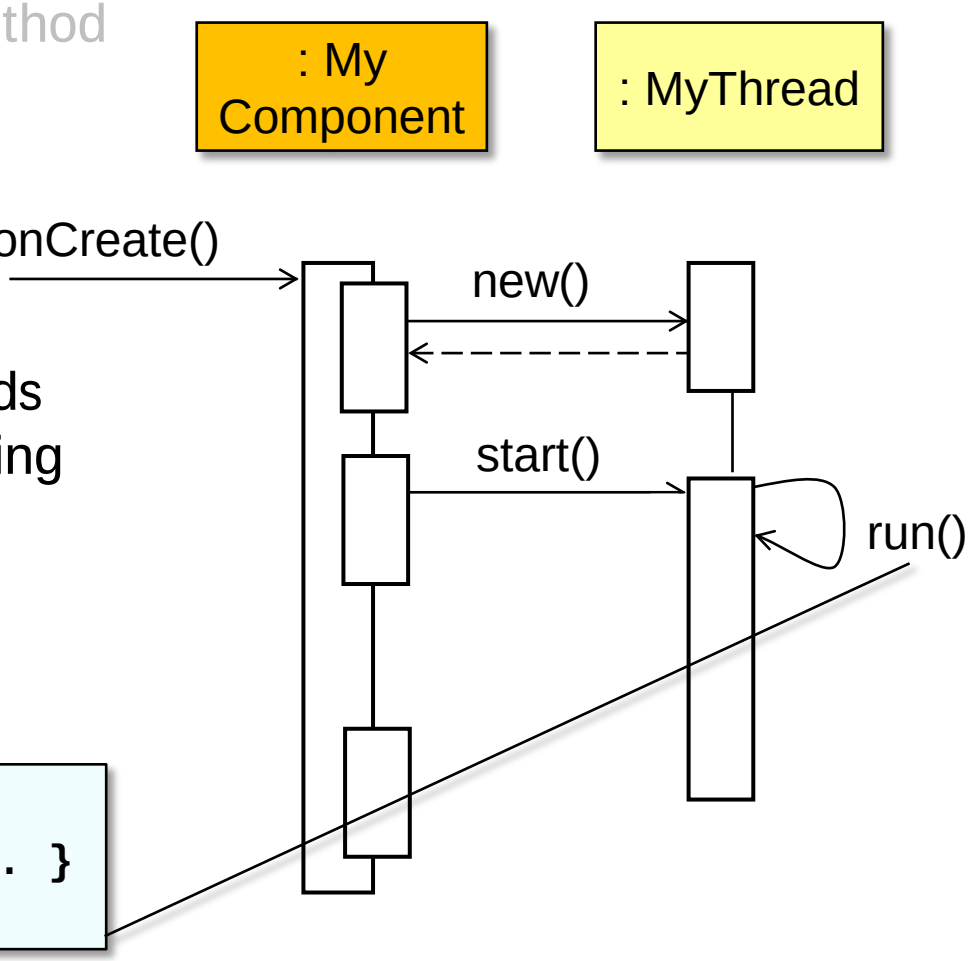
Programming Java Threads in Android

- All threads must be given code to run
- After creating the thread's resources, the Android JVM calls its run() hook method
- A thread can run as long as its run() method hasn't returned
- Android's scheduler can suspend & resume a thread many times



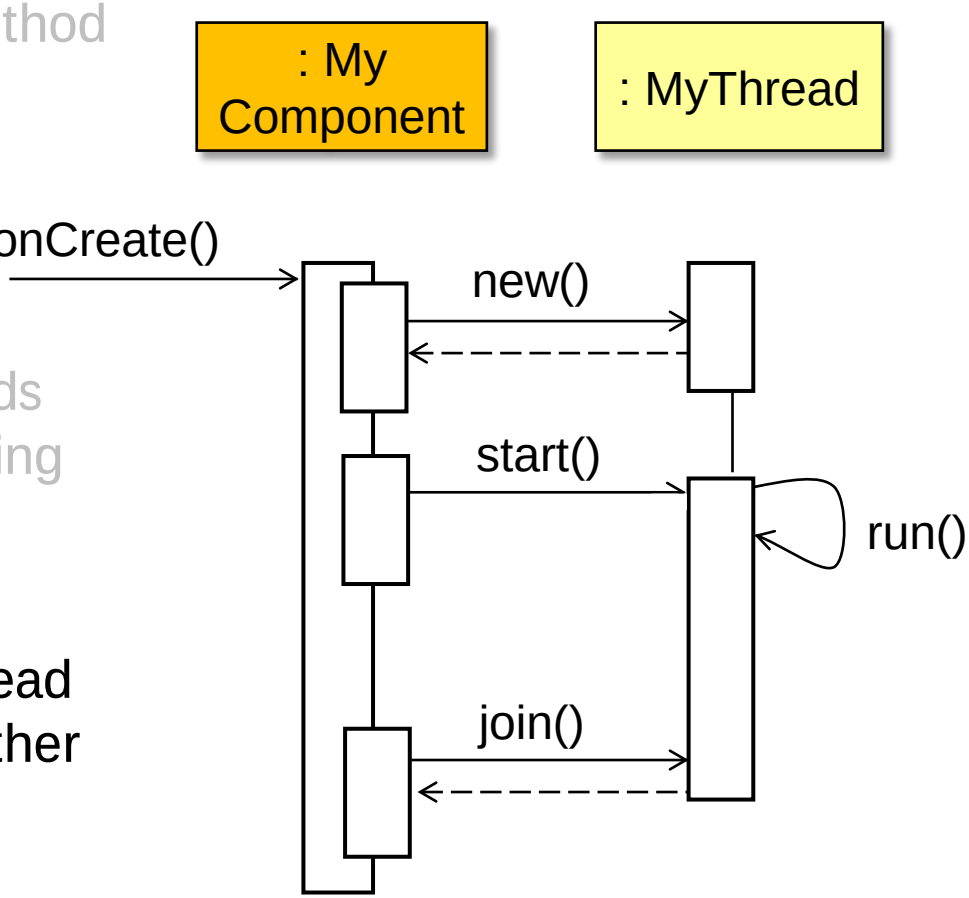
Programming Java Threads in Android

- All threads must be given code to run
- After creating the thread's resources, the Android JVM calls its run() hook method
- A thread can run as long as its run() method hasn't returned
- Android's scheduler can suspend & resume a thread many times
- For a thread to run "forever," it needs a loop that keeps run() from returning



