CS2063 Intro. Mobile Application Development

Background Tasks in Android

Thread

Background Tasks in Android

The Android Platform supports Background Processing in 4 different ways:

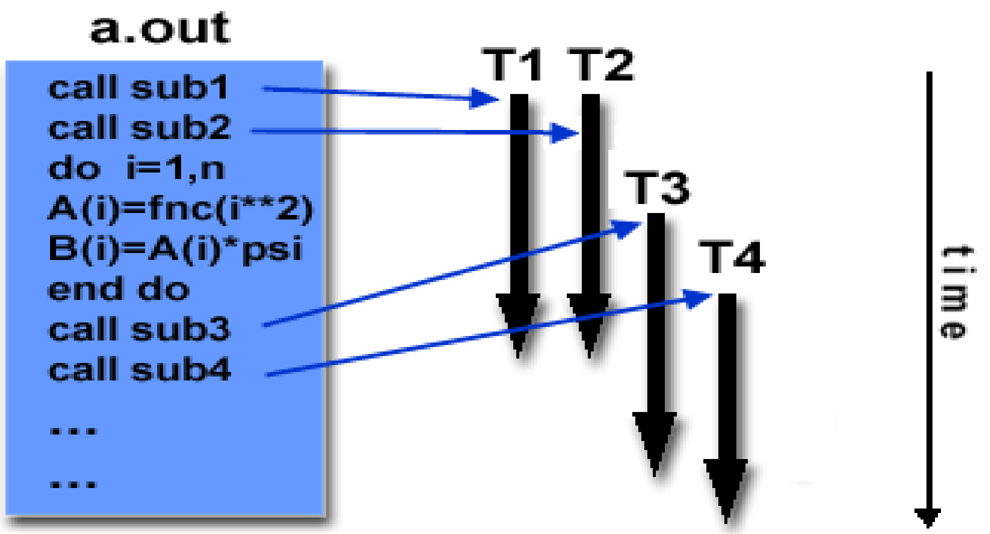
- **Threads**: Android supports the usage of the Threads class to perform **asynchronous processing**.
- **Handler**: The Handler class can update the user interface. A Handler provides methods for receiving instances of the Message or Runnable class.
- **AsyncTask**: Is a special class for Android development that encapsulate background processing and facilitates the communication and updating of the application’s UI.
- **Service**: is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if application is destroyed.

An Overview of Threads In Java

A **thread** is a concurrent unit of execution.

- Threads share **process's resource** but are able to execute **independently**.
- Each thread has a **call stack** for methods being invoked.
- A **VM** may run several threads in **parallel**.
- True parallelism for **multi-core CPU**.
- A **VM** has at least the **main thread** running when it is started.

Threads Model



```
call sub1
call sub2
do  i=1,n
A(i)=fnc(i**2)
B(i)=A(i)*psi
end do
call sub3
call sub4
...
...
```

An Overview of Threads In Java (Cont.)

- **Multithreaded programming challenges** include:
  - Dividing work load.
  - Overriding data.
  - Data dependency.
  - Deadlock.
  - Testing and debugging.

An Overview of Threads In Java (Cont.)

Why to use threads?

- **Multi-thread programming is hard, so why to use it?**

  If the execution time of the **main thread** is higher than **5 s**, then the **OS** displays an error message **(ANR).**

