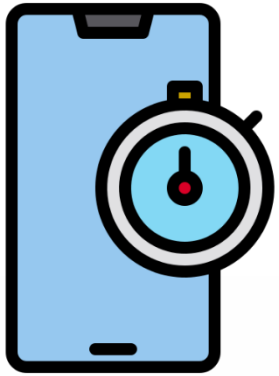
Intro to Java Android Maps and Concurrency

Dmytro Zubov, PhD
dmytro.zubov@ucentralasia.org

Naryn, Kyrgyzstan, 1:11pm, Nov 13, 2022

Lessons learnt last time



- Scope and lifetime of variables
- Intro to file I/O
- A simple Star Wars quiz
- In-class activity
- 2D game in Android Studio (optional)

What we gonna discuss today?

- Java maps
- Intro to Java concurrency
- Develop a web-browser mobile app in 5 min

Java maps

https://examples.javacodegeeks.com/java-basics/java-map-example/
https://developer.android.com/reference/java/util/Map.html

- **Java Map is an interface that maps keys to values**
  - The keys are unique and thus, no duplicate keys are allowed. A map can provide three views, which allow the contents of the map to be viewed as a set of keys, collection of values, or set of key-value mappings. In addition, the order of the map is defined as the order in which, the elements of a map are returned during iteration.
  - The Map interface is implemented by different Java classes, such as `HashMap`, `HashTable`, and `TreeMap`. Each class provides different functionality and can be either synchronized or not. Also, some implementations prohibit null keys and values, and some have restrictions on the types of their keys.

Java maps (cont.)

- A map has the form `Map<K,V>`, where `K` specifies the type of keys maintained in this map, `V` defines the type of mapped values
- The Map interface provides a set of methods that must be implemented. The most used methods:
  - `clear`: Removes all the elements from the map
  - `containsKey`: Returns true if the map contains the requested key
  - `containsValue`: Returns true if the map contains the requested value
  - `equals`: Compares an Object with the map for equality
  - `get`: Retrieve the value of the requested key
  - `keySet`: Returns a Set that contains all keys of the map
  - `put`: Adds the requested key-value pair in the map
  - `remove`: Removes the requested key and its value from the map if key exists
  - `size`: Returns the number of key-value pairs currently in the map

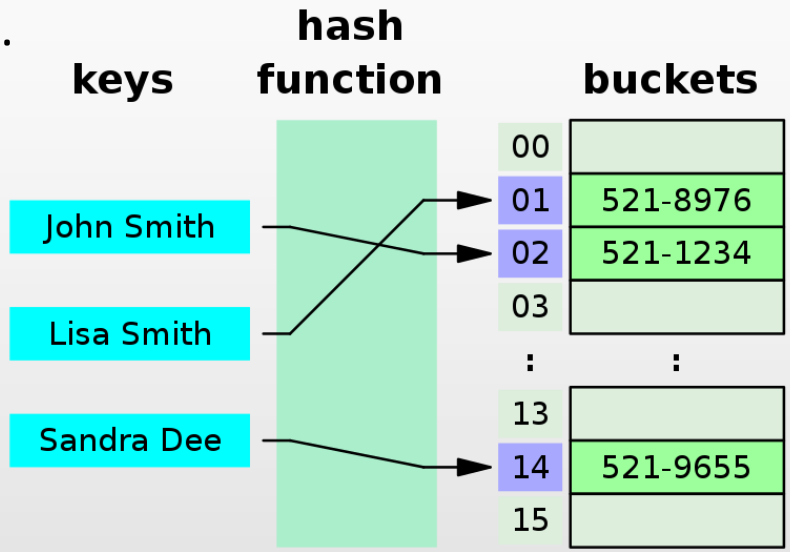
Java maps (cont.)

- `HashMap` is the most common class that implements the Map interface
- `HashMap` is a hash table based on the implementation of the Map interface. **<u>It permits null keys and values</u>**.
  - HashMap does not maintain any order among its elements and especially it does not guarantee that the order will remain constant over time
  - HashMap contains two fundamental parameters: initial capacity and performance. The capacity is defined as the number of buckets in the hash table, while the load factor is a measure that indicates the maximum value the hash table can reach before being automatically increased.

Java maps (cont.)

https://en.wikipedia.org/wiki/Hash_table

- **Just to remind**: A **hash table**, also known as **hash map**, is a data structure that implements an associative array or dictionary. It is an abstract data type that maps keys to values. A hash table uses a hash function to compute an index, also called a hash code, into an array of buckets or slots, from which the desired value can be found. During lookup, the key is hashed and the resulting hash indicates where the corresponding value is stored.



Java maps (cont.)

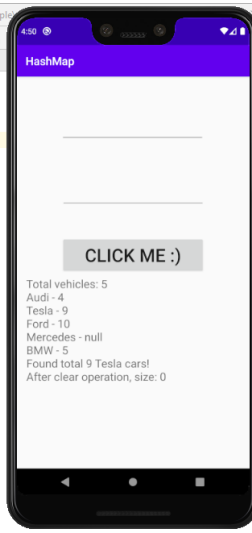
- `HashMap`: An example of Map
  - **Pls pay attention that we <u>CAN</u> use `null` value here**

```java
public void onButtonClick(View view) {
    Map vehicles = new HashMap();

    // Add some vehicles
    vehicles.put("Mercedes", 3);
    vehicles.put("BMW", 5);
    vehicles.put("Audi", 4);
    vehicles.put("Ford", 10);
    vehicles.put("Tesla", 9);
    textView.setText("Total vehicles: " + vehicles.size());

    // Iterate over all vehicles, using the keySet method
    for(Object key: vehicles.keySet())
        textView.setText(textView.getText() + "\r\n" + key + " - " + vehicles.get(key));

    String searchKey = "Tesla";

    if(vehicles.containsKey(searchKey))
        textView.setText(textView.getText() + "\r\n" + "Found total " + vehicles.get(searchKey) + " " + searchKey + " cars!");

    // Clear all values
    vehicles.clear();

    // Equals to zero
    textView.setText(textView.getText() + "\r\n" + "After clear operation, size: " + vehicles.size());
}
}
```

Install successfully finished in 1 s 868 ms.

```java
public void onButtonClick(View view) {
    Map vehicles = new HashMap();

    // Add some vehicles
    vehicles.put("Mercedes", null);
    vehicles.put("BMW", 5);
    vehicles.put("Audi", 4);
    vehicles.put("Ford", 10);
    vehicles.put("Tesla", 9);
    textView.setText("Total vehicles: " + vehicles.size());

    // Iterate over all vehicles, using the keySet method
    for(Object key: vehicles.keySet())
        textView.setText(textView.getText() + "\r\n" + key + " - " + vehicles.get(key));

    String searchKey = "Tesla";

    if(vehicles.containsKey(searchKey))
        textView.setText(textView.getText() + "\r\n" + "Found total " + vehicles.get(searchKey) + " " + searchKey + " cars!");

    // Clear all values
    vehicles.clear();

    // Equals to zero
    textView.setText(textView.getText() + "\r\n" + "After clear operation, size: " + vehicles.size());
}
}
```

Install successfully finished in 1 s 289 ms.

