

Threads In MIDP Applications

- What is a Thread?
- How do they work in CLDC/MIDP?
- When are they used?

Author: Chris Harcourt

What is a thread?

- A thread is a chain of execution within a program.
- A thread can do only one thing at any given time.
 - Updating the display.
 - Handling user input events.
 - Connecting to a network.
 - Nothing! - called *waiting* or *sleeping*.
- A thread has it's own copies of **local** variables for the methods it executes.

Threads in CLDC/MIDP

- CLDC supports multi-threading
- Two approach to writing threads:
 - `java.lang.Runnable` (interface)
 - Classes implement this to have (some) code execute in a thread.
 - Defines one public method : `run()`. This contains code which is run in the thread.
 - `java.lang.Thread` (implements `Runnable`)
 - Represents a thread within the system.
 - Has methods for manipulating thread (start, sleep, etc)
 - Subset of the Java SE class.

Threads in CLDC/MIDP (2)

- Important methods on `java.lang.Thread`:
 - `start()` - starts the thread's execution
 - `interrupt()` - interrupts execution (only in CLDC-1.1)
 - `sleep(long)` - pauses execution for specified time
 - `isAlive()` - tests if the thread is “alive”
 - `set/getPriority()` - manipulate priority (higher priority threads are executed in preference to lower)
- See the MIDP API documentation for more details

Using java.lang.Runnable

Class must implement Runnable

```
class MyRunnable implements Runnable {  
    public void run() {  
        ... do something ...  
        ... your code ...  
    }  
}
```

Somewhere else (e.g. Midlet) creates/starts the thread

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

Example 1 (SimpleRunnable)

- The Midlet class itself implement Runnable.
- Thread is created passing “this” in constructor.
- Execution of thread prints 1-10 to console.

Using java.lang.Thread

Class extends Thread

```
class MyThread extends Thread {  
    public void run() {  
        ... do something ...  
        ... your code ...  
    }  
}
```

Somewhere else (e.g. Midlet) creates/starts the thread

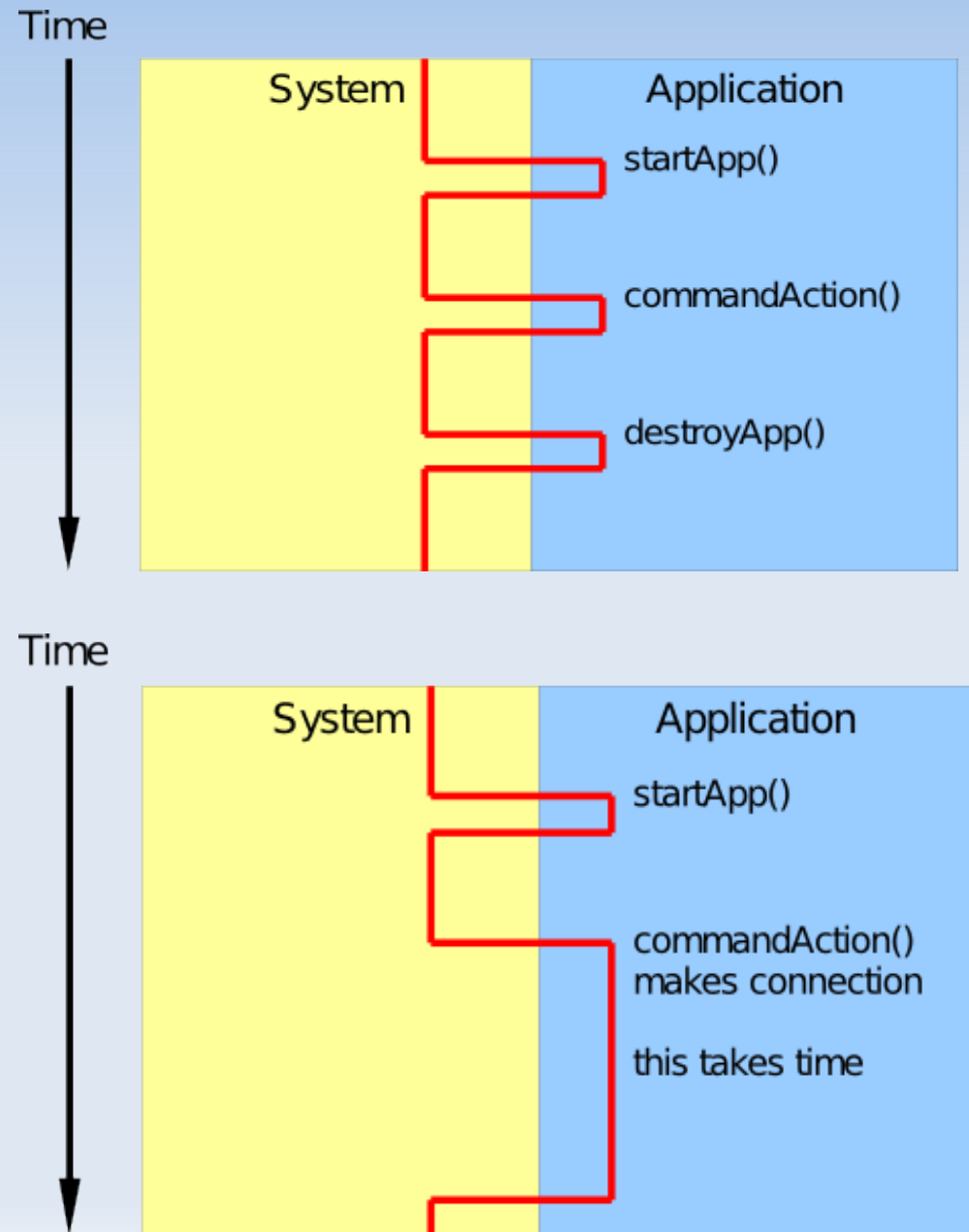
```
...  
Thread myThread = new MyThread();  
myThread.start();  
...
```

