



**android**

# Android

Threads, Services, AIDL



1. Mark MURPHY, *Beginning Android 2*, Apress, 2010
  - Capitolul 30
2. Lesson: Concurrency,  
<http://java.sun.com/docs/books/tutorial/essential/concurrency/>

# Contents



- Threads
  - User
  - Kernel
- Services
  - Envents
  - Events loop
  - Start/Stop
  - Communication
- AIDL



# Processing in Activities

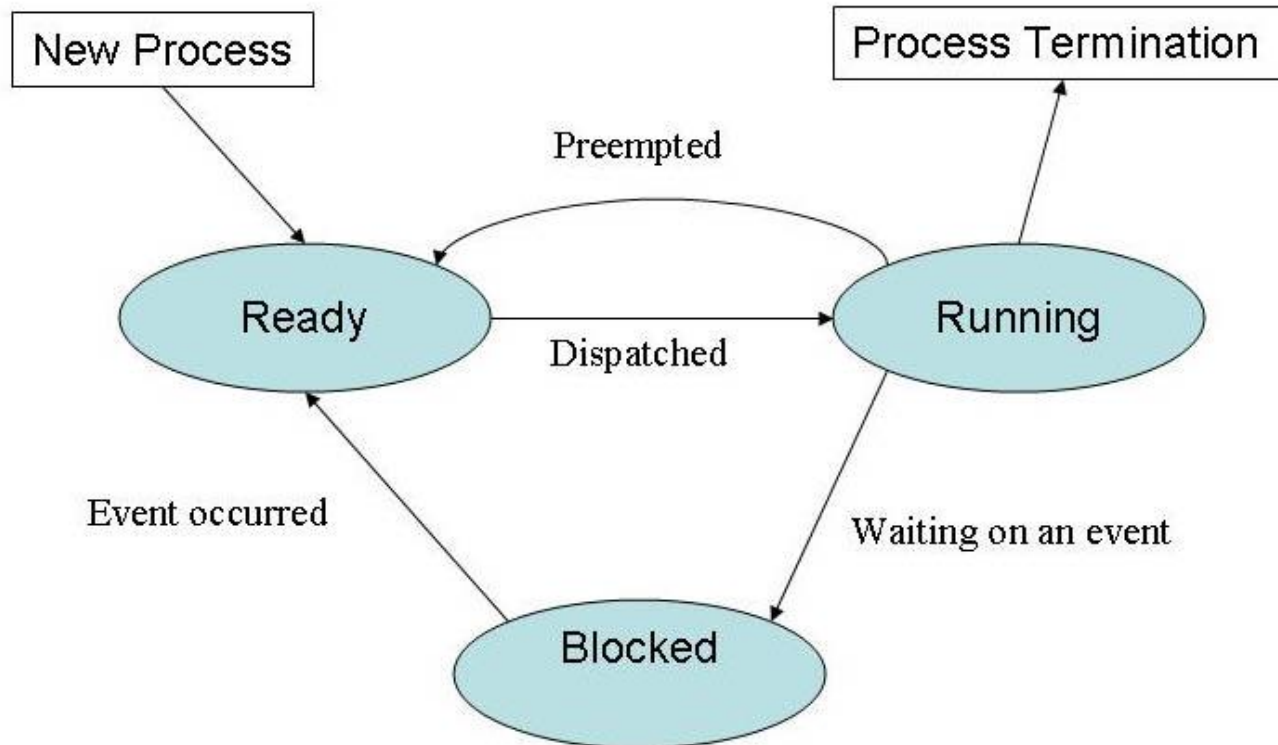
- Processing is done in
  - Activity's methods
    - *onCreate (...)*
    - *onStart (...)*
    - ...
  - Observer's methods
    - *onClick (...)*
- Not a lot of processing
  - Avoiding *Not Responding*
- Solution
  - Threads
  - Services