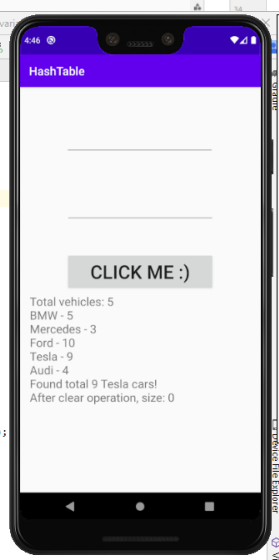


Java maps (cont.)

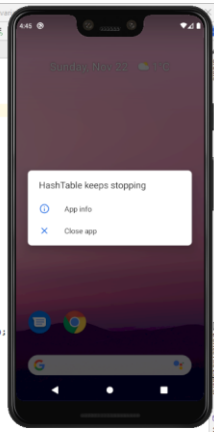
- The HashTable class implements a hash table and maps keys to values. However, **<u>neither the key nor the value can be null</u>**. This class contains two fundamental parameters: initial capacity and performance, with the same definitions as the HashMap class.

° **Pls pay attention that we <u>CANNOT</u> use `null` value here**

```
public void onButtonClick(View view) {
    Map vehicles = new Hashtable();

    // Add some vehicles
    vehicles.put("Mercedes", 3);
    vehicles.put("BMW", 5);
    vehicles.put("Audi", 4);
    vehicles.put("Ford", 10);
    vehicles.put("Tesla", 9);
    textView.setText("Total vehicles: " + vehicles.size());

    // Iterate over all vehicles, using the keySet method
    for(Object key: vehicles.keySet())
        textView.setText(textView.getText() + "\r\n" + key + " - " + vehicles.get(key));

    String searchKey = "Tesla";

    if(vehicles.containsKey(searchKey))
        textView.setText(textView.getText() + "\r\n" + "Found total " + vehicles.get(searchKey) + " " + searchKey + " cars!");

    // Clear all values
    vehicles.clear();

    // Equals to zero
    textView.setText(textView.getText() + "\r\n" + "After clear operation, size: " + vehicles.size());
}
```

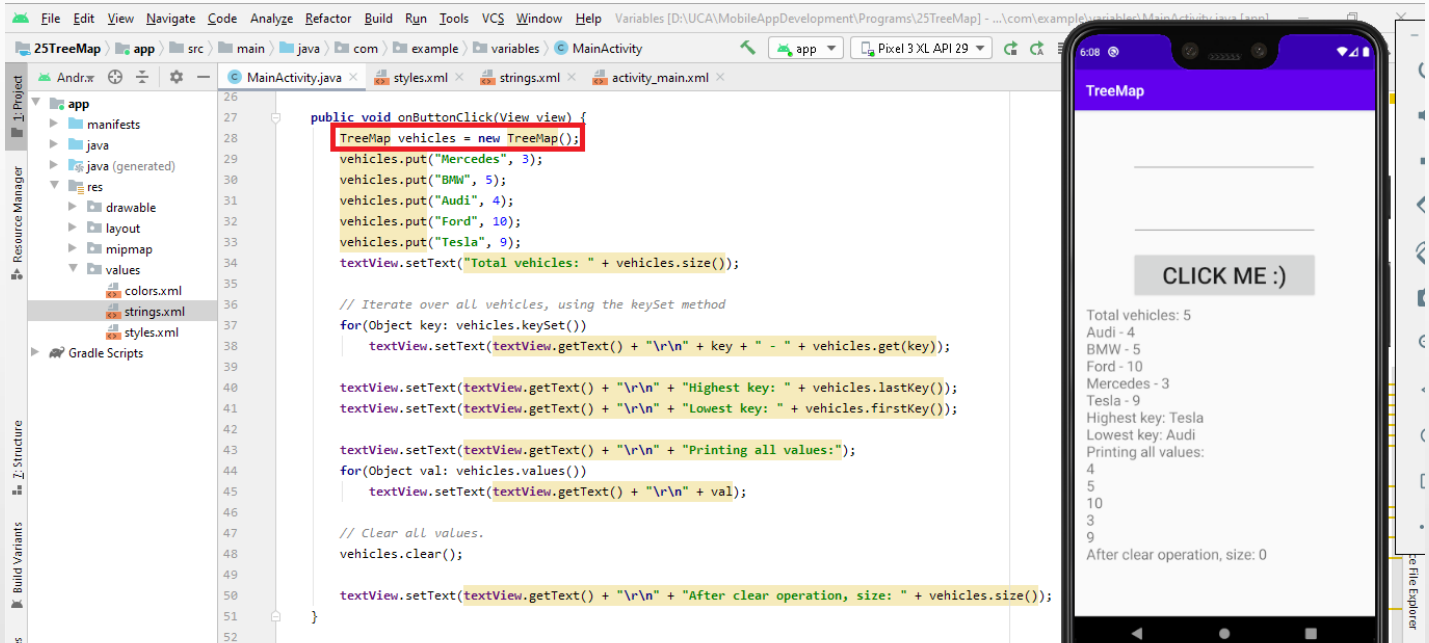



```
public void onButtonClick(View view) {
    Map vehicles = new Hashtable();

    // Add some vehicles
    vehicles.put("Mercedes", null);
    vehicles.put("BMW", 5);
    vehicles.put("Audi", 4);
    vehicles.put("Ford", 10);
    vehicles.put("Tesla", 9);
    textView.setText("Total vehicles: " + vehicles.size());

    // Iterate over all vehicles, using the keySet method
    for(Object key: vehicles.keySet())
        textView.setText(textView.getText() + "\r\n" + key + " - " + vehicles.get(key));

    String searchKey = "Tesla";

    if(vehicles.containsKey(searchKey))
        textView.setText(textView.getText() + "\r\n" + "Found total " + vehicles.get(searchKey) + " " + searchKey + " cars!");

    // Clear all values
    vehicles.clear();

    // Equals to zero
    textView.setText(textView.getText() + "\r\n" + "After clear operation, size: " + vehicles.size());
}
```



Java maps (cont.)

- The TreeMap is a Red-Black tree implementation that is sorted according to the natural ordering of its keys, or by a Comparator provided at the creation time
  - This class maintains an order on its elements
  - This class is not synchronized and hence if an application uses multiple threads, the map must be synchronized externally



Java maps (cont.)