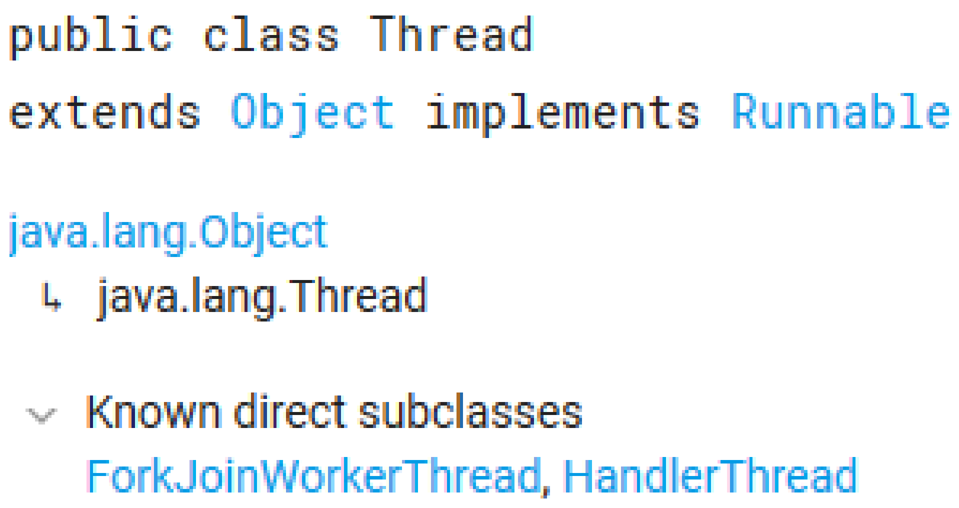
- Slow tasks (like file downloading), cannot run in the main thread; so, in this case you ***must use multiple threads***.

- In a multi-core CPU, multiple threads can truly run in parallel.

How to use multi-tread?

- classical Thread programming.
- **However**, special care must be taken as only **main thread** can update the **UI**.