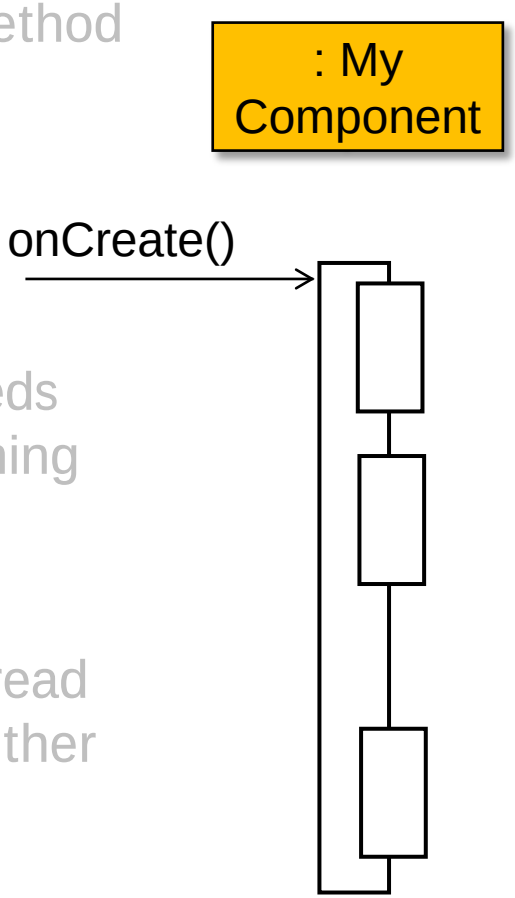
Programming Java Threads in Android

- All threads must be given code to run
- After creating the thread's resources, the Android JVM calls its run() hook method
- A thread can run as long as its run() method hasn't returned
- Android's scheduler can suspend & resume a thread many times
- For a thread to run "forever," it needs a loop that keeps run() from returning
- After run() returns the thread is no longer active
  - The join() method allows one thread to wait for the completion of another



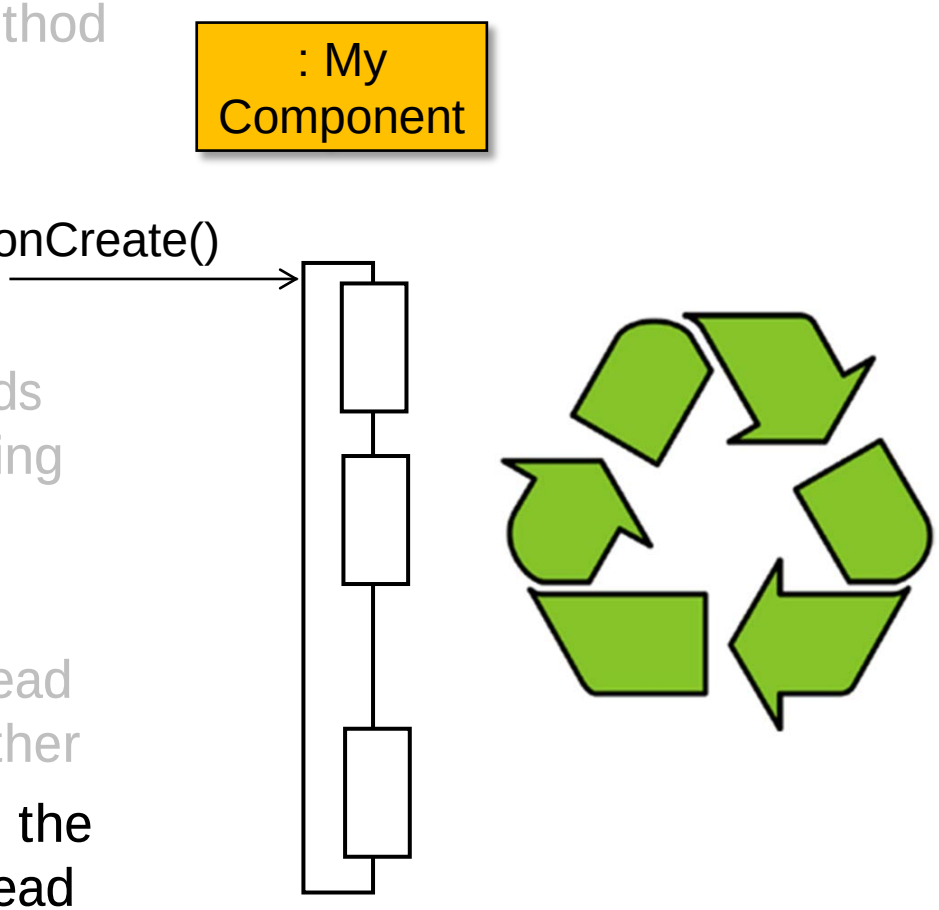
Programming Java Threads in Android

- All threads must be given code to run
- After creating the thread's resources, the Android JVM calls its run() hook method
- A thread can run as long as its run() method hasn't returned
- Android's scheduler can suspend & resume a thread many times
- For a thread to run "forever," it needs a loop that keeps run() from returning
- After run() returns the thread is no longer active
  - The join() method allows one thread to wait for the completion of another



Programming Java Threads in Android

- All threads must be given code to run
- After creating the thread's resources, the Android JVM calls its run() hook method
- A thread can run as long as its run() method hasn't returned
- Android's scheduler can suspend & resume a thread many times
- For a thread to run "forever," it needs a loop that keeps run() from returning
- After run() returns the thread is no longer active
  - The join() method allows one thread to wait for the completion of another
  - The Java virtual machine recycles the resources associated with the thread



Some Common Java Thread Methods

Some Common Java Thread Methods

public class

Thread

extends Object
implements Runnable

Summary: Nested Classes | Constants | Ctors | Methods | Inherited Methods | [Expand All]

Added in API level 1

java.lang.Object
↳java.lang.Thread

▸ Known Direct Subclasses
HandlerThread

Class Overview

A `Thread` is a concurrent unit of execution. It has its own call stack for methods being invoked, their arguments and local variables. Each application has at least one thread running when it is started, the main thread, in the main `ThreadGroup`. The runtime keeps its own threads in the system thread group.