# Multitasking



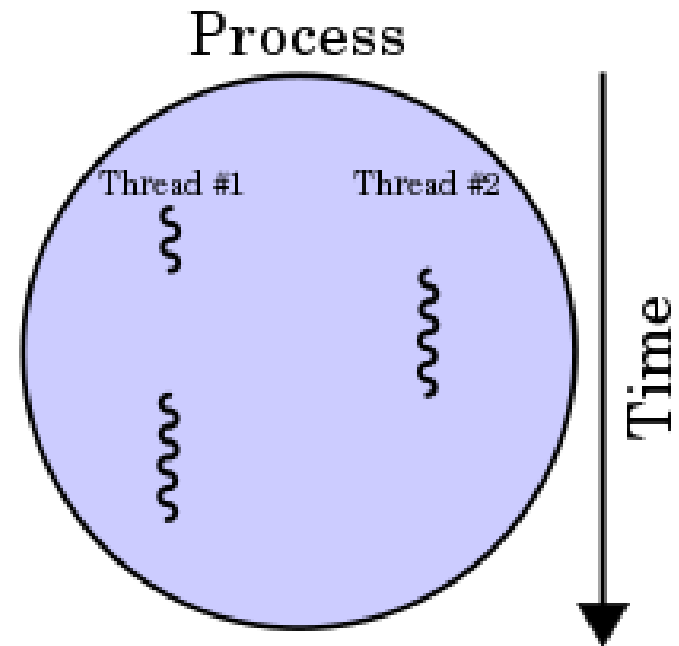
# States of processes



# Threads



- Splitting the program
  - More processing ways
    - *More main() methods*
  - Sharing memory
    - *Variables are shared*





- **Class Thread**
  - **override** *run ()* methos
- ***Runnable Interface***
  - **Implement** *run ()* method





# Class Thread



```
class MyThread extends Thread
{
    // ... constructor, methods etc.
    public void run ()
    {
        // the code of the thread
    }
}
```

# Usage of the class MyThread



```
class MyThread extends Thread
{
    // ... constructors, methods etc.
    public void run ()
    {
        // the code of the thread
    }
}
```

```
MyThread myThread = new MyThread(...);
myThread.start (); <- NOT run()!!!
```

# Runnable Interface



```
class MyThread [extends ...] implements Runnable, ...
{
    // ... constructors, methods etc.
    public void run ()
    {
        // the code of the thread
    }
}
```

# Usage of MyThread



```
class MyThread [extends ...] implements Runnable, ...  
{  
    // ... constructors, methods etc.  
    public void run ()  
    {  
        // the code of the thread  
    }  
}
```

```
Thread myThread = new Thread (new MyThread());  
myThread.start (); <- NOT run()!!!
```

# Runnable VS Thread

---



# Runnable VS Thread

---



## ***Runnable***

- Interface
- The object can extend any other class
- Just implement *Runnable* interface
- More flexible

## **Thread**

- Class
- Object has to extend **Thread**
- Less flexible

# Stopping a thread

---



*Only when run ()* method finishes its execution

# Services



- Android component
- Special for processing
- Runs in the *background*
- Process
  - More stable (in time)





