Android View Bindings using Kotlin

software inside

Prior Knowledge needed on

HTL Villach
Future Inside

- Android Programming using Java
- its loosely coupled components architecture
- their life-cycles
- views for GUIs
- based on DAOs using either local data bases or web services

software inside

Example for a Button, TextView, Life Cycle

```kotlin
package com.examples.che.myfirstkotlinapp

import ...

class MainActivity : AppCompatActivity() {
    val TAG = "StateChange";
    var i : Int = 0;

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

    fun onClickMe(view: View) {
        val v: TextView = findViewById(R.id.tvMessage)
        if (v != null) {   // null reference possible when v has not been created yet
            i++
            v.text = "pressed $i times"
        }
    }

    override fun onStart() {...}
    override fun onResume() {...}
    override fun onPause() {...}
    override fun onStop() {...}
    override fun onRestart() {...}
    override fun onDestroy() {...}
```

as known:
- find view by Id
- all resources are compiled into class R
- and its corresponding sub classes R.layout, R.Id, ..
- up to Android Studio 3.6 this was the one and only option with Java

With View Bindings ..

HTL Villach
Future Inside

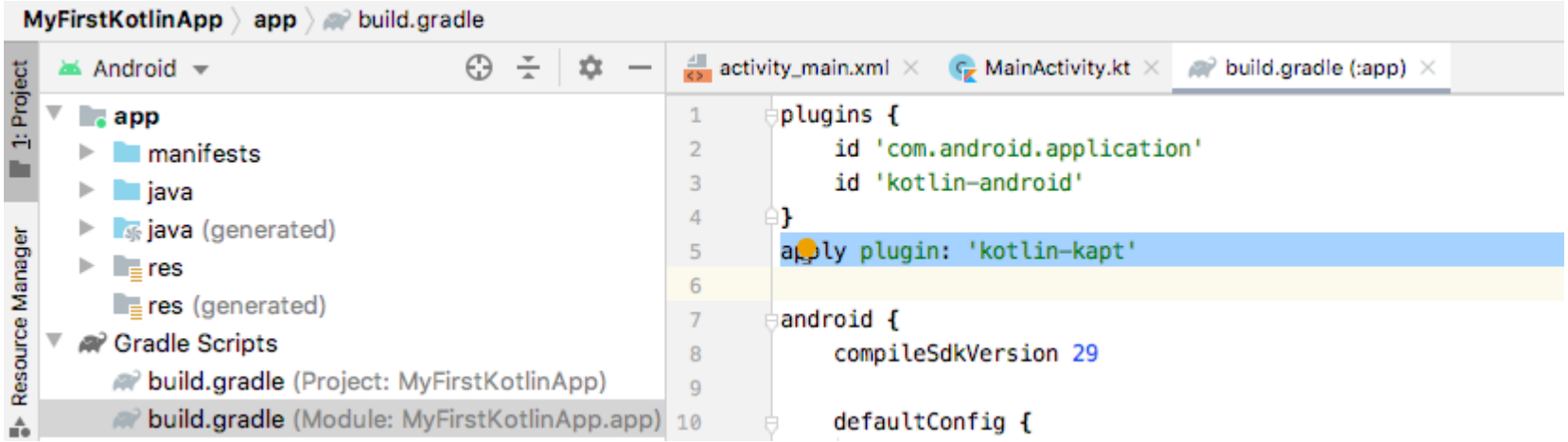
- in app module, Android Studio generates a binding class per layout file
- `findViewById(..)` is no longer needed due to
- camel case “Binding”, e.g. `activity_main.xml` -> `ActivityMainBinding`

1. enable view binding for any project module
2. import generated view binding class
3. obtain a reference to its binding
4. access root view via binding
5. access views as properties of this binding object

software inside

Enable View Binding

apply kotlin-kapt in *Gradle Scripts -> build gradle (Module: app)*



enable the viewBinding property

```
android {
    buildFeatures {
        viewBinding = true
    }
```

and add the data binding compiler library to its dependencies

```
dependencies {
    kapt 'androidx.databinding:databinding-compiler-common:4.0.0'
```

software inside

Inflating/Using a Binding class

```
import com.examples.che.myfirstkotlinapp.databinding.ActivityMainBinding
        // per convention camel case for activity_main.xml

class MainActivity : AppCompatActivity() {
    val TAG = "StateChange"
    var i : Int = 0

    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)  // "blow up" only once
        setContentView(binding.root)
    }

    fun onNewClickMe(view: View) {
        i++
        if (binding.tvMessage.text.isNotEmpty())
            binding.tvMessage.text = "pressed $i times by bindings"   // non-null ensured
    }
```