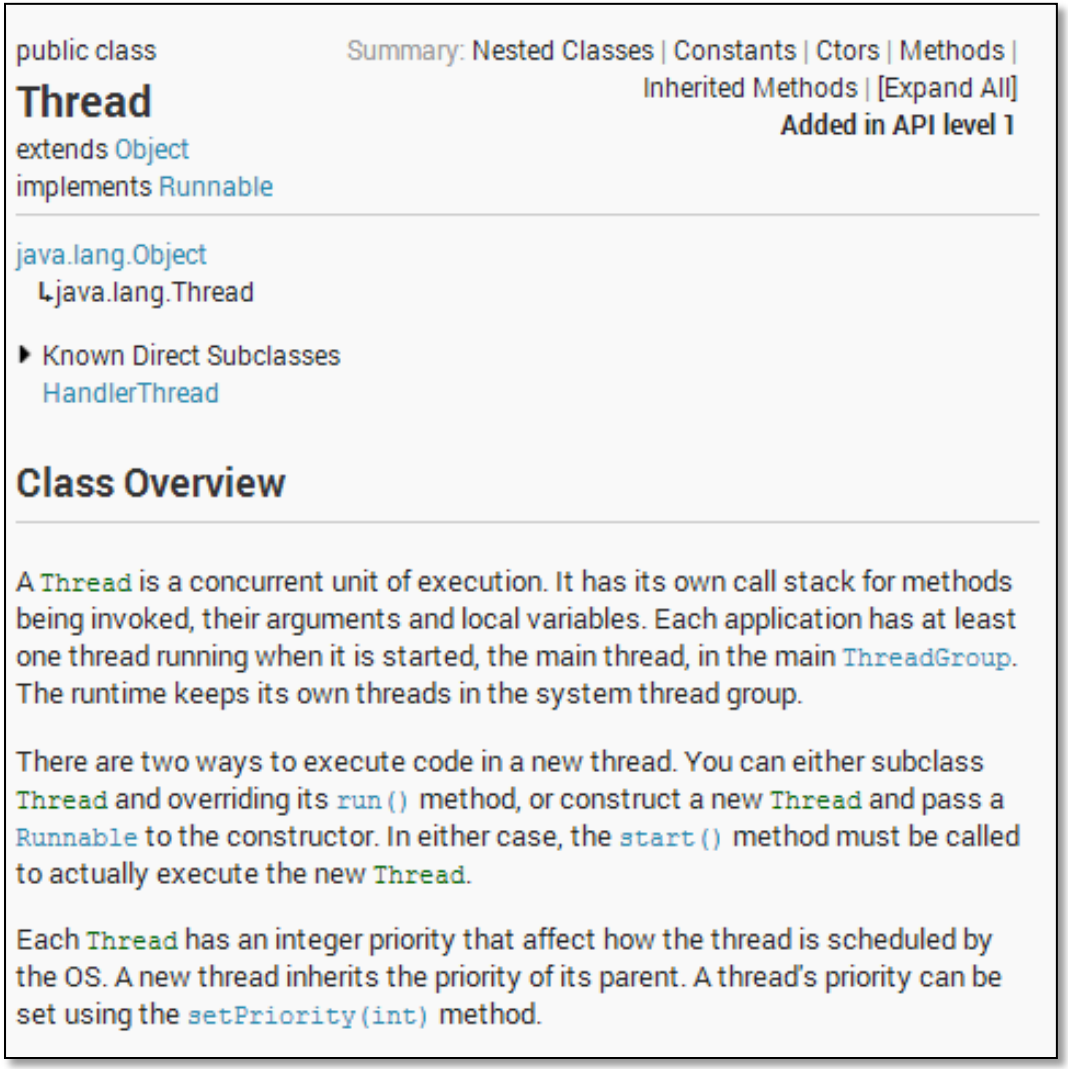
There are two ways to execute code in a new thread. You can either subclass `Thread` and overriding its `run()` method, or construct a new `Thread` and pass a `Runnable` to the constructor. In either case, the `start()` method must be called to actually execute the new `Thread`.

Each `Thread` has an integer priority that affect how the thread is scheduled by the OS. A new thread inherits the priority of its parent. A thread's priority can be set using the `setPriority(int)` method.

developer.android.com/reference/java/lang/Thread.html has more info

Some Common Java Thread Methods

- **`void start()`**
  - Initiates thread execution



public class

Thread

extends Object
implements Runnable

Summary: Nested Classes | Constants | Ctors | Methods | Inherited Methods | [Expand All]

Added in API level 1

java.lang.Object
↳java.lang.Thread

▸ Known Direct Subclasses
HandlerThread

Class Overview

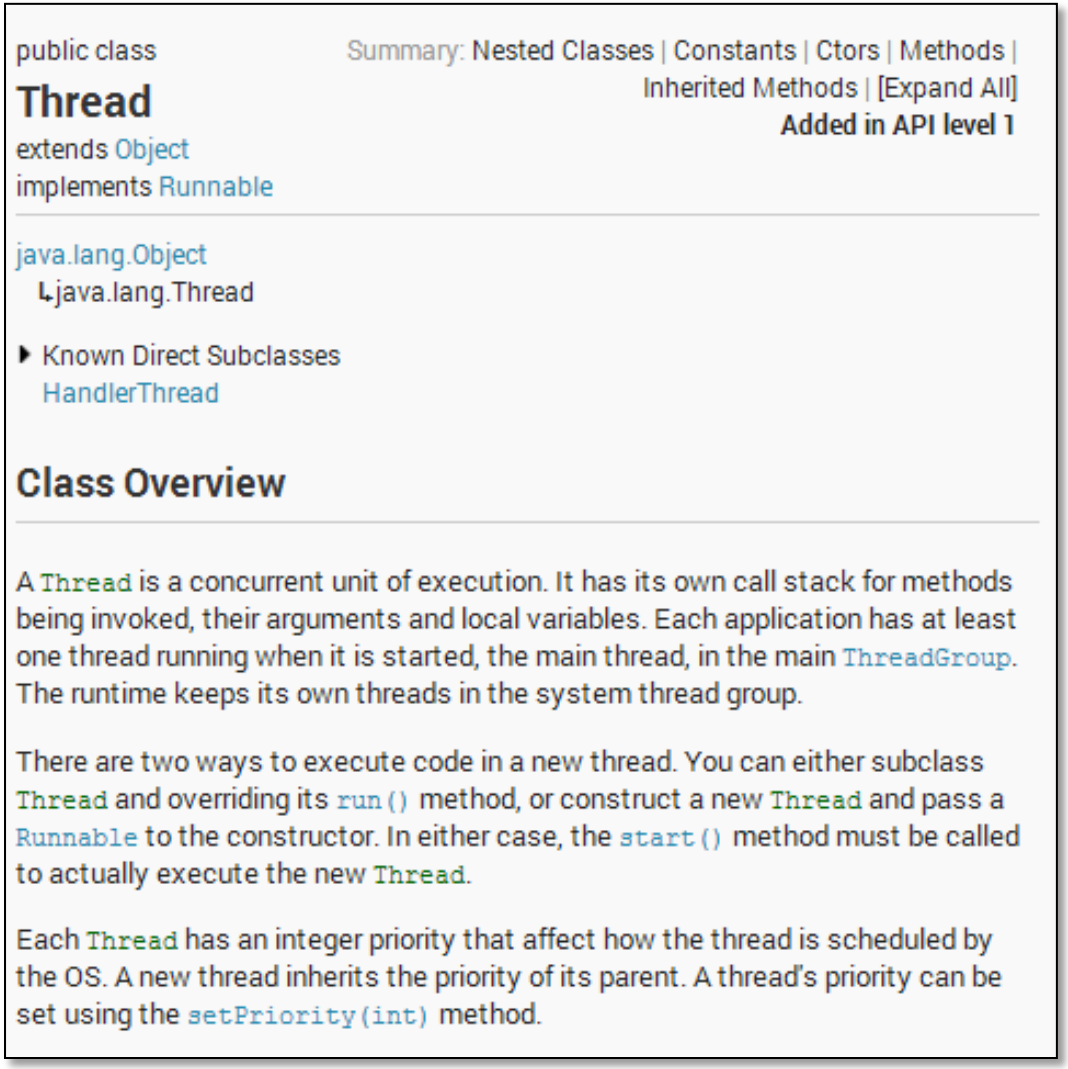
A `Thread` is a concurrent unit of execution. It has its own call stack for methods being invoked, their arguments and local variables. Each application has at least one thread running when it is started, the main thread, in the main `ThreadGroup`. The runtime keeps its own threads in the system thread group.