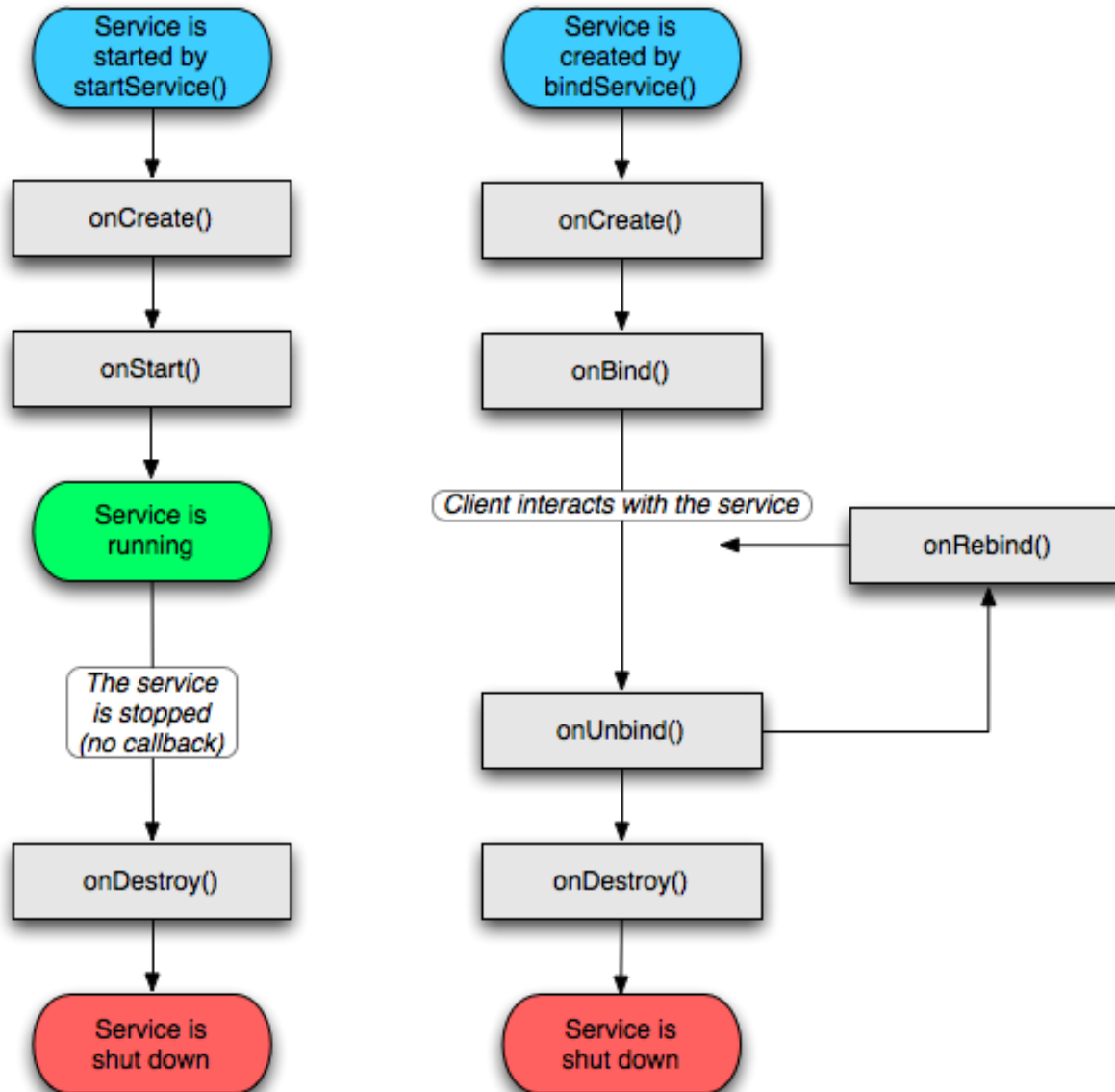
# Service types

---



- Foreground
  - Noticeable to the user
- Background
- Bound
  - Client-server interface
  - Uses AIDL
  - IPC (inter-process communication)

# Events



# Service implementation



- Extends **Service** class
  - Simple service
    - **void** *onCreate* ();
    - **void** *onStart* (**Intent** intent, **int** startID);
    - **void** *onDestroy* ();
  - Using AIDL
    - **void** *onBind* (**Intent** intent);
    - **void** *onUnbind*();



# Service implementation



1. Extends **Service** class
  - Methods implementation
  - **Creating threads**
2. Manifest declaration
3. Starting / Stopping
  - *startService (...)*
  - *stopService (...)*



# Exemple


---



- Print a prime number per second
  - A service is not started automatically
  - A service is not stopped automatically

