LAB 1

# Object Oriented Concepts

1. Object

Object is an instance of a class, that has states and behaviors. These states can be for example data in fields, and the behavior of object are also called methods.

    MyClass myObj = new MyClass();

    System.out.println(myObj.x);

1. Class

Class is sort of the parent from which individual objects can be created. Classes can contain many variable types such as local variables, instance variables and class variables.

public class MyClass {

  int x = 5;

}

1. Instantiation of object

When you instantiate an object, you basically create an instance of a class.

 MyClass myObj = new MyClass();

1. Visibility

A private member can only be accessed in the same class in which is it declared.

A protected member can be accessed in all classes within the same package and also within subclasses in other packages.

A public member is accessible to all classes.

private int i;

protected int k;

public int l;

1. Member datas / methods

Data member is a variable of an object. For example, if you had a customer object, it’s data members could be name and age. Data member can be for example int id; string name; float salary;

String name = "John";

int myNum;

myNum = 15;

1. Inheritance

Inheritance is a mechanism where an object gets all the properties and behaviors of it’s parent object. Inheritance is a key concept in OOP.

    class Vehicle {

      protected String brand = "Ford";

      public void honk() {

        System.out.println("Tuut, tuut!");

      }

    }

    class Car extends Vehicle {

      private String modelName = "Mustang";

      public static void main(String[] args) {

        Car myFastCar = new Car();

        myFastCar.honk();

        System.out.println(myFastCar.brand + " " + myFastCar.modelName);

      }

    }

1. Interface

Interface in Java is an abstract class with which you can group related methods with empty bodies. The goal of this is to achieve abstraction (only showing essential information to the user).

    interface Animal {

      public void animalSound(); // interface method (does not have a body)

      public void run(); // interface method (does not have a body)

    }

1. Polymorphism

You achieve polymorphism by having multiple classes that are related to each other by inheritance.

    class Animal {

      public void animalSound() {

        System.out.println("The animal makes a sound");

      }

    }

    class Pig extends Animal {

      public void animalSound() {

        System.out.println("The pig says: wee wee");

      }

    }

1. Overriding

Overriding allows subclass or a child class to make a specific implementation of a method that is used in their parent class.

    class Animal {

      public void animalSound() {

        System.out.println("The animal makes a sound");

      }

    }

    class Pig extends Animal {

      public void animalSound() {

        System.out.println("The pig says: wee wee");

      }

    }

1. Abstract classes

Abstract class is restricted, so that it cannot be used to create new objects. To access it , it must be inherited from another class.

    abstract class Animal {

      public abstract void animalSound();

      public void sleep() {

        System.out.println("Zzz");

      }

    }

# Android Fundamental Concepts

1. What programming languages can you use for Android development?

You can use Java, Kotlin and C++.

1. What is .apk file?

APK is an android package file, which is basically your code compiled together with other data and resource files.

1. How does Android system run apps?

Normally every app runs in its own Linux process. Android system assigns each app with a unique Linux user ID. System sets permissions for all files in an app so that only the ID assigned to the app can access the files. Each process also has their own VM, which means each app’s code runs in isolation from other apps.

1. Four types of Android components?

* Activities

Acitivity is what interacts with the users. It is a single screen with a UI. An app has many activies, for example email app could have activity for showing new emails, another one for making emails, and another for reading them.

* Services

This is a component that runs in the background performing long-running operations or work for remote processes. Service doesn’t provide a UI. Let’s say the user has Spotify open, and is using another app to read the news. Spotify will play music in the background while the user is reading the news, which is a service.

* Broadcast receivers

Allows the system to deliver events to the app. For example, an app can schedule an alarm to make a notification that will notify the user about an upcoming event. When the alarm is delivered to the receiver, the app does not have to remain running until the alarm goes off.

* Content providers

This component manages app data that can be stored in file system, in a database, on the web, or any other persistent storage which your app can access. Through the provider, other apps can send queries for the data or modify it, if the provider allows. Let’s say there is a provider for managing user’s contact information. Any app with correct permissions can query the provider, to read and write information about a person.

1. What is manifest file, what is its purpose?

In your app, every component that exists must be declared in the AndroidManifest.xml file. Before Android starts your app, it will check that all the components exist.

1. What are resources? Why they are needed?

Resources are pretty much everything and anything the app needs to run, that is not code. Things such as; audio, images, visual presentation of the app such as animations, styles, menus etc. Using resources is smart because it makes it easy to update different things in your app without modifying your code. It also makes it easier to optimize your app, for example for different languages and screen sizes.