There are two ways to execute code in a new thread. You can either subclass `Thread` and overriding its `run()` method, or construct a new `Thread` and pass a `Runnable` to the constructor. In either case, the `start()` method must be called to actually execute the new `Thread`.

Each `Thread` has an integer priority that affect how the thread is scheduled by the OS. A new thread inherits the priority of its parent. A thread's priority can be set using the `setPriority(int)` method.

Some Common Java Thread Methods

- `void start()`
  - Initiates thread execution
- **`void join()`**
  - Waits for a thread to finish



public class
Thread
extends Object
implements Runnable

Summary: Nested Classes | Constants | Ctors | Methods | Inherited Methods | [Expand All]
Added in API level 1

java.lang.Object
↳java.lang.Thread

▸ Known Direct Subclasses
HandlerThread

Class Overview

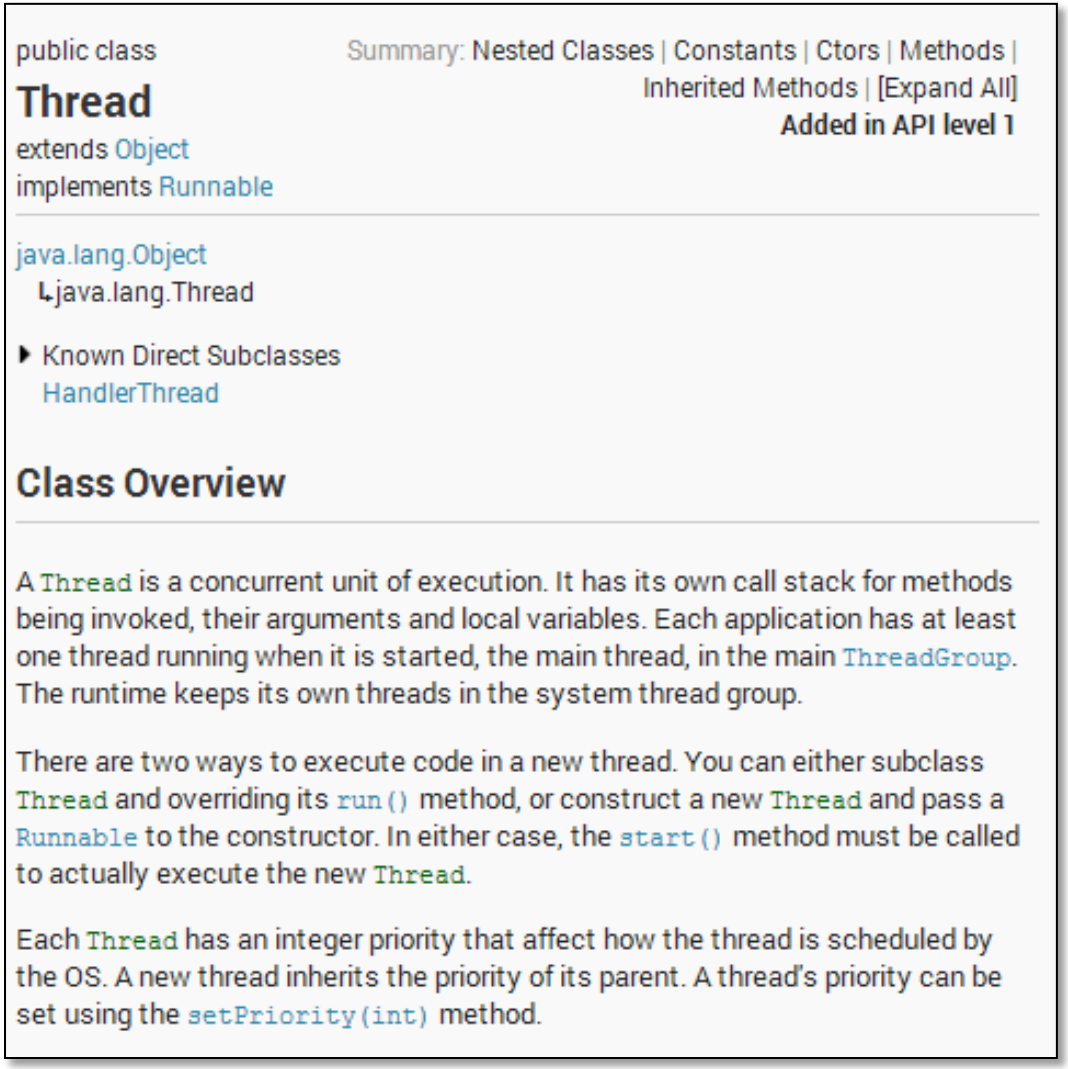
A `Thread` is a concurrent unit of execution. It has its own call stack for methods being invoked, their arguments and local variables. Each application has at least one thread running when it is started, the main thread, in the main `ThreadGroup`. The runtime keeps its own threads in the system thread group.

There are two ways to execute code in a new thread. You can either subclass `Thread` and overriding its `run()` method, or construct a new `Thread` and pass a `Runnable` to the constructor. In either case, the `start()` method must be called to actually execute the new `Thread`.

Each `Thread` has an integer priority that affect how the thread is scheduled by the OS. A new thread inherits the priority of its parent. A thread's priority can be set using the `setPriority(int)` method.