Properties having a non-null type must be initialized in a constructor. This is often not convenient to initialize by dependency injection, or in a setup method of a unit test, or when you cannot supply a constructor, but you still want to avoid null checks referencing a property inside a class.

To handle this, mark a property by `lateinit`.

Summary to View Bindings

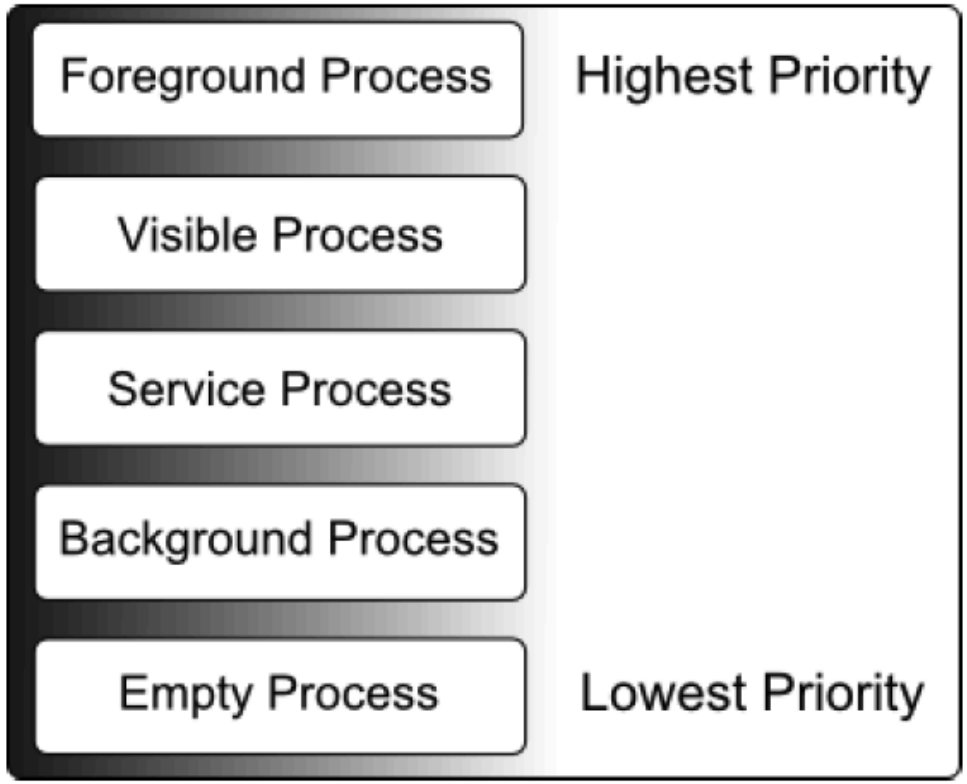
HTL Villach
Future Inside

- clearly safer than `findViewById(..)` to avoid null pointer exceptions
- are not enabled by default (even) in Android Studio 4.x
- manually repeat for each project
- adapt `build.gradle`
- and change `onCreate`