- **ConcurrentHashMap** class is a hash table that supports full concurrency of retrievals. Thus, this structure is safe to use in case of multiple threads. **<u>This class does not allow neither keys nor values to be null.</u>**

```java
public void onButtonClick(View view) {
    ConcurrentHashMap vehicles = new ConcurrentHashMap();

    // Add some vehicles.
    vehicles.put("Mercedes", 3);
    vehicles.put("BMW", 5);
    vehicles.put("Audi", 4);
    vehicles.put("Ford", 10);
    vehicles.put("Tesla", 9);

    textView.setText("Total vehicles: " + vehicles.size());

    // Iterate over all vehicles, using the keySet method.
    for(Object key: vehicles.keySet())
        textView.setText(textView.getText() + "\r\n" + key + " - " + vehicles.get(key));

    String searchKey = "Tesla";
    if(vehicles.containsKey(searchKey))
        textView.setText(textView.getText() + "\r\n" + "Found total " + vehicles.get(searchKey) + " " + searchKey + "

    Enumeration elems = vehicles.elements();
    while(elems.hasMoreElements())
        textView.setText(textView.getText() + "\r\n" + elems.nextElement());

    Object val = vehicles.putIfAbsent("Audi", 9);
    if(val != null)
        textView.setText(textView.getText() + "\r\n" + "Audi was found in the map and its value was updated!");

    val = vehicles.putIfAbsent("Nissan", 9);
    if(val == null)
        textView.setText(textView.getText() + "\r\n" + "Nissan wasn't found in map, thus a new pair was created!");
```

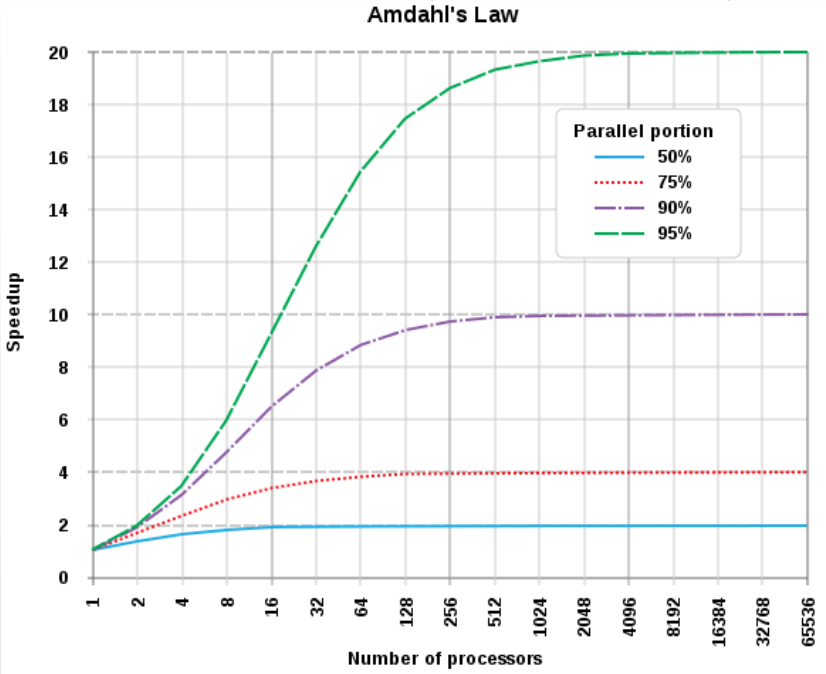
```
Total vehicles: 5
Audi - 4
Tesla - 9
Ford - 10
Mercedes - 3
BMW - 5
Found total 9 Tesla cars!
4
9
10
3
5
Audi was found in the map and its value
was updated!
Nissan wasn't found in map, thus a new
pair was created!
After clear operation, size: 0
```

Intro to Java concurrency (multi-threading)

- Concurrency is the ability to run several programs or several parts of a program in parallel. **If a time-consuming task can be performed asynchronously or in parallel, this improve the throughput and the interactivity of the program.**
  - A **process** runs independently and isolated to other processes. It cannot directly access shared data in other processes. The resources of the process, e.g., memory and CPU time, are allocated to it via the OS.
  - A **thread** is a so-called lightweight process. It has its own call stack but can access shared data of other threads in the same process. Every thread has its own memory cache. If a thread reads shared data, it stores this data in its own memory cache. A thread can re-read the shared data.
  - A Java application runs by default in one process.

Intro to Java concurrency (cont.)

- Limits of concurrency gains
  - Java apps can work with several threads to achieve parallel processing, i.e., asynchronous behavior
  - Concurrency promises to perform certain task faster as these tasks can be divided into subtasks and these subtasks can be executed in parallel
  - The runtime is limited by parts of the task which can be performed in parallel
  - The theoretical possible performance gain can be calculated by the following rule which is referred to as Amdahl's Law:



https://en.wikipedia.org/wiki/Amdahl%27s_law

Intro to Java concurrency (cont.)

- Concurrency issues
  - Threads have their own call stack but can also access shared data. Therefore, we have two basic problems – ***visibility*** and ***access*** problems.
  - A **visibility problem** occurs if thread `A` reads shared data which is later changed by thread `B` and thread `A` is unaware of this change
  - An **access problem** can occur if several threads access and change the same shared data at the same time
  - Visibility and access problem can lead to:
    - Liveness failure: The program does not react anymore due to problems in the concurrent access of data, e.g., deadlocks
    - Safety failure: The program creates incorrect data

Intro to Java concurrency (cont.)

- Threads

```
public class Thread
extends Object implements Runnable

java.lang.Object
   ↳    java.lang.Thread
```

° A thread is a thread of execution in a program. *JVM allows an application to have multiple threads of execution running concurrently*.

° Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority.

° Each thread may or may not also be marked as a daemon. When code running in some thread creates a new Thread object, the new thread has its priority initially set equal to the priority of the creating thread and is a daemon thread if and only if the creating thread is a daemon.

Intro to Java concurrency (cont.)

- Threads (cont.)
  - The `java.lang.Thread` class is a thread of execution in a program. A `Thread` executes an object of type `java.lang.Runnable`.
  - `Runnable` is an interface that defines the `run()` method. This method is called by the `Thread` object and contains the work which should be done. Therefore, the `Runnable` is the task to perform. **The Thread is the worker who is doing this task**.

https://www.vogella.com/tutorials/JavaConcurrency/article.html

Intro to Java concurrency (cont.)

- The following example demonstrates a task (`Runnable`) which counts the sum of a given range of numbers

As far as you see, 4 threads calculate the sum approximately 4 times faster compared with 1 thread.