Some Common Java Thread Methods

- `void start()`
  - Initiates thread execution
- `void join()`
  - Waits for a thread to finish
- **`abstract void run()`**
  - Hook method for user code



public class

Thread

extends Object
implements Runnable

Summary: Nested Classes | Constants | Ctors | Methods | Inherited Methods | [Expand All]

Added in API level 1

java.lang.Object
↳java.lang.Thread

▸ Known Direct Subclasses
HandlerThread

Class Overview

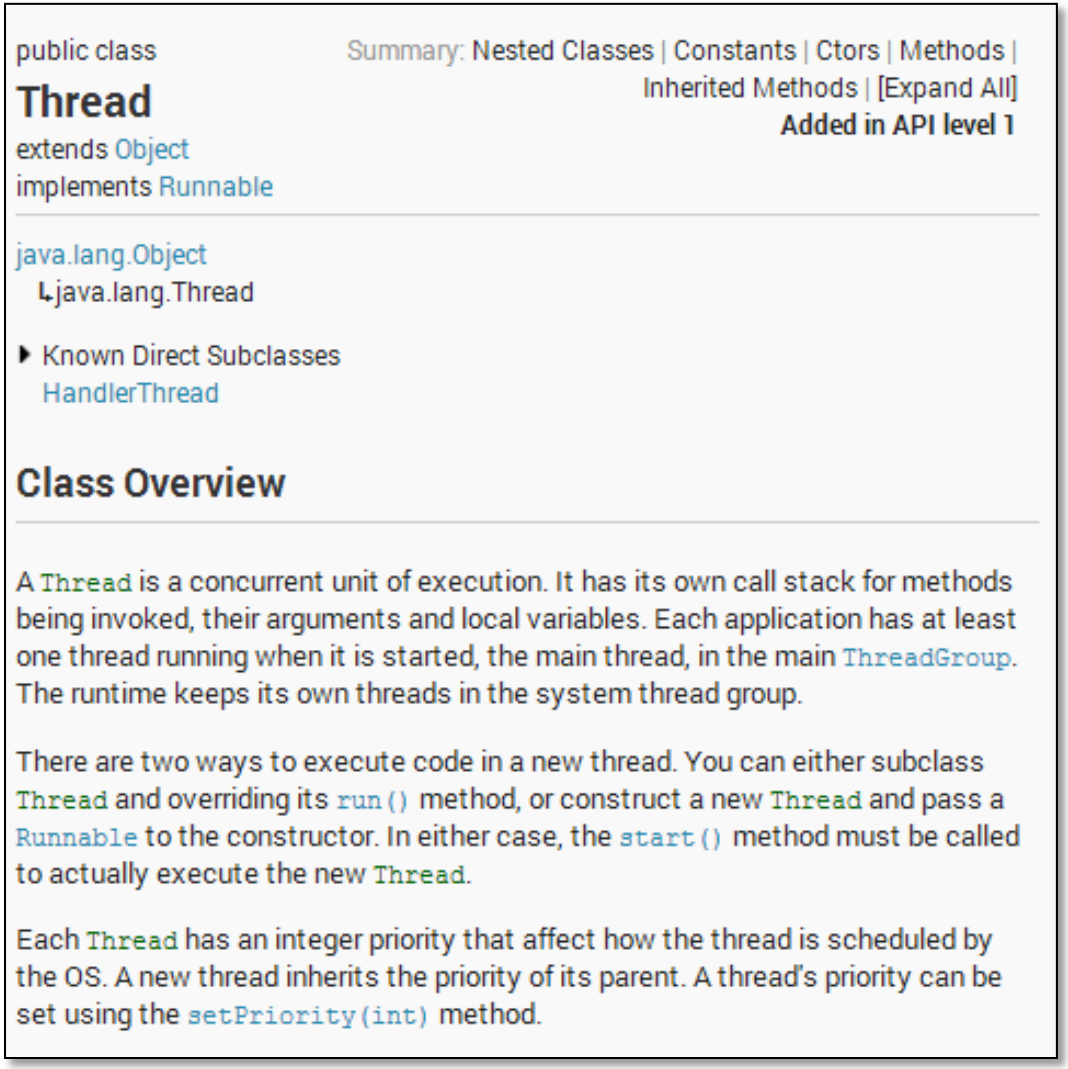
A `Thread` is a concurrent unit of execution. It has its own call stack for methods being invoked, their arguments and local variables. Each application has at least one thread running when it is started, the main thread, in the main `ThreadGroup`. The runtime keeps its own threads in the system thread group.

There are two ways to execute code in a new thread. You can either subclass `Thread` and overriding its `run()` method, or construct a new `Thread` and pass a `Runnable` to the constructor. In either case, the `start()` method must be called to actually execute the new `Thread`.

Each `Thread` has an integer priority that affect how the thread is scheduled by the OS. A new thread inherits the priority of its parent. A thread's priority can be set using the `setPriority(int)` method.

Some Common Java Thread Methods

- `void start()`
  - Initiates thread execution
- `void join()`
  - Waits for a thread to finish
- `abstract void run()`
  - Hook method for user code
- **`void sleep(long time)`**
  - Sleeps for given time in ms



public class

Thread

extends Object
implements Runnable

Summary: Nested Classes | Constants | Ctors | Methods | Inherited Methods | [Expand All]

Added in API level 1

java.lang.Object
↳java.lang.Thread

▸ Known Direct Subclasses
HandlerThread

Class Overview

A `Thread` is a concurrent unit of execution. It has its own call stack for methods being invoked, their arguments and local variables. Each application has at least one thread running when it is started, the main thread, in the main `ThreadGroup`. The runtime keeps its own threads in the system thread group.

There are two ways to execute code in a new thread. You can either subclass `Thread` and overriding its `run()` method, or construct a new `Thread` and pass a `Runnable` to the constructor. In either case, the `start()` method must be called to actually execute the new `Thread`.

Each `Thread` has an integer priority that affect how the thread is scheduled by the OS. A new thread inherits the priority of its parent. A thread's priority can be set using the `setPriority(int)` method.

Some Common Java Thread Methods

- `void start()`
  - Initiates thread execution
- `void join()`
  - Waits for a thread to finish
- `abstract void run()`
  - Hook method for user code
- `void sleep(long time)`
  - Sleeps for given time in ms
- **`void interrupt()`**
  - Post an interrupt request to a Thread

public class **Thread**
extends Object
implements Runnable

Summary: Nested Classes | Constants | Ctors | Methods | Inherited Methods | [Expand All]
Added in API level 1

java.lang.Object
↳java.lang.Thread

▸ Known Direct Subclasses
HandlerThread

Class Overview

A `Thread` is a concurrent unit of execution. It has its own call stack for methods being invoked, their arguments and local variables. Each application has at least one thread running when it is started, the main thread, in the main `ThreadGroup`. The runtime keeps its own threads in the system thread group.

There are two ways to execute code in a new thread. You can either subclass `Thread` and overriding its `run()` method, or construct a new `Thread` and pass a `Runnable` to the constructor. In either case, the `start()` method must be called to actually execute the new `Thread`.

Each `Thread` has an integer priority that affect how the thread is scheduled by the OS. A new thread inherits the priority of its parent. A thread's priority can be set using the `setPriority(int)` method.