software inside

Remember Application Life Cycle

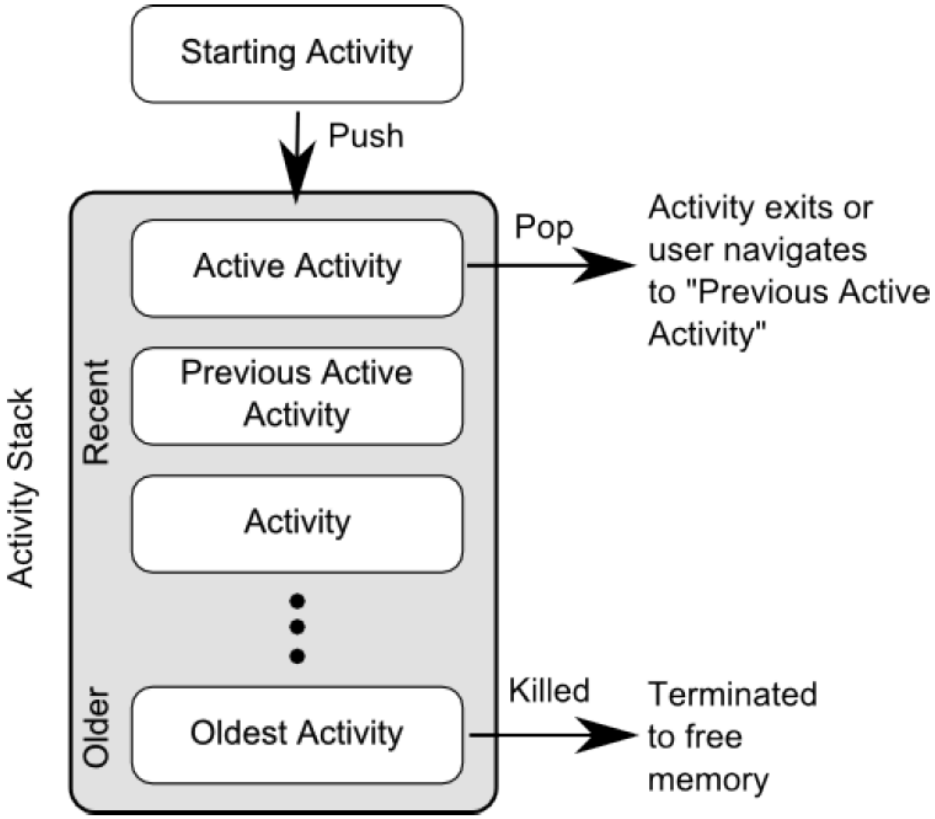
- each app runs in a separate process
- when OS runs out of resources, it terminate processes to free memory
- background processes contains one or more activities which are currently not visible for a user
- empty processes contain inactive apps held in memory to be re-launched



Activity Life Cycle

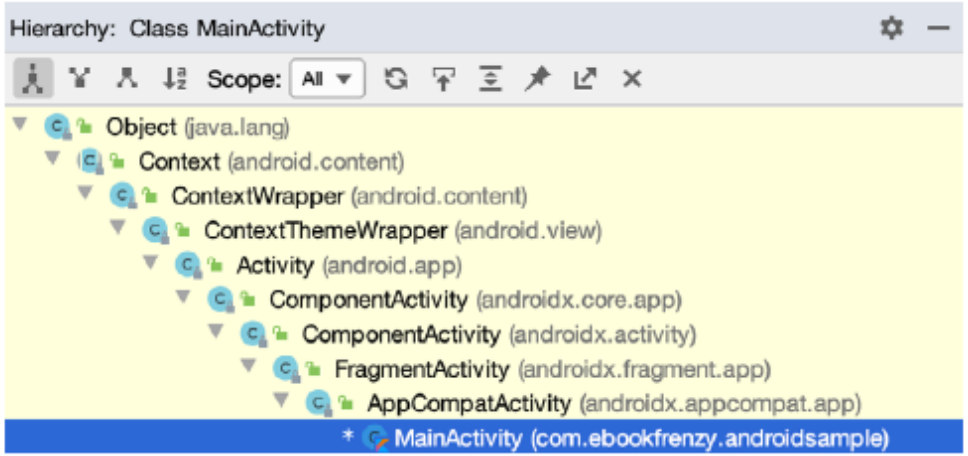
Activity states:

