Introduction

About Myself

Me: Dr Norman Lee

Office: 1.602.28

Email: norman_lee@sutd.edu.sg

Queries on this course, please put on discussion forum on eDimension

Materials

“Official Material” - Android Developer Fundamentals Version 2

https://developers.google.com/training/courses/android-fundamentals

Correspondence

This table shows the connection between our lessons and the materials in the android developer fundamentals

Our course	**Android Developer Fundamentals Concepts** **All from Version 2 unless stated**
Lesson 0	Lesson 1.1
Lesson 1	Lesson 1.2, Lesson 1.3
Lesson 2	Lesson 2, Lesson 9.1, Lesson 3.2, Lesson 4.1 - 4.2
Lesson 3	Lesson 7.1 - 7.2
Lesson 4	Lesson 4.5, Lesson 9.0
Lesson 5	Nil

Android Device

Getting an **Android device** is highly recommended, Android version 5.0 or higher is recommended.
(To check, Open the Settings app → About Phone → Android Version)

If not, you can use the **emulator** in Android studio. However, be warned that students' previous experience with the emulator is that it can take a long time to load and consume a lot of system resources.

Another option for an emulator is **Genymotion**. However, your experience may vary with this emulator.

We will need you to build a simple android app in the Final Exam. You will be allowed to bring in an Android Phone and we will check that it is in flight mode.

Other Resources

One thing to note

These resources can be out of date for the following reasons

- New libraries get introduced
- Code in exgets deprecated

For this reason, books and online resources get out of date very quickly.

Still they can be useful.

Here are the resources

Please use at your discretion

- Codepath guides to android https://guides.codepath.com/android
- Online course on Udacity https://www.udacity.com/course/new-android-fundamentals--ud851
- Stanford CS193A http://web.stanford.edu/class/cs193a/

My Android Experience

- **Frustrating** - code in tutorials is found to be deprecated
- **Moves fast** - new ways of doing things are released very quickly
- **Lots of applications** - commercial 43” screens run android!

How we’ll learn

- I will first highlight and reinforce the Java that you need to know
- Relevant android features will also be introduced to you
- Then we will code an app to apply those features

Java Revision

The Java you need to know

ArrayList

```
List<Integer> a = new ArrayList<>();
a.add(1);
a.add(2);
a.add(1,3);
a.add(5);
System.out.println(a.toString());
```

What is printed on the screen?

b.socrative.com 593583

Private vs Public

```
class Point2D{
    private double x;
    private double y;

    Point2D(){
        //code not shown
    }

    Point2D(double x, double y){
        this.x = x;
        this.y = y; }

    public double getX() { return x; }
    public double getY() { return y; }
}

class Point3D extends Point2D{

    private double z;

    Point3D( double x, double y, double z ){
    }
}
```

Complete the constructor for Point3D.

Recall Polymorphism, Overriding vs Overloading

We see **overriding** and **overloading** in android very often, so it is good to recap these concepts. To override a method in a super-class, the **method signature** in the subclass must be the same.
The `@Override` annotation allows the compiler to help you check if you have got this condition correct.

Polymorphism allows a variable of a *supertype* to refer to a *subtype*. Thus, in the example below, any variable of `Dog` can refer to its subclasses.
The methods that are available depends on the *declared type*. If a method is overridden in the subclass, in **dynamic binding**, the Java VM decides which method to invoke, starting from the *actual type*.

```java
abstract class Dog{
    public void bark(){ System.out.println("woof"); }
    public void drool(){ System.out.println("drool");}
}

class Hound extends Dog{
    public void sniff(){ System.out.println("sniff ");}
    @Override public void bark(){ System.out.println("growl");}
    public void drool(int time){ System.out.println("drool" + time);}
}
```

Given `Dog g = new Hound();`

- What will you see on the screen for `h.bark()` ?
- What will you see on the screen for `h.drool(1)` ?
- What will you see on the screen for `h.drool()` ?
- What will you see on the screen for `h.sniff()`?

Recall Polymorphism

```
class A {

   void f(int x){System.out.println("Af");}
   void h(int x){System.out.println("Ah");}
}

class B extends A{

   void f(int x){System.out.println("Bf");}
   void g(int x){System.out.println("Bg");}
```

Given

`A x = new B();`

Which of the following can subsequently be executed?

```
x.f(1); //statement (i)
x.g(1); //statement (ii)
x.h(1); //statement (iii)
```

(a) (i) only

(b) (i) and (ii)

(c) (i) and (iii)

(d) (i), (ii) and (iii)

Interfaces (1)

An interface is like a contract for the implementations of classes. It acts as a *supertype* for all classes that implement it.

```
interface I {
   void m(int x);
}

class K implements I{
   void m(int x){System.out.println("m");}
}
```

Which of the following statements is/are legal?

```
(i) K x = new K();
(ii) K x = new I();
(iii) I x = new K();
(iv) I x = new I();
```

(a) (i) only
(b) (i) and (ii)
(c) (i) and (iii)
(d) (i), (ii) and (iii)

Interfaces help in the maintenance of software.

Bearing in mind interface **`I`** and class **`K`** implements **`I`** (defined above)

Which method below is better?

```
void firstMethod(K k){ //do something;}
void secondMethod(I i){ //do something;}
```

A method that takes in an interface is more flexible.
It will be able to accept any object that implements that interface.
Suppose you create a new class implementing **I** that has a better implementation of **m**, you are able to pass it to **`secondMethod`** without having to change its signature.

Interfaces (2)

All method signatures in interfaces are automatically abstract, you do not need to specify the keyword.

```
interface Pokemon{

    void adjustCP(int value);
    void attack();
    void defend();
}

class Bulbasaur implements Pokemon{

    void adjustCP(){
        //code not shown
    }

    void attack(){
        //code not shown
    }

}
```

In the code above, which method(s) does class **`Bulbasaur`** still need to implement?

(a) **`defend`** **(b)** **`adjustCP`**

(c) **`attack`** and **`defend`** **(d)** **`defend`** and **`adjustCP`**

Exceptions (1)

```java
public class TestExceptions1 {

   public static void main(String[] args){
       try{
           f(-1);
           System.out.print("R");
       }catch(Exception e){
           System.out.print("S") ;
       }
   }

   static void f(int x) throws Exception {
      if( x < 0) throw new Exception();
      System.out.print("P");
   }
}
```

In the code above, what is printed out?

(a) S **(b)** PRS

(c) RS **(d)** PR

Exceptions (2)

```
public class TestExceptions2 {

   public static void main(String[] args){
       try{
           f(-1);
           System.out.print("R");
       }catch(Exception e){
           System.out.print("S") ;
       }
   }

   static void f(int x) throws Exception {
       try{
           if( x < 0) throw new Exception();
           System.out.print("P");
       }catch( Exception e){
           System.out.print("Q");
       }
   }
}
```

In the code above, what is printed out?

(a) Q **(b)** S

(c) QR **(d)** QRS

Points to note

- When an **exception** is thrown, the Java runtime searches through the **call stack** to find the first method that will handle the exception.
- The `finally` block is always executed regardless of what happens in the `try` block.
- It is good programming practice to specify exactly the type of exception that is handled in each catch block, as you will have specific details of the exception that occurred. Hence code examples here are not good …