Some Common Java Thread Methods

- `void start()`
  - Initiates thread execution
- `void join()`
  - Waits for a thread to finish
- `abstract void run()`
  - Hook method for user code
- `void sleep(long time)`
  - Sleeps for given time in ms
- `void interrupt()`
  - Post an interrupt request to a Thread
- **`Thread currentThread()`**
  - Object for current thread

public class **Thread**
extends Object
implements Runnable

Summary: Nested Classes | Constants | Ctors | Methods | Inherited Methods | [Expand All]
Added in API level 1

java.lang.Object
↳java.lang.Thread

▸ Known Direct Subclasses
HandlerThread

Class Overview

A `Thread` is a concurrent unit of execution. It has its own call stack for methods being invoked, their arguments and local variables. Each application has at least one thread running when it is started, the main thread, in the main `ThreadGroup`. The runtime keeps its own threads in the system thread group.

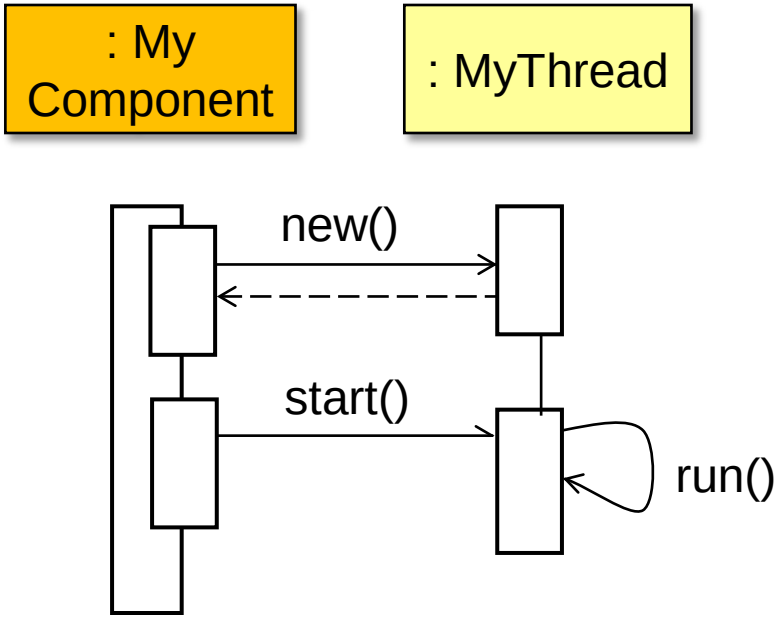
There are two ways to execute code in a new thread. You can either subclass `Thread` and overriding its `run()` method, or construct a new `Thread` and pass a `Runnable` to the constructor. In either case, the `start()` method must be called to actually execute the new `Thread`.

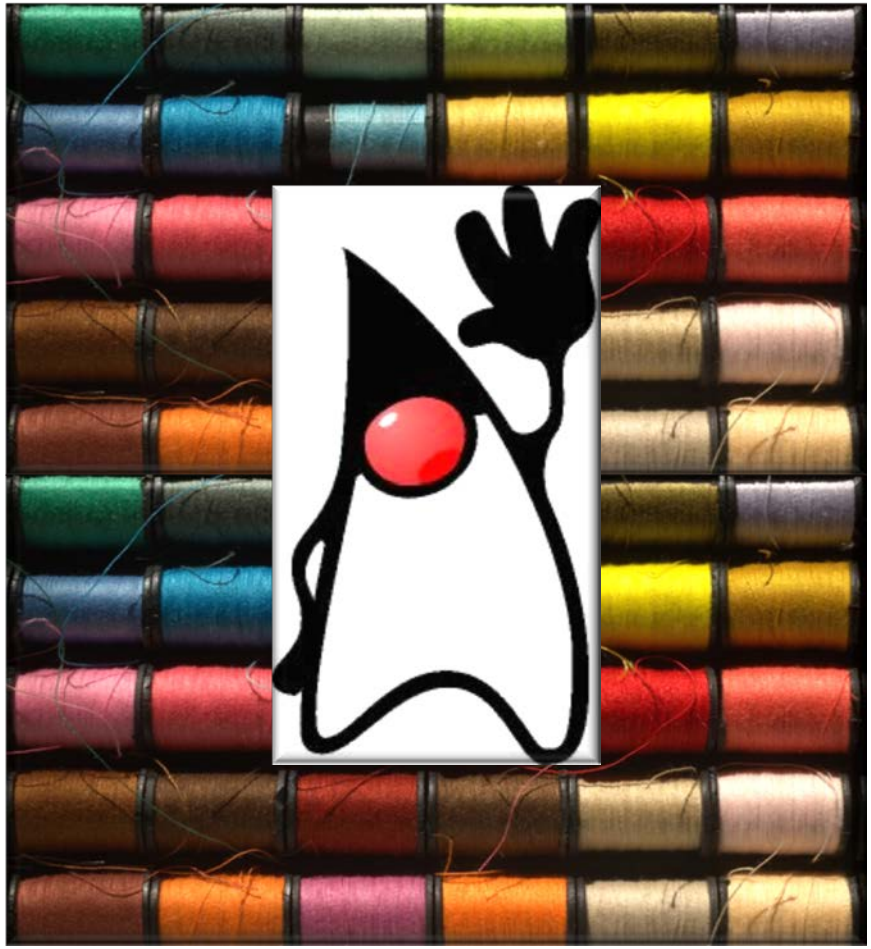
Each `Thread` has an integer priority that affect how the thread is scheduled by the OS. A new thread inherits the priority of its parent. A thread's priority can be set using the `setPriority(int)` method.

Summary

Summary

- Some Android concurrency mechanisms are based on standard Java classes familiar to many developers