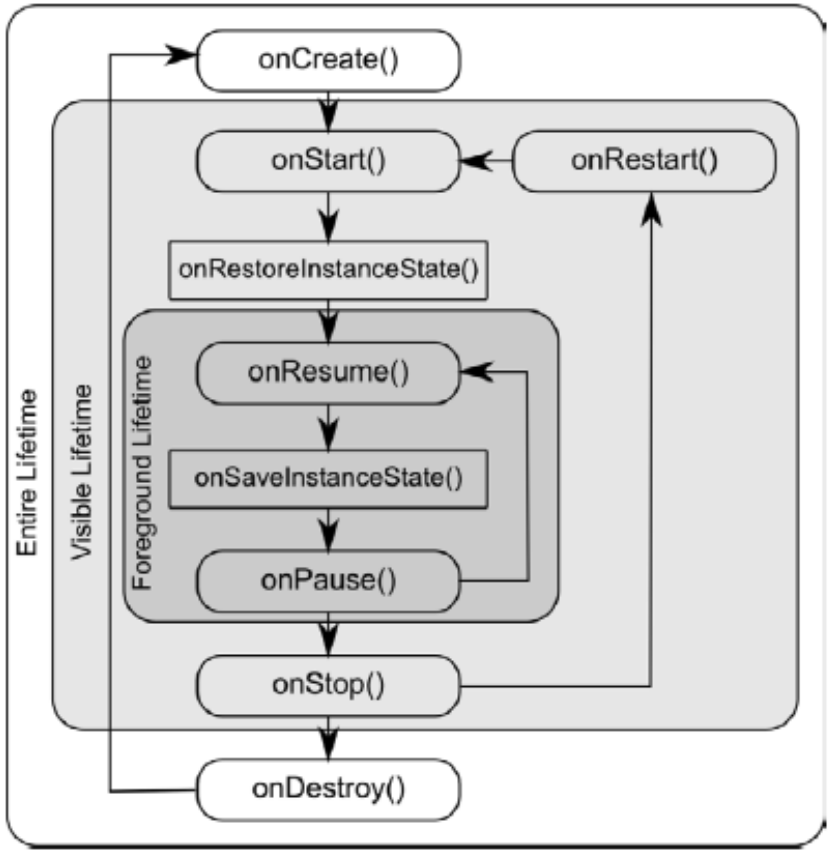
- active / running is in foreground having focus
- paused, ie (partially) visible but without focus
- stopped, ie totally obscured on display by other activities
- killed



State Changes

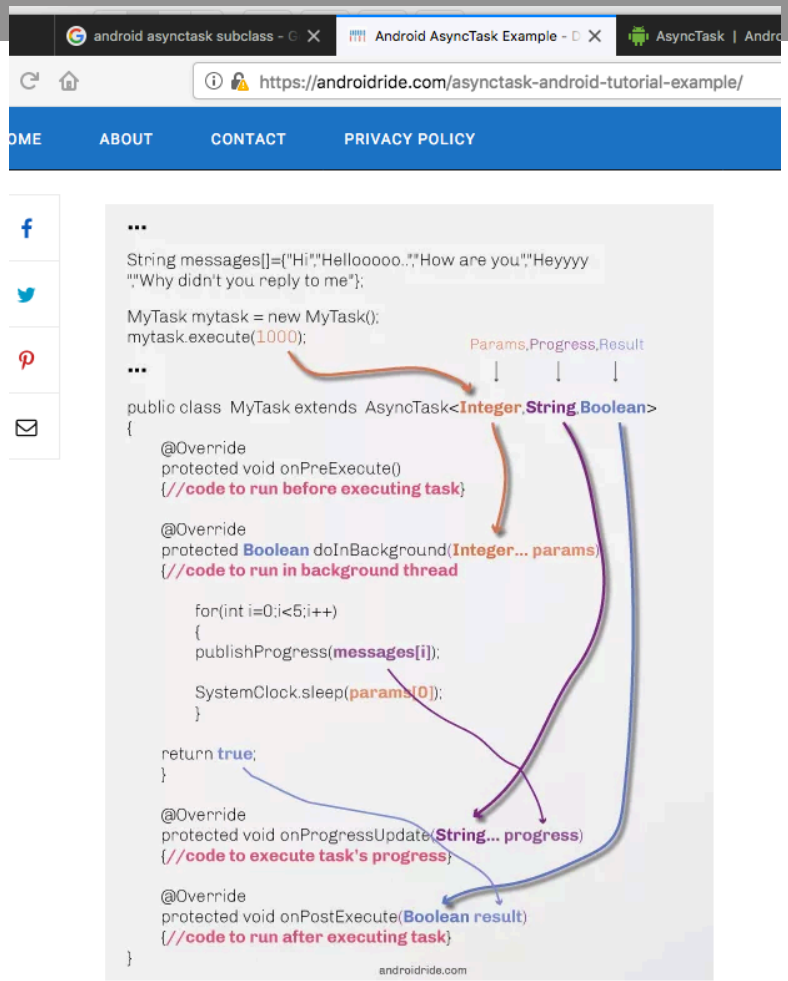
- introduced with Jetpack Android Architecture Components