# PrimeNumbers - Service



```
class PrimeNumbers: Service() {  
  
    lateinit var calculatorThread: PrimeNumbersCalculator;  
  
    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {  
        calculatorThread = PrimeNumbersCalculator()  
        calculatorThread.start()  
  
        return super.onStartCommand(intent, flags, startId)  
    }  
  
    override fun onDestroy() {  
        super.onDestroy()  
        calculatorThread.quit()  
    }  
  
    >  override fun onBind(p0: Intent?): IBinder? {...}  
}
```

# NumerePrime – Thread efectiv



```
class PrimeNumbersCalculator: Thread() {
    var quit: Boolean = false;

    override fun run() {
        super.run()
        val n = 2.0;
        while(!quit) {
            if (isPrime(n)) {
                println("Prime number: $n")

                try {
                    Thread.sleep( millis: 1000)
                } catch (_: Exception) { }
            }
        }
    }

    fun isPrime(n: Double): Boolean {
        var result = true
        var max: Double = round(sqrt(n))
        for (i in 2 ≤ .. ≤ max.toInt()) {
            if (result) {
                if ((n%i).toInt() == 0) {
                    result = false
                }
            }
        }
        return result
    }

    fun quit() {
        quit = true;
    }
}
```

# Starting the service

---



```
Intent starter = new Intent (context,  
    PrimeNumbersCalculator.class);  
context.startService (starter);
```



