Learning Objectives in this Part of the Module

• Understand how the *Thread-Specific Storage* pattern is implemented & applied in Android
Thread-Specific Storage POSA2 Synchronization

Implementation

- Implement thread-specific object proxies
  - Mediates access to the underlying thread-specific objects

Definitions:

- Defines a variable for which each thread has its own value
Thread-Specific Storage  POSA2 Synchronization

Implementation

- Implement thread-specific object proxies
- Implement the thread-specific object sets
  - There are two alternatives

Thread-internal Thread-Specific Object Set
Thread-Specific Storage POSA2 Synchronization

**Implementation**

- Implement thread-specific object proxies
- Implement the thread-specific object sets
  - There are two alternatives
Thread-Specific Storage POSA2 Synchronization

Implementation

- Implement thread-specific object proxies
- Implement the thread-specific object sets
  - There are two alternatives
  - Define data structures that map keys & thread ids to thread-specific object sets

Per-thread data structure that maps keys to thread-specific objects
**Implementation**

- Implement thread-specific object proxies
- Implement the thread-specific object sets
  - There are two alternatives
  - Define data structures that map keys & thread ids to thread-specific object sets

*The thread identifier, thread-specific object set, & the proxy cooperate to obtain the correct thread-specific object*
Applying Thread-Specific Storage in Android
Applying Thread-Specific Storage in Android

- Instances of ThreadLocal implement the *Thread-Specific Storage* pattern

*The Java ThreadLocal class doesn’t use a full-blown proxy*

[developer.android.com/reference/java/lang/ThreadLocal.html](http://developer.android.com/reference/java/lang/ThreadLocal.html) has more info
Applying Thread-Specific Storage in Android

- Instances of ThreadLocal implement the *Thread-Specific Storage* pattern

- All threads share the same ThreadLocal object

*Defines a variable for which each thread has its own value*
Applying Thread-Specific Storage in Android

- Instances of ThreadLocal implement the *Thread-Specific Storage* pattern

- All threads share the same ThreadLocal object
- Each thread sees a different value when accessing it, so changes made by one thread don’t affect values of ThreadLocal objects in other threads

**Per-thread map of ThreadLocal instances to values**
Applying Thread-Specific Storage in Android

- Instances of ThreadLocal implement the *Thread-Specific Storage* pattern

The thread identifier, Values map, & the ThreadLocal object cooperate to obtain the correct thread-specific Looper object

- All threads share the same ThreadLocal object
- Each thread sees a different value when accessing it, so changes made by one thread don’t affect values of ThreadLocal objects in other threads
Applying Thread-Specific Storage in Android

- The ThreadLocal.set() method identifies the corresponding Values map based on the current Thread Id & stores the value

```java
public class ThreadLocal<T> {
    ...
    public void set(T value) {
        Thread currentThread = Thread.currentThread();
        Values values = values(currentThread);
        if (values == null) {
            values = initializeValues (currentThread);
        }
        values.put(this, value);
    }
    ...
}
```

Note there’s no synchronization involved at all!

[libcore/luni/src/main/java/java/lang/ThreadLocal.java](libcore/luni/src/main/java/java/lang/ThreadLocal.java) has the source code
Applying Thread-Specific Storage in Android

- The `ThreadLocal.set()` method identifies the corresponding Values map based on the current Thread Id & stores the value.

- The `ThreadLocal.get()` method does the same thing, but returns the thread-specific object.

```java
public class ThreadLocal<T> {
...
    public T get() {
        Thread currentThread = Thread.currentThread();
        Values values = values(currentThread);
        if (values != null) {
            Object[] table = values.table;
            int index = hash & values.mask;
            if (this.reference == table[index]) {
                return (T) table[index + 1];
            }
            table[index] = this.reference;
            return this.reference;
        }
    }
    ...
}
```

*Note there’s no synchronization involved at all!*

`libcore/luni/src/main/java/java/lang/ThreadLocal.java` has the source code.
Applying Thread-Specific Storage in Android

- The Looper classes use a ThreadLocal object to ensure only one Looper is created per Thread

```java
public class Looper {
    ...
    static final static final ThreadLocal<Looper>
        sThreadLocal = new ThreadLocal<Looper>();
    ...
    private static void prepare() {
        if (sThreadLocal.get() != null)
            throw new RuntimeException("Only one Looper may be created per thread");
        sThreadLocal.set(new Looper(quitAllowed));
    }
    ...
}
```

*Thread-Specific Storage doesn’t incur locking overhead on each object access*

[developer.android.com/reference/java/lang/ThreadLocal.html](http://developer.android.com/reference/java/lang/ThreadLocal.html) has more info
Applying Thread-Specific Storage in Android

- The Looper classes uses a ThreadLocal object to ensure only one Looper is created per Thread
- The myLooper() method returns the thread-specific Looper object, which is used in various others methods

```java
public class Looper {
    ...  
    final MessageQueue mQueue;

    public static Looper myLooper() {
        return sThreadLocal.get();
    }

    public static void loop() {
        final Looper me = myLooper();
        if (me == null)
            throw new RuntimeException("No Looper; Looper. prepare() wasn't called on this thread.");
        final MessageQueue queue = me.mQueue;
        ...  
```
Applying Thread-Specific Storage in Android