- generate methods to be overridden by using “Alt-Insert” shortcut (or Command+n on Mac); see at slide 4

Overview on Threads



- threads are a cornerstone for multi-tasking OS
- when an app starts, the runtime creates a single thread, its main thread, to handle user interactions
- any additional component also runs within this main thread
- a (long running) task will lock the entire app until it is completed, and hence, freeces the GUI
- note, Androids UI toolkit is not thread-safe
- use Services or AsyncTaks to avoid such problems

AsyncTask

- is the best approach to implement a background thread for a time consuming task
- just create a sub class of AsyncTask using following template

```
import android.os.AsyncTask

class MainActivity : AppCompatActivity() {

    private inner class MyTask : AsyncTask<String, Void, String>() {

    override fun onPreExecute() {

    }

    override fun doInBackground(vararg params: String): String {

    }

    override fun onProgressUpdate(vararg values: Int?) {

    }

    override fun onPostExecute(result: String) {

    }
}
```

AsyncTask Example

```
import android.os.AsyncTask

class MainActivity : AppCompatActivity() {
    val TAG = "StateChange"
    var i : Int = 0

    private lateinit var binding: ActivityMainBinding

    private inner class MyTask : AsyncTask<String, Int, String>() {

        override fun onPreExecute() { }

        override fun doInBackground(vararg params: String): String {
            var n = 0
            while (n <= 5) {
                try {
                    Thread.sleep( millis: 1000)
                    publishProgress(n)
                    n++
                } catch (e: Exception) {
                    return(e.localizedMessage)
                }
            }
            return "AsyncTask Finished"
        }

        override fun onProgressUpdate(vararg values: Int?) {
            super.onProgressUpdate(*values)
            val counter = values.get(0)
            binding.tvMessage.text = "AsyncTask on progress update $counter"
        }

        override fun onPostExecute(result: String) {
            binding.tvMessage.text = result
        }
    }
```

apply in activity

```
fun onClickStartAsyncTask(view: View) {
    val task = MyTask().executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR)
    // and stopp it per task.cancel()
}
```

To GET from a Web Service
permit Internet Access in Android Manifest

vity_main.xml | MainActivity.kt | AndroidManifest.xml | build.gradle (:app)

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.examples.che.myfirstkotlinapp">
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
```

Responses

JSON XML

YQL Query

select * from weather.forecast where woeid in (select woeid from geo.places(1) where

Test

Response


```json
{
 "query": {
  "count": 1,
  "created": "2017-12-05T03:58:10Z",
  "lang": "en-GB",
  "results": {
   "channel": {
    "units": {
     "distance": "mi",
     "pressure": "in",
     "speed": "mph",
```


software inside

GET from Web Access

```kotlin
private inner class GetWeatherAsyncTask : AsyncTask<String, String, String>() {

    override fun onPreExecute() { }

    override fun doInBackground(vararg urls: String?): String {
        var urlConnection: HttpURLConnection? = null

        try {
            val url = URL(urls[0])

            urlConnection = url.openConnection() as HttpURLConnection
            urlConnection.connectTimeout = CONNECTON_TIMEOUT_MILLISECONDS
            urlConnection.readTimeout = CONNECTON_TIMEOUT_MILLISECONDS

            var inString = streamToString(urlConnection.inputStream)
            publishProgress(inString)
        } catch (ex: Exception) {
            Log.i( tag: "ERR", ex.localizedMessage)
        } finally {
            if (urlConnection != null) {
                urlConnection.disconnect()
            }
        }
        return " "
    }