MainActivity.java | activity_main.xml | strings.xml

```java
class SyncThread implements Runnable{
    private int in;  // Number of the part
    private int min; // Minimum index in the array
    private int max; // Maximum index in the array

    public SyncThread(int in, int min, int max){
        this.in=in; this.min=min; this.max=max;
    }

    public void run() {
        //Getting the current date
        Date date1 = new Date();
        //Returns current time in millis
        timeMilli1[in] = date1.getTime();
        for (i1[in]=min; i1[in]<=max; i1[in] = i1[in] + 0.000001) {}
        Date date2 = new Date();
        timeMilli2[in] = date2.getTime();
    }
}
```

it View Navigate Code Analyze Refactor Build Run Tools VCS Window Help Variables [D:\UCA\MobileAppDe

dRunnable 〉 app 〉 src 〉 main 〉 java 〉 com 〉 example 〉 variables 〉 MainActivity

MainActivity.java | activity_main.xml | strings.xml

```java
        //Getting the current date
        Date date1 = new Date();
        //Returns current time in millis
        timeMilli1[in] = date1.getTime();
        for (i1[in]=min; i1[in]<=max; i1[in] = i1[in] + 0.000001) {}
        Date date2 = new Date();
        timeMilli2[in] = date2.getTime();
    }
}

double [] i1= new double[5];
long [] timeMilli1 = new long [5];
long [] timeMilli2 = new long [5];

public void onButtonClick(View view) {
    for (int i=0; i<5; i++) {i1[i] = 0;}
    Thread t1 = new Thread(new SyncThread( in: 0,  min: 1,  max: 250),  name: "t1");
    Thread t2 = new Thread(new SyncThread( in: 1,  min: 251,  max: 500),  name: "t2");
    Thread t3 = new Thread(new SyncThread( in: 2,  min: 501,  max: 750),  name: "t3");
    Thread t4 = new Thread(new SyncThread( in: 3,  min: 751,  max: 1000),  name: "t4");
    Thread t5 = new Thread(new SyncThread( in: 4,  min: 1,  max: 2500),  name: "t5");

    try {
        t1.start(); t2.start(); t3.start(); t4.start(); t5.start();
        t1.join();  t2.join();  t3.join();  t4.join(); t5.join();
    } catch (Exception e) { textView.setText(e.toString());}

    i1[0] = i1[0] + i1[1] + i1[2] + i1[3];
    textView.setText(i1[0] + "  " + (timeMilli2[3]-timeMilli1[3]) + "\r\n" + i1[4] + "  " + (timeMilli2[4]-timeMilli1[4]));
}
```

ainActivity 〉 onButtonClick()

Intro to Java concurrency (cont.)

- Threads (cont.)
  - *When a JVM starts up, there is usually a single non-daemon thread* (which typically calls the method named main of some designated class). The JVM continues to execute threads until either of the following occurs:
    - The `exit` method of class Runtime has been called and the security manager has permitted the exit operation to take place
    - All threads that are not daemon threads have died, either by returning from the call to the `run` method or by throwing an exception that propagates beyond the `run` method

Intro to Java concurrency (cont.)

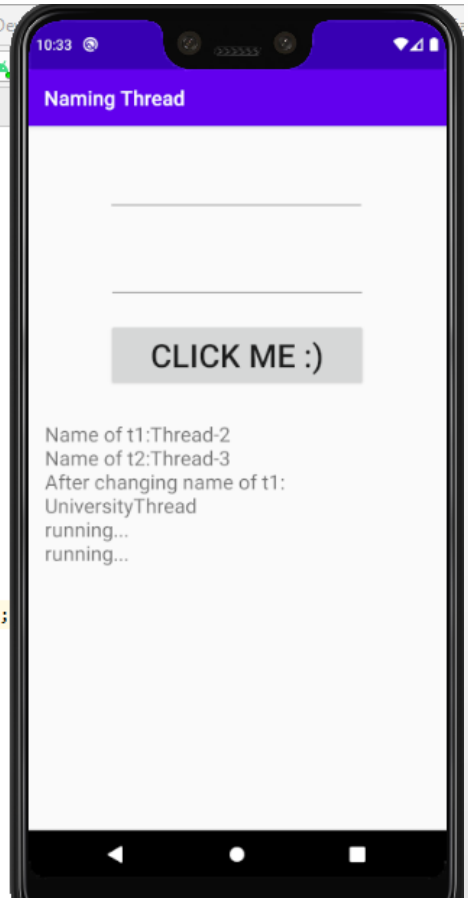
- Naming thread:
  - The `Thread` class provides methods to change and get the name of a thread
  - By default, each thread has a name `thread-0`, `thread-1` and so on. We can change the name of the thread by using `setName()` method. The syntax of `setName()` and `getName()` methods are as follows:
    - `public String getName()` is used to return the name of a thread
    - `public void setName(String name)` is used to change the name of a thread

https://www.javatpoint.com/naming-a-thread

Intro to Java concurrency (cont.)

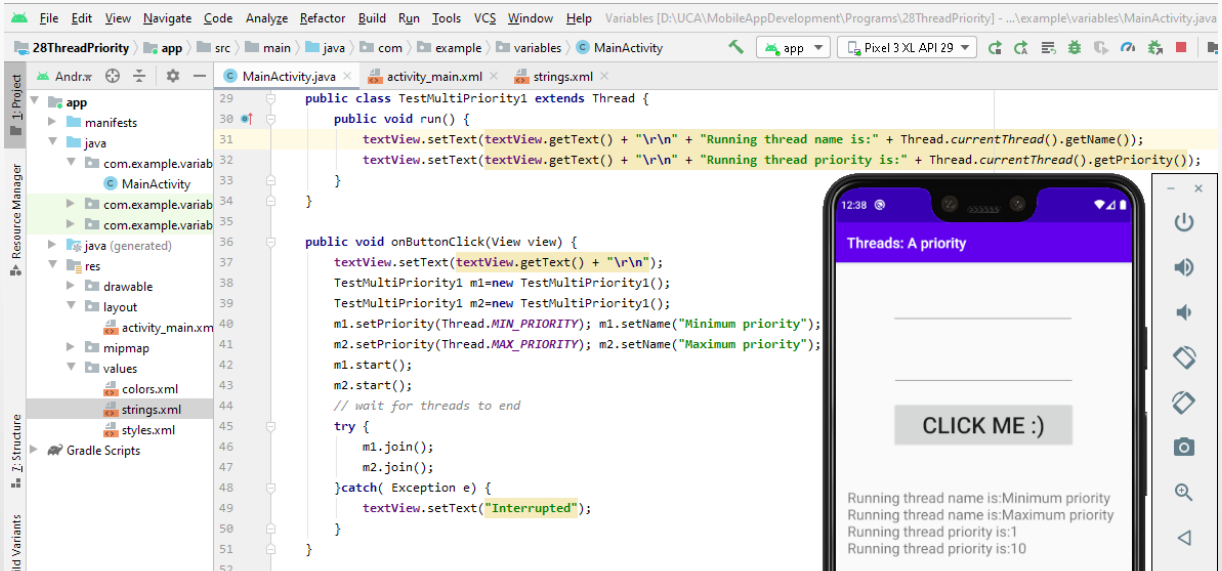
- The `run` method is started when the object of class *TestMultiPriority1* is started

```
class TestMultiNaming1 extends Thread {
    public void run() {
        s1 = s1 + "\r\n" + "running...";
    }
}

static String s1="";

public void onButtonClick(View view) {
    TestMultiNaming1 t1=new TestMultiNaming1();
    TestMultiNaming1 t2=new TestMultiNaming1();
    s1 = s1 + "\r\n" + "Name of t1:"+t1.getName();
    s1 = s1 + "\r\n" + "Name of t2:"+t2.getName();
    t1.start();
    t2.start();
    t1.setName("UniversityThread");
    s1 = s1 + "\r\n" + "After changing name of t1: "+t1.getName();
    try{
        // Wait for threads to end
        t1.join(); t2.join();
    } catch (Exception e)
    {
        textView.setText("Interrupted: " + e);
    }
    textView.setText(s1);
}
```