Thread



```
public class Thread
extends Object implements Runnable
```

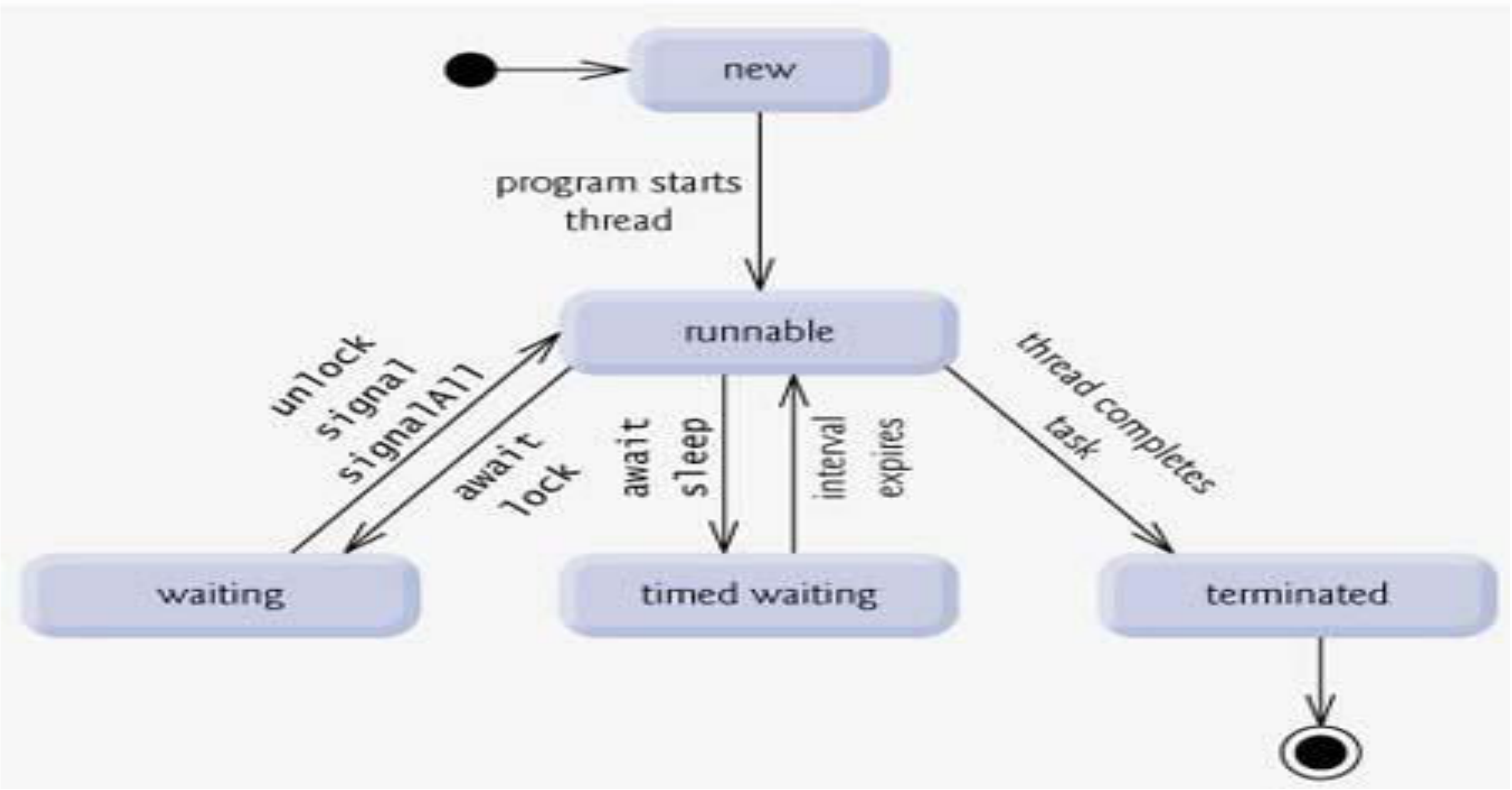
java.lang.Object
  ↳ java.lang.Thread

⌄ Known direct subclasses
ForkJoinWorkerThread, HandlerThread

The **Thread class** defines several methods that help manage threads.

Method	Meaning
getName()	Obtain thread’s name.
getPriority()	Obtain thread’s priority.
isAlive()	Determine if a thread is still running.
join()	Wait for a thread to terminate.
run()	Entry point for the thread.
sleep()	Suspend a thread for a period of time.

Life Cycle of a Thread



How to create a thread

- **Creating a Thread:**
  - You can **implement** the **Runnable interface**.

```
Runnable myRunnable1 = new MyRunnableClass();

Thread t1 = new Thread(myRunnable1);

t1.start();
```

  - You can **extend** the **Thread class**.
    - Create a new class that *extends* **Thread** and override its **run()** method.

```
MyThread t = new MyThread();

t.start();
```

- **In both cases**, the **start() method** must be called to actually execute the new Thread.

Max and min numbers?

```
import java.util.Arrays;
import java.util.Collections;
import java.io.*;
public class MinMax extends Thread {
    static Integer[] numbers = { 8, 2, 7, 1, 4, 9, 5};
    int i;
     MinMax(int i) {
             this.i = i;
             this.start();
           }
           public void run() {
           if(i==0) {
               int min = (int) Collections.min( Arrays.asList(numbers) );
               System.out.println("Min number: " + min);
        }
       else {
               int max = (int) Collections.max( Arrays.asList(numbers) );
               System.out.println("Max number: " + max);
         }
} // run
    public static void main(String args[])
    {
              MinMax min = new MinMax(0);
              MinMax max = new MinMax(1);
               try {
                  min.join();
                    max.join();
               } catch(Exception e){ }
              System.out.println("done :");
    } // end main()
} // end class
```

```
class PrintJava
{
        public static void main(String args[])
        {

                        Q q = new Q();

                        new Producer( q );

                        new Consumer( q );

                        System.out.println("Press Control-C to  stop.");
        }
}
```

```
class Producer implements Runnable
{
        Q q;
        Producer(Q q)
        {
                this.q = q;
                new Thread(this, "Producer").start();
        }

        public void run()
        {
                int i = 0;
                while(true)
                {
                q.put(i++);
                }
        }
}
```

```
class Consumer implements Runnable
{
        Q q;

        Consumer(Q q)
        {
                this.q = q;
                new Thread(this, "Consumer").start();
        }

        public void run()
        {
                while(true)
                {
                        q.get();
                }
        }
}
```

```
class Q
{
        int n;
        boolean valueSet = false;

        synchronized int get()
        {
            if(!valueSet)
                  try
                  {
                        wait();
                  }
                  catch(InterruptedException e)
                  {
                    System.out.println(" InterruptedException caught");
                  }
                  System.out.println("Got: " + n);
                  valueSet = false;
                  notify();
                  return n;
      }
```

```
    synchronized void put(int n)
    {

        if(valueSet)
        try
        {
            wait();
        }
        catch(InterruptedException e)
        {
            System.out.println("InterruptedException caught");
        }
        this.n = n;
        valueSet = true;
        System.out.println("Put: " + n);
        notify();
    }
}
```