# Stopping the service

---



```
context.stopService (starter);
```

```
stopSelf ();
```

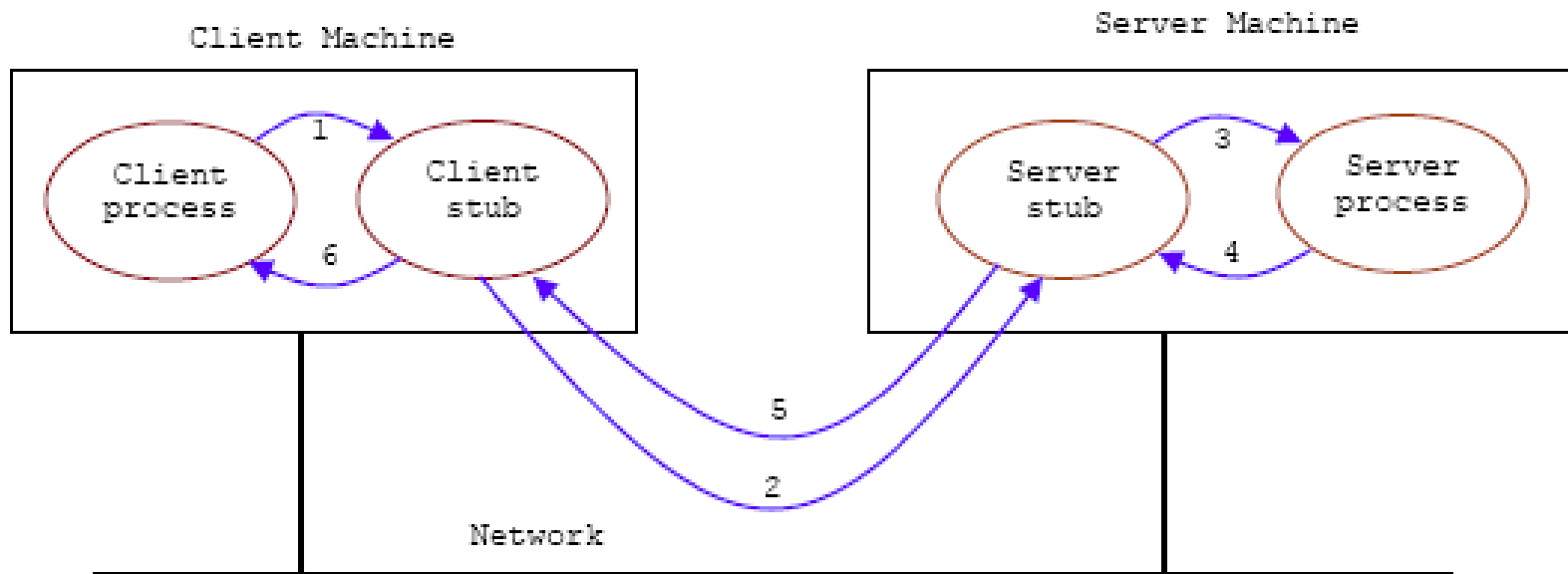


- Android Interface Definition Language
  - **RPC** of Android
- Connection between activity and service



I gotta take this, it's a remote procedure call

# Remote Procedure Call



- (1) and (3) are ordinary procedure calls.
- (2) and (5) are messages.
- (4) and (6) are ordinary procedure returns.



- Java-like syntax
- Identical declarations
- Limited data types
  - primitive (int, long, float, double, char, boolean)
  - String
  - List\*
  - Map\*
  - Special types

\* Must only contain AIDL data types

# Example ... IMoviesService.aidl

---



```
package pdm.movies;
```

```
interface IMoviesService
```

```
{
```

```
    int moviesNumber ();
```

```
    String movieTitle (int numar);
```

```
    String movieDirector (int numar);
```

```
}
```

# Create service with binder



```
class LocalService : Service() {
    // Binder given to clients.
    private val binder = LocalBinder()

    // Random number generator.
    private val mGenerator = Random()

    /** Method for clients. */
    val randomNumber: Int
        get() = mGenerator.nextInt(100)

    /**
     * Class used for the client Binder. Because we know this service always
     * runs in the same process as its clients, we don't need to deal with IPC.
     */
    inner class LocalBinder : Binder() {
        // Return this instance of LocalService so clients can call public methods.
        fun getService(): LocalService = this@LocalService
    }

    override fun onBind(intent: Intent): IBinder {
        return binder
    }
}
```

# Connecting to a Binder Service



```
class BindingActivity : Activity() {
    private lateinit var mService: LocalService
    private var mBound: Boolean = false

    /** Defines callbacks for service binding, passed to bindService(). */
    private val connection = object : ServiceConnection {

        override fun onServiceConnected(className: ComponentName, service: IBinder) {
            // We've bound to LocalService, cast the IBinder and get LocalService instance.
            val binder = service as LocalService.LocalBinder
            mService = binder.getService()
            mBound = true
        }

        override fun onServiceDisconnected(arg0: ComponentName) {
            mBound = false
        }
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main)
    }

    override fun onStart() {
        super.onStart()
        // Bind to LocalService.
        Intent(this, LocalService::class.java).also { intent ->
            bindService(intent, connection, Context.BIND_AUTO_CREATE)
        }
    }

    override fun onStop() {
        super.onStop()
        unbindService(connection)
        mBound = false
    }
}
```

# Connecting to a Binder Service



```
class BindingActivity : Activity() {
    private lateinit var mService: LocalService
    private var mBound: Boolean = false

    /** Defines callbacks for service binding, passed to bindService(). */
    private val connection = object : ServiceConnection {

        /** Called when a button is clicked (the button in the layout file attaches to
         * this method with the android:onClick attribute). */
        fun onClick(v: View) {
            if (mBound) {
                // Call a method from the LocalService.
                // However, if this call is something that might hang, then put this request
                // in a separate thread to avoid slowing down the activity performance.
                val num: Int = mService.randomNumber
                Toast.makeText(this, "number: $num", Toast.LENGTH_SHORT).show()
            }
        }
    }

    override fun onStart() {
        super.onStart()
        // Bind to LocalService.
        Intent(this, LocalService::class.java).also { intent ->
            bindService(intent, connection, Context.BIND_AUTO_CREATE)
        }
    }

    override fun onStop() {
        super.onStop()
        unbindService(connection)
        mBound = false
    }
}
```



# Questions

---