Intro to Java concurrency (cont.)

- Java threads: A priority example
  - Three constants defined in Thread class:
    - public static int MIN_PRIORITY
    - public static int NORM_PRIORITY
    - public static int MAX_PRIORITY
  - Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.



Intro to Java concurrency (cont.)

- Locks and thread synchronization
  - Java provides locks to protect certain parts of the code to be executed by several threads at the same time. *The simplest way of locking a certain method or Java class is to define the method or class with the* ***synchronized*** *keyword.* **All code which is protected by the same lock can only be executed by one thread at the same time.**
  - The synchronized keyword in Java ensures:
    - that only a single thread can execute a block of code at the same time
    - that each thread entering a synchronized block of code sees the effects of all previous modifications that were guarded by the same lock

Intro to Java concurrency (cont.)

- An example WITHOUT synchronization (**numbers aren't sorted**)

```java
public void onButtonClick(View view) {
    Table obj = new Table();//only one object
    MyThread1 t1=new MyThread1(obj);
    MyThread2 t2=new MyThread2(obj);
    t1.start();
    t2.start();
    // wait for threads to end
    try {
        t1.join(); t2.join();
    } catch (Exception e) {
        s1 = s1 + "\r\n" + e;
    }
    textView.setText(s1);
}
```

```java
class Table {
//      synchronized void printTable (int n) {//method is synchronized
        void printTable (int n) {//method isn't synchronized
            for(int i=1;i<=5;i++){
                s1 = s1 + "\r\n" + n*i;
                try{
                    Thread.sleep( millis: 400);
                }catch(Exception e){s1 = s1 + "\r\n" + e;}
            }
        }
    }

    class MyThread1 extends Thread{
        Table t;
        MyThread1(Table t){ this.t=t; }
        public void run(){ t.printTable( n: 5); }
    }

    class MyThread2 extends Thread{
        Table t;
        MyThread2(Table t){ this.t=t; }
        public void run(){ t.printTable( n: 100); }
    }

    static String s1="";

    public void onButtonClick(View view) {
        Table obj = new Table();//only one object
        MyThread1 t1=new MyThread1(obj);
```

Intro to Java concurrency (cont.)

- An example WITH synchronization (**numbers are sorted**)

```
public void onButtonClick(View view) {
    Table obj = new Table();//only one object
    MyThread1 t1=new MyThread1(obj);
    MyThread2 t2=new MyThread2(obj);
    t1.start();
    t2.start();
    // wait for threads to end
    try {
        t1.join(); t2.join();
    } catch (Exception e) {
        s1 = s1 + "\r\n" + e;
    }
    textView.setText(s1);
}
```

```
class Table {
    synchronized void printTable (int n) {//method is synchronized
//  void printTable (int n) {//method isn't synchronized
        for(int i=1;i<=5;i++){
            s1 = s1 + "\r\n" + n*i;
            try{
                Thread.sleep( millis: 400);
            }catch(Exception e){s1 = s1 + "\r\n" + e;}
        }
    }
}

class MyThread1 extends Thread{
    Table t;
    MyThread1(Table t){ this.t=t; }
    public void run(){ t.printTable( n: 5); }
}

class MyThread2 extends Thread{
    Table t;
    MyThread2(Table t){ this.t=t; }
    public void run(){ t.printTable( n: 100); }
}

static String s1="";

public void onButtonClick(View view) {
    Table obj = new Table();//only one object
    MyThread1 t1=new MyThread1(obj);
```

Intro to Java concurrency (cont.)

- `volatile` keyword:
  - If a variable is declared with the `volatile` keyword, then it is guaranteed that any thread that reads the field will see the most recent written value. The `volatile` keyword will not perform any mutual exclusive lock on the variable.

https://dzone.com/articles/java-volatile-keyword-0

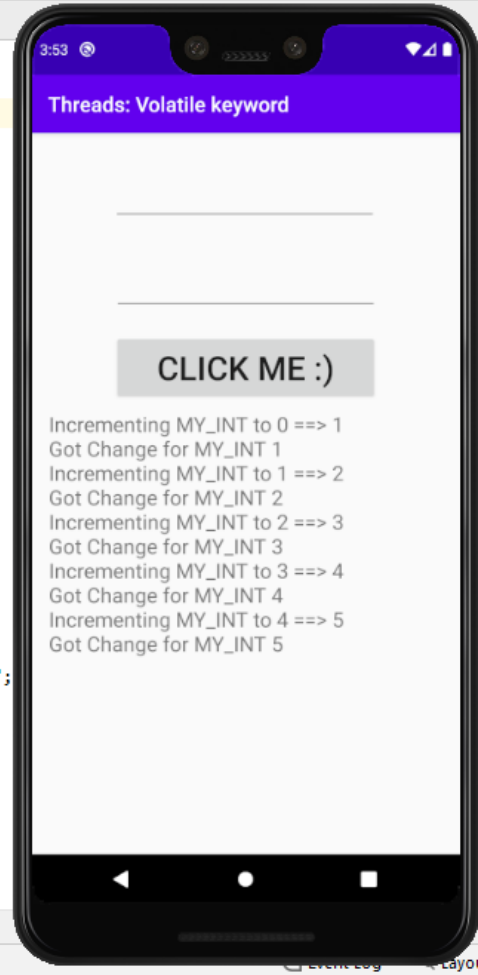
Intro to Java concurrency (cont.)