Example-1: define a class to be a subclass of Thread.

```
class PrimeThread extends Thread {
            long minPrime;
            PrimeThread(long minPrime) {
            this.minPrime = minPrime;
        }

        public void run() {
            // compute primes larger than minPrime
             . . .
        }
}
```

- **The following code would then create a thread and start it running:**

```
PrimeThread p = new PrimeThread(143);
 p.start();
```

Example-2: define a class that implements the Runnable interface.

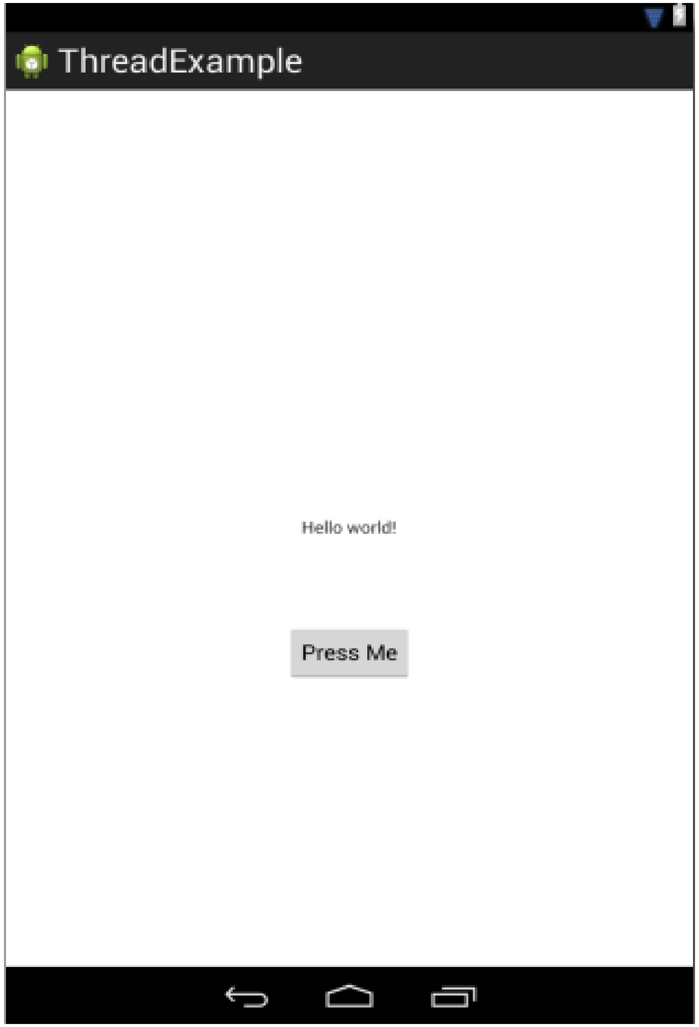
```
class PrimeRun implements Runnable {
          long minPrime;
          PrimeRun(long minPrime) {
          this.minPrime = minPrime;
       }

       public void run() {
          // compute primes larger than minPrime
           . . .
       }
}
```

- **The following code would then create a thread and start it running:**

```
    PrimeRun p = new PrimeRun(143);
     new Thread(p).start();
```

Example-3 : A Basic Threading in Android

- The **first step** will be to highlight the risks involved in not performing **time-consuming** tasks in a **separate thread** from the **main thread**.