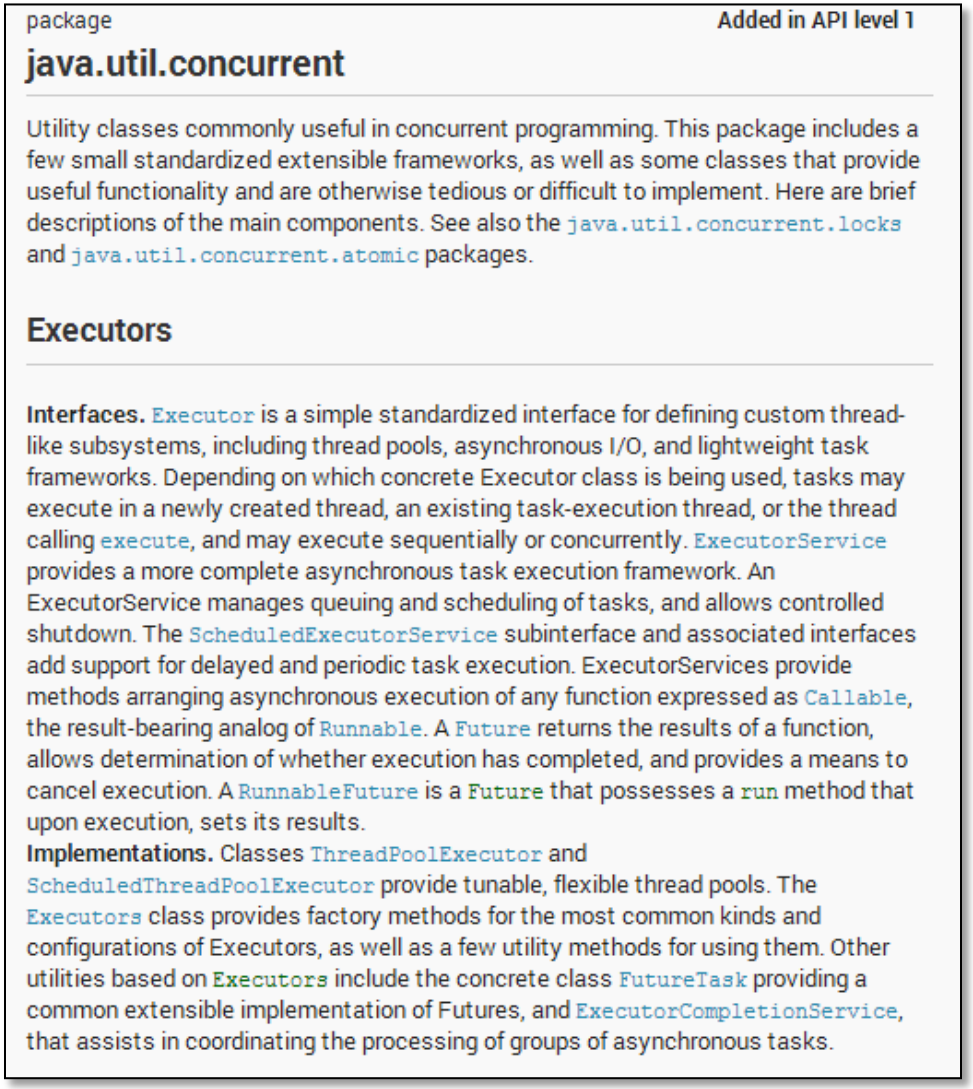




See docs.oracle.com/javase/tutorial/essential/concurrency

Summary

- Some Android concurrency mechanisms are based on standard Java classes familiar to many developers
- Android also supports powerful concurrency mechanisms from the java.util.concurrent package
  - e.g., ThreadPoolExecutor & Future



package — Added in API level 1

java.util.concurrent

Utility classes commonly useful in concurrent programming. This package includes a few small standardized extensible frameworks, as well as some classes that provide useful functionality and are otherwise tedious or difficult to implement. Here are brief descriptions of the main components. See also the `java.util.concurrent.locks` and `java.util.concurrent.atomic` packages.

Executors

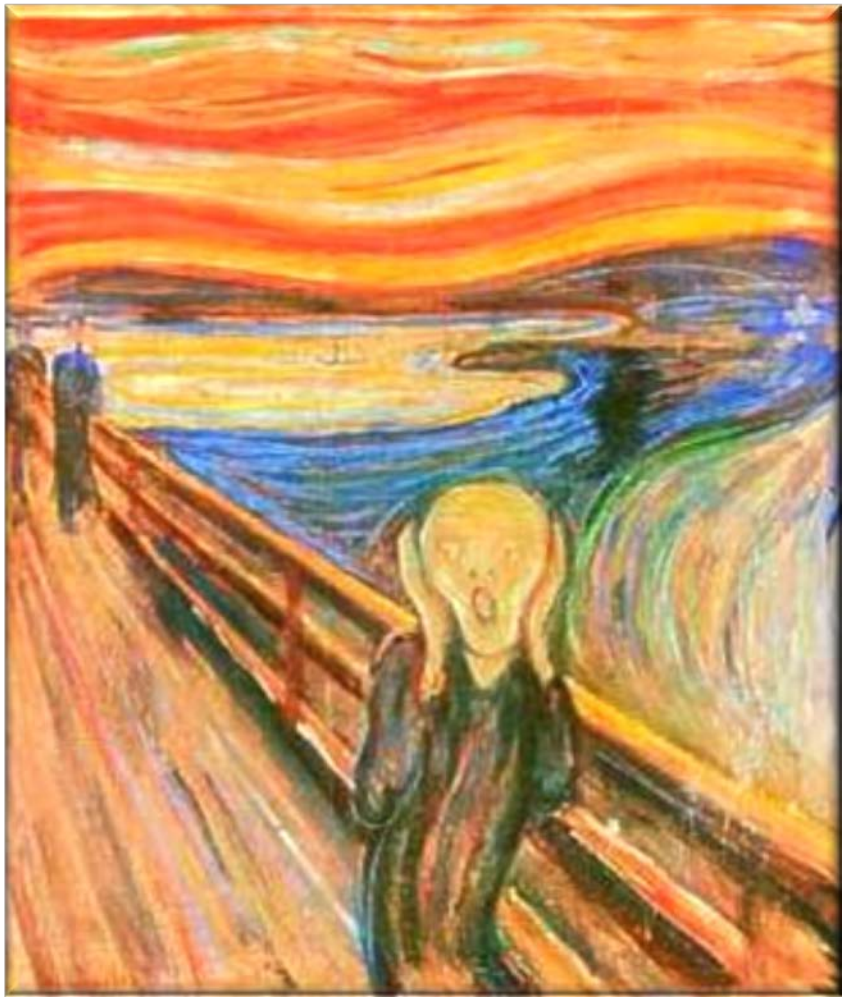
Interfaces. `Executor` is a simple standardized interface for defining custom thread-like subsystems, including thread pools, asynchronous I/O, and lightweight task frameworks. Depending on which concrete Executor class is being used, tasks may execute in a newly created thread, an existing task-execution thread, or the thread calling `execute`, and may execute sequentially or concurrently. `ExecutorService` provides a more complete asynchronous task execution framework. An ExecutorService manages queuing and scheduling of tasks, and allows controlled shutdown. The `ScheduledExecutorService` subinterface and associated interfaces add support for delayed and periodic task execution. ExecutorServices provide methods arranging asynchronous execution of any function expressed as `Callable`, the result-bearing analog of `Runnable`. A `Future` returns the results of a function, allows determination of whether execution has completed, and provides a means to cancel execution. A `RunnableFuture` is a `Future` that possesses a `run` method that upon execution, sets its results.