    override fun onProgressUpdate(vararg values: String?) {
        try {
            var json = JSONObject(values[0])

            val query = json.getJSONObject( name: "query")
            val results = query.getJSONObject( name: "results")
            val channel = results.getJSONObject( name: "channel")
            val location = channel.getJSONObject( name: "location")
            val city = location.get("city")
            val country = location.get("country")
            val humidity = channel.getJSONObject( name: "atmosphere").get("humidity")
            val condition = channel.getJSONObject( name: "item").getJSONObject( name: "condition")
            val temp = condition.get("temp")
            val text = condition.get("text")
            binding.tvMessage.text = "Location: " + city + " - " + country + "\n" +
                                     "Humidity: " + humidity + "\n" +
                                     "Temperature: " + temp + "\n" +
                                     "Status: " + text
        } catch (e: Exception) {
            binding.tvMessage.text = e.localizedMessage
        }
    }

    override fun onPostExecute(result: String?) {}
}
```

```kotlin
fun onClickAccessWebService(view: View) {
    var city = "Villach"
    val url = "https://query.yahooapis.com/v1/public/yql?q='select%20*%20
    val task = GetWeatherAsyncTask().execute(url)
}
```

GET cont'd

HTL Villach
Future Inside

```kotlin
fun streamToString(inputStream: InputStream): String {

    val bufferReader = BufferedReader(InputStreamReader(inputStream))
    var line: String
    var result = ""

    try {
        do {
            line = bufferReader.readLine()
            if (line != null) {
                result += line
            }
        } while (line != null)
        inputStream.close()
    } catch (ex: Exception) {

    }
    return result
}
```

```kotlin
import android.os.AsyncTask
import org.json.JSONObject
import java.io.BufferedReader
import java.io.InputStream
import java.io.InputStreamReader
import java.net.HttpURLConnection
import java.net.URL
```

software inside

Same WLAN needed to access localhost

use a text view or settings menu for the host URL

```kotlin
private inner class GetLocalHostAsyncTask : AsyncTask<String, String, String>() {

    override fun onPreExecute() { }

    override fun doInBackground(vararg urls: String?): String {
        var urlConnection: HttpURLConnection? = null
        try {
            val url = URL(urls[0])
            urlConnection = url.openConnection() as HttpURLConnection
            urlConnection.connectTimeout = CONNECTON_TIMEOUT_MILLISECONDS
            urlConnection.readTimeout = CONNECTON_TIMEOUT_MILLISECONDS
            var inString = streamToString(urlConnection.inputStream)
            publishProgress(inString)
        } catch (ex: Exception) {
            Log.i( tag: "ERR", ex.localizedMessage)
        } finally {
            if (urlConnection != null) {
                urlConnection.disconnect()
            }
        }
        return " "
    }

    override fun onProgressUpdate(vararg values: String?) {
        try {
            binding.tvMessage.text = values[0]
        } catch (e: Exception) {
            binding.tvMessage.text = e.localizedMessage
        }
    }

    override fun onPostExecute(result: String?) {}
}
```

```kotlin
fun onClickGetLocalhost(view: View) {
    //val url = "http://172.27.14.122:8080/HelloWeb/HiServlet"   // use a text view for the URL
    val url = "http://172.27.14.122:8080/02_HelloWebService/rest/privet"
    val task = GetWeatherAsyncTask().execute(url)

}
```

localhost:8080/HelloWeb/HiSe | Apache Tomcat/9.0.20 | Hello HTML Jersey-based REST

localhost:8080/02_HelloWebService/rest/privet

Hello REST World *HTML* Jersey-based RESTful Web Service using JDK/JRE 12.0.1

172.27.14.122:8080/02_HelloWebService/rest/privet | Search

Hello REST World *HTML* Jersey-based RESTful Web Service using JDK/JRE 12.0.1

localhost:8080/HelloWeb/HiSe | Apache Tomcat/9.0.20

localhost:8080/HelloWeb/HiServlet

Hello World: served at: /HelloWeb

172.27.14.122:8080/HelloWeb/HiServlet

Hello World: served at: /HelloWeb

software inside