- The Looper classes uses a ThreadLocal object to ensure only one Looper is created per Thread
- The myLooper() method returns the thread-specific Looper object, which is used in various others methods

```java
public class Looper {
    ...
    final MessageQueue mQueue;

    public static Looper myLooper() {
        return sThreadLocal.get();
    }

    public static void loop() {
        final Looper me = myLooper();
        if (me == null)
            throw new RuntimeException("No Looper; Looper.
            prepare() wasn't called on this thread.");
        final MessageQueue queue = me.mQueue;
        ...
    }
```
Applying Thread-Specific Storage in Android

- The Looper classes uses a ThreadLocal object to ensure only one Looper is created per Thread.

- The myLooper() method returns the thread-specific Looper object, which is used in various others methods.

```java
public class Looper {
    ...
    final MessageQueue mQueue;

    public static Looper myLooper() {
        return sThreadLocal.get();
    }

    public static void loop() {
        final Looper me = myLooper();
        if (me == null)
            throw new RuntimeException("No Looper; Looper.
                prepare() wasn't called on this thread.");
        final MessageQueue queue = me.mQueue;
        ...
    }
}
```

*Cache the thread-specific Looper object in a field*
Applying Thread-Specific Storage in Android

- The Looper classes uses a ThreadLocal object to ensure only one Looper is created per Thread
- The myLooper() method returns the thread-specific Looper object, which is used in various others methods

```java
public class Looper {
    ...
    final MessageQueue mQueue;

    public static Looper myLooper() {
        return sThreadLocal.get();
    }

    public static void loop() {
        final Looper me = myLooper();
        if (me == null)
            throw new RuntimeException("No Looper; Looper. prepare() wasn't called on this thread.");
        final MessageQueue queue = me.mQueue;
        ...
    }
}
```

Cache Looper instance data from the thread-specific Looper object
Applying Thread-Specific Storage in Android

- The Looper classes uses a ThreadLocal object to ensure only one Looper is created per Thread.
- The myLooper() method returns the thread-specific Looper object, which is used in various others methods.
- The Handler constructor also uses myLooper() to connect a Handler the Thread where it’s created.

```java
public class Handler {
    ...
    public Handler() {
        mLooper = Looper.myLooper();
        if (mLooper == null)
            throw new RuntimeException("Can't create handler inside thread that has not called Looper.prepare()");
        mQueue = mLooper.mQueue;
        mCallback = null;
    }
    ...
```

[frameworks/base/core/java/android/os/Handler.java](frameworks/base/core/java/android/os/Handler.java) has the source code
• Android implements the *Thread-Specific Storage* pattern via the Java ThreadLocal class.

```java
class ThreadLocal<T> {
    ...
    public void set(T value) {
        ...
    }
    public T get() {
        ...
    }
}
```

See [javarevisited.blogspot.com/2012/05/how-to-use-threadlocal-in-java-benefits.html](http://javarevisited.blogspot.com/2012/05/how-to-use-threadlocal-in-java-benefits.html)
Summary

- Android implements the *Thread-Specific Storage* pattern via the Java ThreadLocal class

```java
frameworks/base/core/java/android/animation/ValueAnimator.java
frameworks/base/core/java/android/app/ActivityThread.java
frameworks/base/core/java/android/os/Looper.java
frameworks/base/core/java/android/os/Handler.java
frameworks/base/core/java/android/os/StrictMode.java
frameworks/base/core/java/android/view/FocusFinder.java
frameworks/base/core/java/android/view/HardwareRenderer.java
frameworks/base/core/java/android/view/View.java
frameworks/base/core/java/android/view/ViewRootImpl.java
frameworks/base/core/java/android/view/ViewDebug.java
packages/apps/Browser/src/com/android/browser/browser/Tab.java
packages/apps/Gallery2/src/com/android/gallery3d/ui/BasicTexture.java
...
```

Java ThreadLocal is used throughout Android
Summary

- Android implements the *Thread-Specific Storage* pattern via the Java `ThreadLocal` class.
- Android uses `ThreadLocal` to ensure a Thread has a single `Looper`.
Summary

• Android implements the *Thread-Specific Storage* pattern via the Java `ThreadLocal` class

• Android uses `ThreadLocal` to ensure a Thread has a single `Looper`

• It’s also used in the constructor of `Handler`

```java
public class Handler {
    ...

    public Handler() {
        mLooper = Looper.myLooper();
        if (mLooper == null)
            throw new RuntimeException(
                "Can't create handler inside thread that has not called Looper.
                prepare()");

        mQueue = mLooper.mQueue;
        mCallback = null;
    }
    ...
```
End of Android & Java Concurrency: The Thread-Specific Storage Pattern (Part 2)