- WITH `volatile` keyword:

MainActivity.java | activity_main.xml | strings.xml

```java
    private String s1 = "";
    private static volatile int MY_INT = 0;
//    private static int MY_INT = 0;

    class ChangeListener extends Thread {
        @Override
        public void run() {
            int local_value = MY_INT;
            while ( local_value < 5){
                if( local_value!= MY_INT){
                    s1 = s1 + "Got Change for MY_INT " + MY_INT + "\r\n";
                    local_value= MY_INT;
                }
            }
        }
    }
    class ChangeMaker extends Thread{
        @Override
        public void run() {
            int local_value = MY_INT;
            while (MY_INT <5){
                s1 = s1 + "Incrementing MY_INT to " + MY_INT + " ==> " + (local_value+1)  + "\r\n";
                MY_INT = ++local_value;
                try {
                    Thread.sleep( millis: 500);
                } catch (InterruptedException e) { e.printStackTrace(); }
            }
        }
    }
```

```java
public void onButtonClick(View view) {
    ChangeListener t1 = new ChangeListener();
    ChangeMaker t2 = new ChangeMaker();
    t1.start();
    t2.start();
    try {
        t1.join();
        t2.join();
    } catch (Exception e) { s1 = s1 + e;}
    textView.setText(s1);
}
```



MainActivity

Profiler | 6: Logcat | Terminal

Intro to Java concurrency (cont.)

- WITHOUT `volatile` keyword:

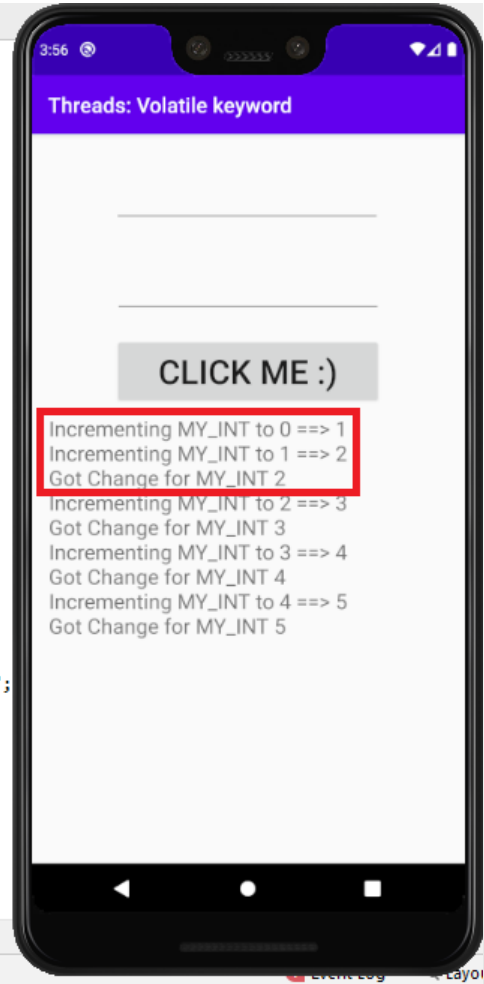
MainActivity.java | activity_main.xml | strings.xml

```java
    private String s1 = "";
//  private static volatile int MY_INT = 0;
    private static int MY_INT = 0;

    class ChangeListener extends Thread {
        @Override
        public void run() {
            int local_value = MY_INT;
            while ( local_value < 5){
                if( local_value!= MY_INT){
                    s1 = s1 + "Got Change for MY_INT " + MY_INT + "\r\n";
                    local_value= MY_INT;
                }
            }
        }
    }
    class ChangeMaker extends Thread{
        @Override
        public void run() {
            int local_value = MY_INT;
            while (MY_INT <5){
                s1 = s1 + "Incrementing MY_INT to " + MY_INT + " ==> " + (local_value+1)  + "\r\n";
                MY_INT = ++local_value;
                try {
                    Thread.sleep( millis: 500);
                } catch (InterruptedException e) { e.printStackTrace(); }
            }
        }
    }
```

```java
public void onButtonClick(View view) {
    ChangeListener t1 = new ChangeListener();
    ChangeMaker t2 = new ChangeMaker();
    t1.start();
    t2.start();
    try {
        t1.join();
        t2.join();
    } catch (Exception e) { s1 = s1 + e;}
    textView.setText(s1);
}
```

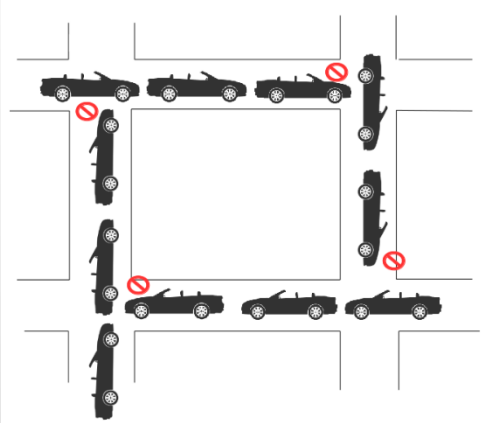
MainActivity › onCreate()

d | Profiler | 6: Logcat | Terminal



Intro to Java concurrency (cont.)

- Deadlock
  - Deadlock describes a situation where two or more threads are blocked forever, waiting for each other
  - Deadlock occurs when multiple threads need the same locks but obtain them in different order
  - A Java multithreaded program may suffer from the deadlock condition because the `synchronized` keyword causes the executing thread to block while waiting for the lock, or monitor, associated with the specified object



https://www.tutorialspoint.com/java_concurrency/concurrency_deadlock.htm

Intro to Java concurrency (cont.)

- Deadlock (cont.)
  - Click the button … and nothing is changed on the screen … deadlock …

```java
public static Object Lock1 = new Object();
public static Object Lock2 = new Object();
static String s1 = "";

private static class ThreadDemo1 extends Thread {

    public void run() {

        synchronized (Lock1) {
            s1 = s1 + "Thread 1: Holding lock 1...\r\n";

            try {
                Thread.sleep( millis: 10);
            } catch (InterruptedException e) {}
            s1 = s1 + "Thread 1: Waiting for lock 2...\r\n";

            synchronized (Lock2) {
                s1 = s1 + "Thread 1: Holding lock 1 & 2...\r\n";
            }
        }
    }
}
```

MainActivity.java | activity_main.xml | strings.xml

```java
48      private static class ThreadDemo2 extends Thread {
49          public void run() {
50
51              synchronized (Lock2) {
52                  s1 = s1 + "Thread 2: Holding lock 2...\r\n";
53
54                  try {
55                      Thread.sleep( millis: 10);
56                  } catch (InterruptedException e) {}
57                  s1 = s1 + "Thread 2: Waiting for lock 1...\r\n";
58
59                  synchronized (Lock1) {
60                      s1 = s1 + "Thread 2: Holding lock 1 & 2...\r\n";
61                  }
62              }
63          }
64      }
65
66      public void onButtonClick(View view) {
67          ThreadDemo1 t1 = new ThreadDemo1();
68          ThreadDemo2 t2 = new ThreadDemo2();
69          t1.start();
70          t2.start();
71          try {
72              t1.join();
73              t2.join();
74          } catch (Exception e) { s1 = s1 + e;}
75          textView.setText(s1);
76      }
```

MainActivity › ThreadDemo1 › run()

Intro to Java concurrency (cont.)

