Lab 1 This word document contains lab **1.1** and **1.3**

1 OOP concepts in Java

1. **Object**

Object is an instance of a class; it contains both data and the function, which operates the data

Nanny mPoppins;

1. **Class**

Class is a group of similar entities. For example, Class Children has only the children in it, not the nanny’s or daycare managers

**public class** Children {}

1. **Instantiation of object (creating an object)**

Calling a constructor of a class, which creates an instance or object of the type of the class.

Nanny mPoppins = **new** Nanny();

1. **Visibility (public / private / protected)**

Private methods or variables can be accessed only within their class. Public methods and variables can be accessed by any class in any package, provided that the class itself can be seen. Protected methods or variables are accessed in the class and subclasses of that class.

**protected void** onCreate(Bundle savedInstanceState) {}

**private** String **cryingSensitivity**;

**public** String getCryingSensitivity() {}

1. **Member datas / methods**

Member data means a variable of an object, for example object player has a member data “name”. Method is same as a function in JavaScript. Method performs an action on data and returns a result or not.

mPoppins.setPersonnelNr(12);  
mPoppins.setName(**"Mary Poppins"**);

1. **Inheritance**

A subclass inherits the object properties from the superclass, meaning that one object acquires the behaviors and properties from its parent class

**public class** MainActivity **extends** AppCompatActivity {}

1. **Interface**

Interface looks like a class, but it is not a class. Interface can have variables and methods, but the methods are by default abstract. Methods don’t have a body, so they have to be implemented by a class before you can access them. That class has to implement all methods of the interface. You can implement more than one interface in a class.

**public interface** DaycareManager {  
}

1. **Polymorphism**

Single action can be performed in multiple ways when using polymorphism. Variable, function or object can take multiple forms. For example, in English language some word can have multiple meanings depending on the context. This is equivalent to what polymorphism means in programming: the context definiens the meaning of variable, function or object. There are two types of polymorphism in Java: overloading and overriding. There are several differences between these two, e.g. overloading is example of compile time polymorphism and overriding is example of runtime polymorphism.

abstract public class Person {

public void add() {

//your person code

}

}

public class Children extends Person {

public void add() {

//your modified children code

}

}

1. **Overriding**

Declaring a method in subclass which is already present in superclass, is called method overriding. Subclass can give its own implementation of the method which is already provided by the superclass

@Override  
**public** String greet() {}

1. **Abstract classes**

Abstract class can have concrete methods (regular methods with body) and abstract methods (methods without a body). You cannot instantiate abstract class. Abstract class is needed, if method in all subclasses require overriding of the superclass.

**abstract public class** Person {  
 **private** String **Name**;  
 **private** String **socialSecurityNumber**;  
}

3 Android fundamental concepts

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

Java and Kotlin. You can also use C++ with Android Studio.

1. **What is .apk file?**

An APK (Android package) file consists of a ZIP archive that contains all the files that comprise your app. These files include Java class files, resource files, and a file containing compiled resources.

1. **How Android system runs apps?**

Every app runs in its own Linux process. The Android system starts the process when any of the app's components need to be executed, and then shuts down the process when it's no longer needed or when the system must recover memory for other apps.

1. **Name four types of Android components. Describe each.**

Activities, services, broadcast receivers and content providers.

Activity is a single UI screen of the app, which interacts with the user. For example in email, mailbox view is one activity, writing new email is another activity and reading email is third activity.

A service is a general-purpose entry point for keeping an app running in the background for all kinds of reasons. It is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity.

Broadcast receiver enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements. Because broadcast receivers are another well-defined entry into the app, the system can deliver broadcasts even to apps that are not currently running. So, for example, an app can schedule an alarm to post a notification to tell the user about an upcoming event.

Content provider manages a shared set of app data that you can store in the file system, in a SQLite database, on the web, or on any other persistent storage location that your app can access. Through the content provider, other apps can query or modify the data if the content provider allows it. For example, the Android system provides a content provider that manages the user's contact information.

1. **What is manifest file and what is its purpose?**

The manifest file describes essential information about your app to the Android build tools, the Android operating system, and Google Play. It lists the name, version, access rights, and referenced library files of the app.

1. **What are resources? Why they are needed?**

Resources are the additional files and static content that your code uses, such as bitmaps, layout definitions, user interface strings, animation instructions, and more.