Using Java 8 on Android

Eduardo Bonet

Motivation

- Java 8 was released over 2 years ago, bringing many new features, but that are not yet availabe on Android.
- The new Jack & Kill toolchain brought us some official Java 8 support, but some features are only available after Android N and others were left aside.

Java 8

- Much less verbose.
- New API's:
  - java.time.*
  - java.util.stream.*
  - java.util.function.*
- Language changes:
  - Default Methods and static methods for interfaces
  - Lambda Functions
  - Method References

Streams

Improved API for dealing with collections, also making parallelization much easier.

```
// Java 7
names = new ArrayList<>();
for (Person p : people) {
    if(p.age > 16)
        names.add(p.name);
}
```

```
// Java 8
names = people.stream()
    .filter(p -> p.age > 16)
    .map(p -> p.name)
    .collect(Collectors.toList());
```

Time API

New API to deal with date and time, fully replacing java.util.Calendar and java.util.Date.

```
// Java7
Date date, datePlusThreeDays;
date = new GregorianCalendar(2014, Calendar.FEBRUARY, 11).getTime()

Calendar c = Calendar.getInstance();
c.setTime(date);
c.add(Calendar.DATE, 3)
datePlusThreeDays = c.getTime()
```

```
// Java 8
LocalDate otherDate, otherDatePlusThreeDays;
otherDate = LocalDate.of(2014, Month.FEBRUARY, 11);
otherDatePlusThreeDays = otherDate.plus(3, ChronoUnit.DAYS);
```

Lambda Functions

Lambda Functions are a easier and cleaner way to create objects that implement a single method interface, i. e., Functors.

```
// Java 7
v.setOnClickListener(new View.OnClickListener() {
    @Override public void onClick(View view) {
        Log.d(TAG, "onClick: ");
    }
});
```

```
// Java 8 Lambda
v.setOnClickListener(view -> Log.d(TAG, "onClick: "));
```

```
Observable.from(people)
    .filter(new Func1<Person, Boolean>() {
        @Override
        public Boolean call(Person person) {
            return person.age > 16;
        }
    })
    .map(new Func1<Person, String>() {
        @Override
        public String call(Person person) {
            return person.name;
        }
    })
    .subscribe(new Action1<String>() {
        @Override
        public void call(String s) {
            System.out.println(s);
        }
    });
```

```
Observable.from(people)
    .filter(person -> person.age > 16)
    .map(person -> person.name)
    .subscribe(s -> System.out.println(s));
```

Method References

Method References are an even simpler version of Lambda Functions, where arguments are simply passed on to another function.

```
Observable.from(people)
    .filter(person -> person.age > 16)
    .map(person -> person.name)
    .subscribe(System.out::println);  // .subscribe(s -> System.out.println(s));
```

Try-with-resources

Better Syntax for objects that must be closed after use (they must implement the Closeable interface).

```
// Java 7
BufferedReader br = new BufferedReader(new FileReader(path));
try {
    System.out.println(br.readLine());
} finally {
    if (br != null) br.close();
}
```

```
// Java 8
try (BufferedReader br = new BufferedReader(new FileReader(path))) {
   System.out.println(br.readLine());
}
```

Default Methods for Interfaces

```
interface Vehicle {
    default void print(){
        System.out.println("I am a vehicle!");
    }
}

class Car implements Vehicle {
    public void print(){
        Vehicle.super.print();
        System.out.println("I am a car!");
    }
}

class Boat implements Vehicle {

}
```

Bringing Java 8 to Android

Jack

- New Android tool that transforms .java into .dex
- Introduced on Android N.
- Suports Lambda Functions and Method References on all Android versions.
- Suports Default Methods and Streams only for Android 24+
- **Does nto support java.time.***

Streams: LightweightStreams

Streams API implementation using Java 7 Collections.

```
List<String> names = Stream.of(people)
    .filter(p -> p.age > 16)
    .map(p -> p.name)
    .collect(Collectors.toList());
```

- Method Count: 719 (1.1.2)

Streams: RxJava

Observables are fundamentally different from Streams (push vs. pull), but similar functionality can be obtained using Observable.from(myCollection).

```
List<String> names = Observable.from(people)
    .filter(p -> p.age > 16)
    .map(p -> p.name)
    .toList().toBlocking().first();
```

- Method Count: 5492 (1.1.8)

Streams

```
// Stream
people.stream()
    .filter(p -> p.age > 16)
    .map(p -> p.name)
    .collect(Collectors.toList());
```

```
// LightweightStreams
Stream.of(people)
    .filter(p -> p.age > 16)
    .map(p -> p.name)
    .collect(Collectors.toList());
```

```
// RxJava
Observable.from(people)
    .filter(p -> p.age > 16)
    .map(p -> p.name)
    .toList().toBlocking().first();
```

java.time.*:

ThreeTenABP

- ThreeTenBP optimized version for Android.
- Same API as java.time.*, making them interchangeable.
- Method Count: 3280

Retrolambda

- Transforms Java 8 code into code compatible with Java 5, 6 and 7.
- Operates during compile time.
- Full support to Lambdas, Try-With-Resources and Method References.
- Partial support to default methods.

RetroLambda vs Jack

Code	Retrolambda 2.1.0	Retrolambda 2.3.0	Jack 24.0.1
`new Runnable() { @Override public void run() { greeter.sayHi(); } }`	2	2	2
`() -> greeter.sayHi()`	6 or 7	4	3
`greeter::sayHi`	4	3 or 4	3

https://speakerdeck.com/jakewharton/exploring-hidden-java-costs-360-andev-july-2016?slide=126

Resumo

	RL	Jack	RxJava	LS	TT
Streams		24+	✔	✔	
Default Methods	Partial	24+			
Lambda	✔	✔			
Method References	✔	✔			
Try-With-Resources	✔	✔			
java.time.*					✔

- RL: Retrolambda, LS: LightweightStreams, TT: ThreeTenABP

References

- Jack
- Jack e Java 8
- Retrolambda
- Lightweight Stream
- ThreeTenABP
- Estudo sobre API para date
- RxJava
- Retrolambda vs Jack

Thanks

Questions?

blog | github | linkedin