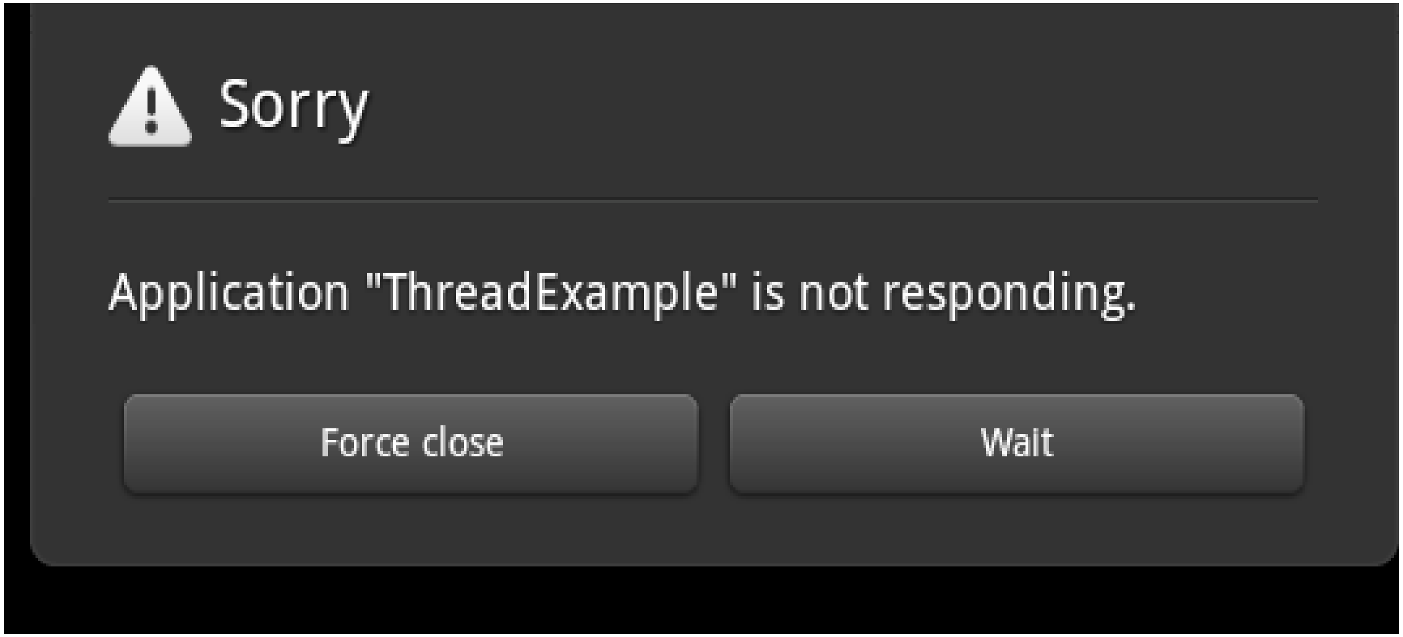
ThreadExample

Hello world!

Example -1: without Threads

```
…
public void buttonClick(View view)
    {
        long endTime = System.currentTimeMillis() + 20*1000;

        while (System.currentTimeMillis() < endTime) {
          synchronized (this) {
            try {
              wait(endTime - System.currentTimeMillis());
            } catch (Exception e) {   }
          }
        }
        TextView myTextView = (TextView)findViewById(R.id.myTextView);
        myTextView.setText("Button Pressed");
    }
…
```

To avoid ANR



Sorry

Application "ThreadExample" is not responding.

Example -2: Using Threads

```
…
public void buttonClick(View view)
        {
            Runnable runnable = new Runnable() {
                public void run() {
                    long endTime = System.currentTimeMillis() + 20*1000;

                    while (System.currentTimeMillis() < endTime) {
                        synchronized (this) {
                            try {
                                wait(endTime - System.currentTimeMillis());
                            } catch (Exception e) { }
                        }
                    }
                }
            };
            Thread mythread = new Thread(runnable);
            mythread.start();
        }
```

Reference

- Background tasks in Android
- Threads in Java
- Android – Threads