- Deadlock (cont.)
  - Changing the order of the locks prevents the program in going into a deadlock situation:

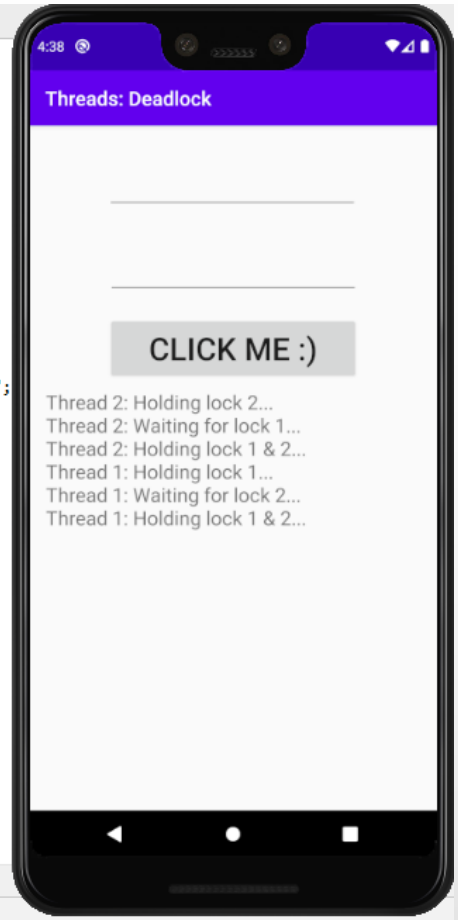
```java
public static Object Lock1 = new Object();
public static Object Lock2 = new Object();
static String s1 = "";

private static class ThreadDemo1 extends Thread {

    public void run() {

        synchronized (Lock1) {
            s1 = s1 + "Thread 1: Holding lock 1...\r\n";

            try {
                Thread.sleep( millis: 10);
            } catch (InterruptedException e) {}
            s1 = s1 + "Thread 1: Waiting for lock 2...\r\n";

            synchronized (Lock2) {
                s1 = s1 + "Thread 1: Holding lock 1 & 2...\r\n";
            }
        }
    }
}
```

MainActivity.java | activity_main.xml | strings.xml

```java
48  private static class ThreadDemo2 extends Thread {
49      public void run() {
50
51          synchronized (Lock1) {
52              s1 = s1 + "Thread 2: Holding lock 2...\r\n";
53
54              try {
55                  Thread.sleep( millis: 10);
56              } catch (InterruptedException e) {}
57              s1 = s1 + "Thread 2: Waiting for lock 1...\r\n";
58
59              synchronized (Lock2) {
60                  s1 = s1 + "Thread 2: Holding lock 1 & 2...\r\n";
61              }
62          }
63      }
64  }
65
66  public void onButtonClick(View view) {
67      ThreadDemo1 t1 = new ThreadDemo1();
68      ThreadDemo2 t2 = new ThreadDemo2();
69      t1.start();
70      t2.start();
71      try {
72          t1.join();
73          t2.join();
74      } catch (Exception e) { s1 = s1 + e;}
75      textView.setText(s1);
76  }
```

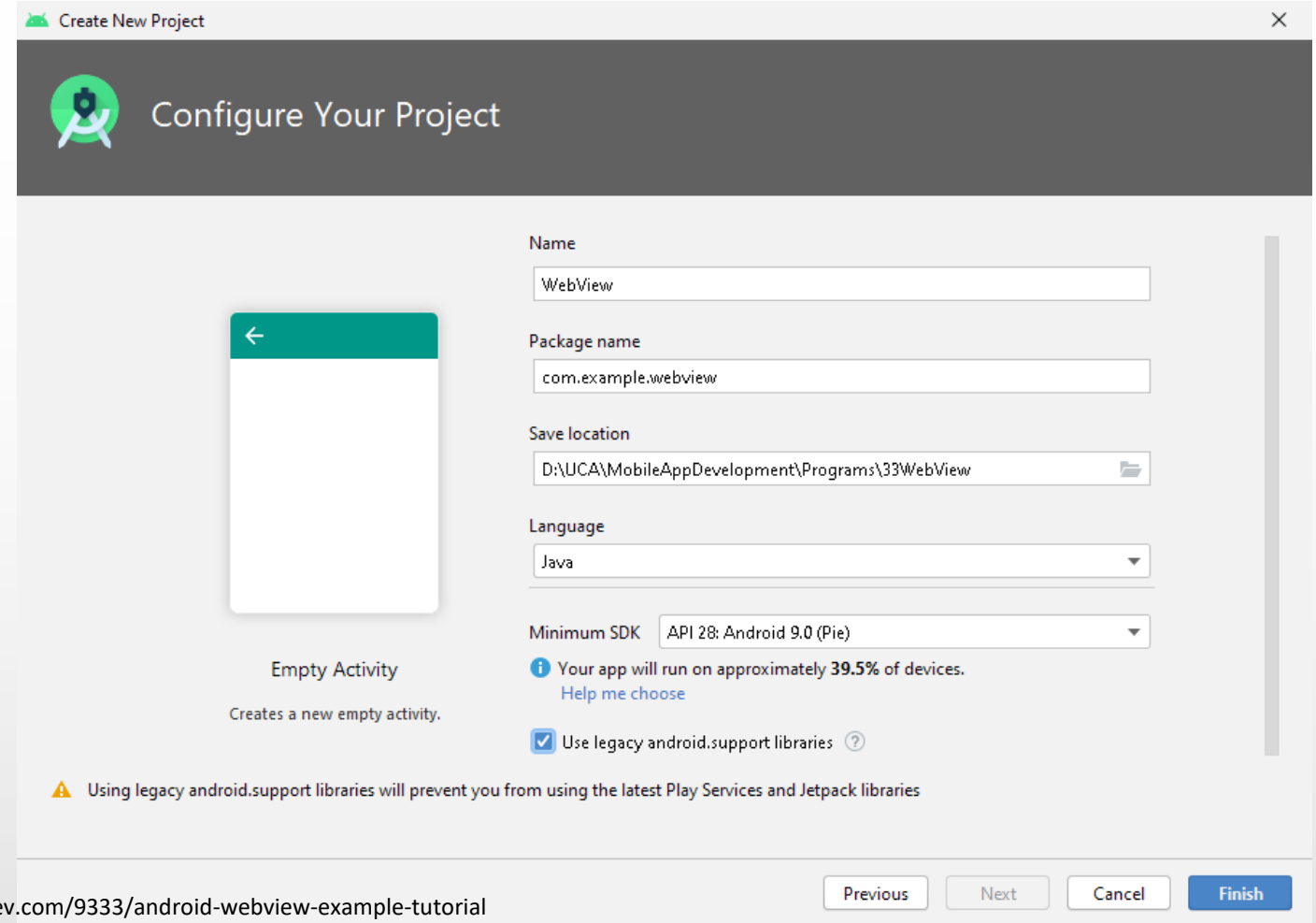
MainActivity › ThreadDemo1 › run()