Implementations. Classes `ThreadPoolExecutor` and `ScheduledThreadPoolExecutor` provide tunable, flexible thread pools. The `Executors` class provides factory methods for the most common kinds and configurations of Executors, as well as a few utility methods for using them. Other utilities based on `Executors` include the concrete class `FutureTask` providing a common extensible implementation of Futures, and `ExecutorCompletionService`, that assists in coordinating the processing of groups of asynchronous tasks.

developer.android.com/reference/java/util/concurrent/package-summary.html

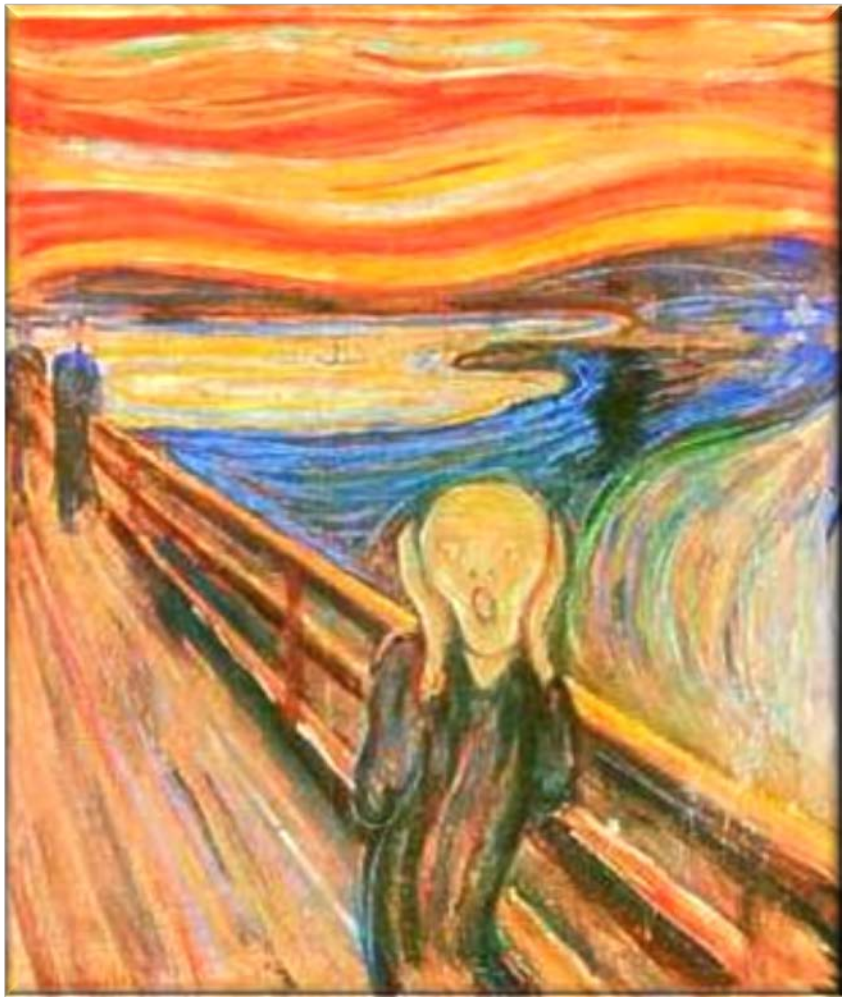
Summary

- Some Android concurrency mechanisms are based on standard Java classes familiar to many developers
- Android also supports powerful concurrency mechanisms from the java.util.concurrent package
- However, writing Android programs using Java threads has some drawbacks



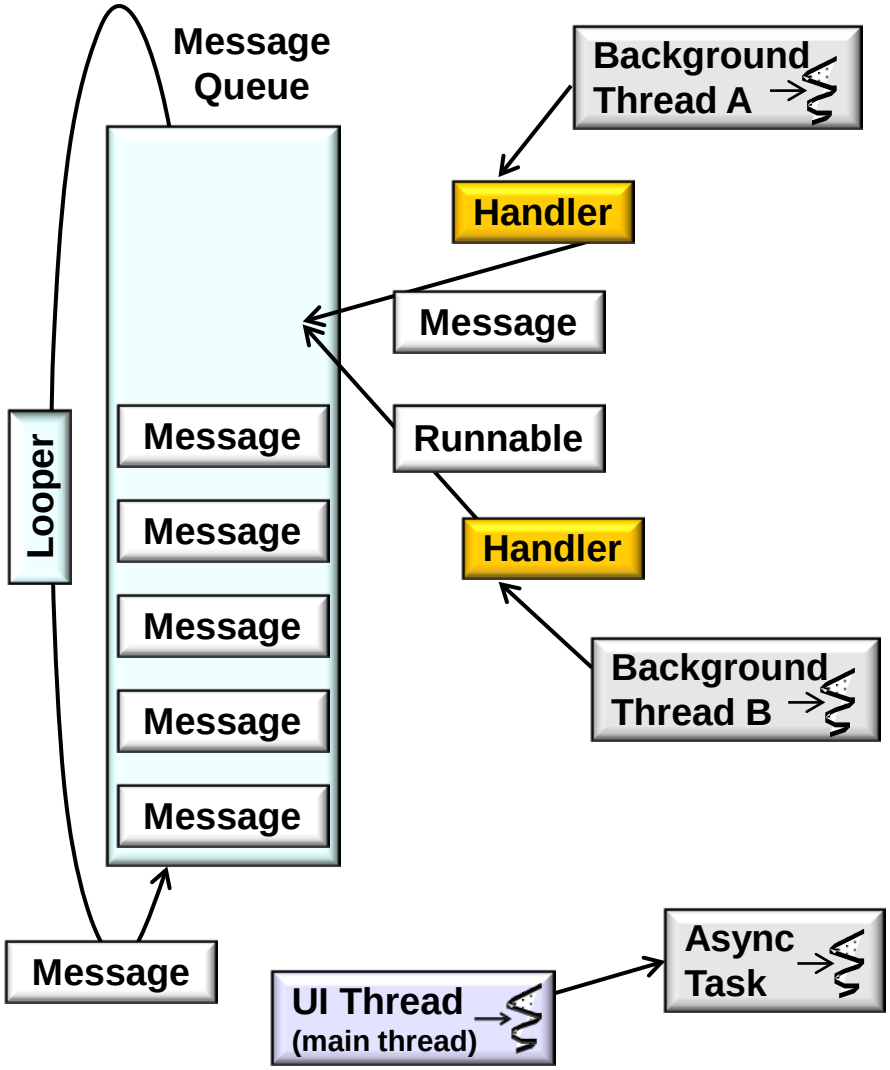
Summary

- Some Android concurrency mechanisms are based on standard Java classes familiar to many developers
- Android also supports powerful concurrency mechanisms from the java.util.concurrent package
- However, writing Android programs using Java threads has some drawbacks
  - e.g., Android doesn't allow background Java threads to access the display



Summary

- Some Android concurrency mechanisms are based on standard Java classes familiar to many developers
- Android also supports powerful concurrency mechanisms from the java.util.concurrent package
- However, writing Android programs using Java threads has some drawbacks
- In practice, many Android apps use its idiomatic concurrency frameworks



See developer.android.com/guide/components/processes-and-threads.html