Example 2 (SimpleThread)

- Midlet contains an inner class which extends Thread
- No need to pass anything in thread's constructor.
- Thread body prints 1 – 10 to console.
- This time using sleep() to pause the thread between each number.

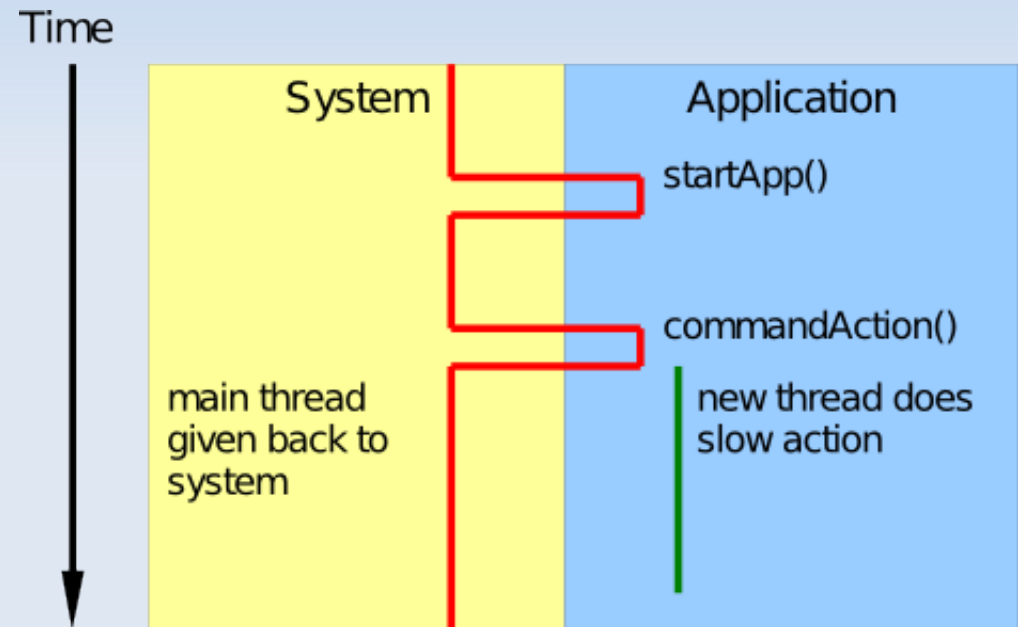
Why Use Threads?

- *User Experience*
- `commandAction()` runs in the system's thread
- Actions which take time will “block” this thread, which results in a “hung” user interface.
- To the user it appears that the application has frozen!
- Not what we want...



Why Use Threads? (2)

- Solution: perform slow (blocking) actions in a separate thread.
- This means the UI can be used while another thread performs the slow action in the background.
- This technique should be used for *anything* which could take longer than a second or so.



Synchronisation

- When multiple threads could access the same data care must be taken to avoid corruption.
- This is done using the “synchronized” statement.
 - Ensures only one thread can access a resource at a time.
- Exactly the same as in Java SE.
- Lucky for us, most of MIDP is already “thread safe”.
- But, if your application is multi-threaded you *must* consider synchronisation issues.
- Care must be taken to avoid “deadlocks”.

Synchronisation (2)

- Synchronisation applied to an object:

```
synchronized (anObject) {  
    ... some synchronised code ...  
    ... only one thread can access anObject ...  
}
```

- Synchronisation applied to a method:

```
public synchronized void doSomething() {  
    ... some synchronised code ...  
    ... only one thread can access doSomething() ...  
}
```

General Considerations

- Resource constraints
 - Don't expect to use lots of concurrent threads.
 - Clean-up unused threads.
- Keep it simple – debugging threading issues is hard.

Golden Rule:

*If you didn't create the thread yourself you are not
allowed to block or sleep it.
(don't hijack the system thread!)*