d | Profiler | 6: Logcat | Terminal



Develop a web-browser mobile app in 5 min

- Create a new empty project:



https://www.journaldev.com/9333/android-webview-example-tutorial

Develop a web-browser mobile app in 5 min (cont.)

- In the file activity_main.xml, delete TextView element and drag and drop WebView:

activity_main.xml | MainActivity.java | AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <WebView
        android:id="@+id/webView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

Develop a web-browser mobile app in 5 min (cont.)

- Open up the file MainActivity.java and enter the following statements inside the callback OnCreate:

activity_main.xml | MainActivity.java | AndroidManifest.xml

```java
package com.example.webview;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.webkit.WebView;
import android.webkit.WebViewClient;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        WebView webView= (WebView)findViewById(R.id.webView);
//findViewById returns an instance of View ,which is casted to target class
        webView.setWebViewClient(new WebViewClient());
//This statement keeps page navigation within the WebView and hence within the app
        webView.getSettings().setJavaScriptEnabled(true);
//This statement is used to enable the execution of JavaScript
        webView.setVerticalScrollBarEnabled(false);
//This statement hides the Vertical scroll bar and does not remove it
        webView.setHorizontalScrollBarEnabled(false);
//This statement hides the Horizontal scroll bar and does not remove it
        webView.loadUrl("https://www.google.com/");
//This statement loads the URL
    }
}
```

Develop a web-browser mobile app in 5 min (cont.)

- In the file AndroidManifest.xml, add permission to access the Internet from within the app:

activity_main.xml | MainActivity.java | AndroidManifest.xml

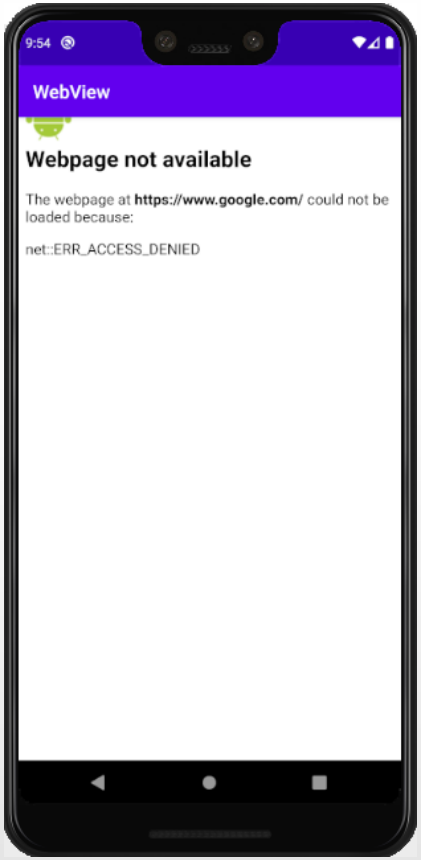
```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.webview">

    <uses-permission android:name="android.permission.INTERNET"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="WebView"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
```

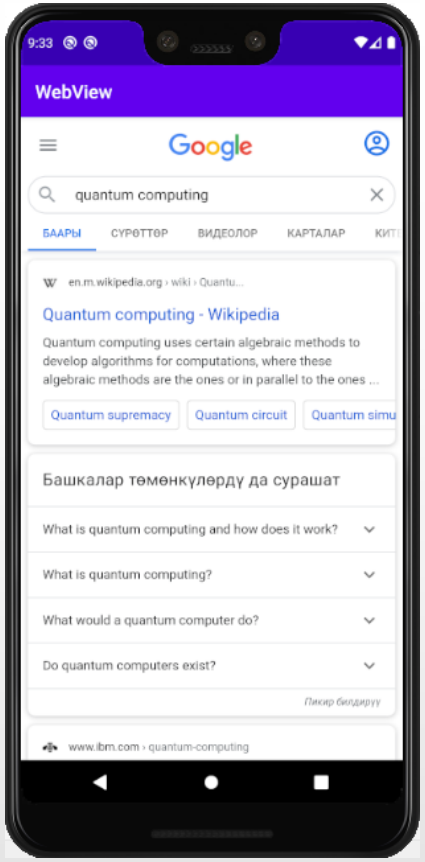
Develop a web-browser mobile app in 5 min (cont.)

- Run the web-browser app:

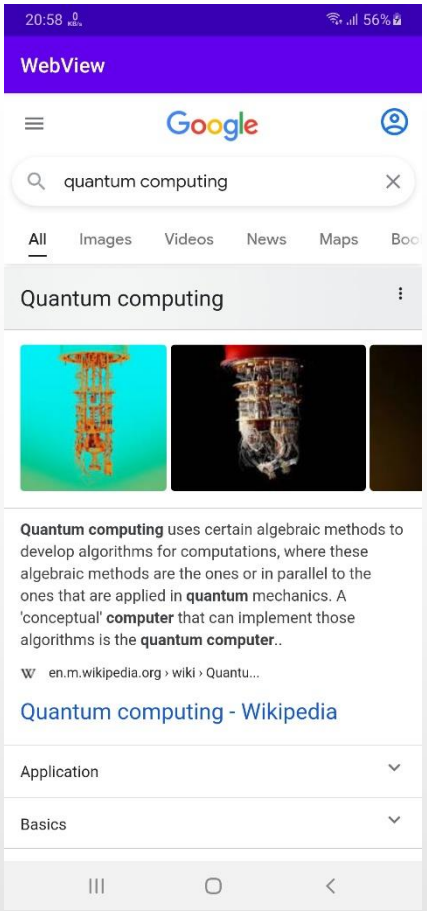
AVD isn't working sometimes :(

Creating a new AVD can solve the problem :)





Samsung Galaxy M31 is working :)



Do you have any questions or comments?

Thank you
for your attention !

In this presentation:

- Some icons were downloaded from flaticon.com and iconscout.com