Al-Imam Mohammad ibn Saud Islamic University
College of Computer and Information Sciences
Computer Science Department

CS392: Software Engineering 2
1st Semester 1443H / Fall 2021

Quality assessment Report of Alinma Bank App

Workshop 1

Name	ID	Email
Sadeem Faisal Alqahtani	440021429	sfaalqahtani29@sm.imamu.edu.sa
Sarah Khalid Alaradi	440023365	skmaloridi@sm.imamu.edu.sa
Shoug Ali Alsuhaibani	440022732	ssuhaibani@sm.imamu.edu.sa
Sarah Abdullah Alsarami	440020811	samalsarami@sm.imamu.edu.sa

Section: 371

Supervised by: Dr.Lamees Alhazaa

Abstract

In this report, we were able to achieve the goal of a workshop focused on quality assessment for a local banking application, which was to provide relevant insights and improve the application's quality by performing a static analysis with the MobSF tool. This report looked at the overall quality of the Alinma Bank application in terms of clean code, code documentation, and security problems including application permissions, network security, and manifest analysis.

Table of Contents

Introduction

In March 2006, three Saudi government funds established Alinma Bank as an Islamic retail and corporate bank. Its initial owners were the Saudi Arabian General Investment Fund (GIF), the Saudi Arabian General Retirement Fund (GRF), and the Saudi Arabian General Organization for Social Insurance (GOSI). The company's headquarters is located in Riyadh, Saudi Arabia.
To maintain delivering the best services to their customers with the technological revolution pace that's happening in Saudi Arabia, Alinma provided an online banking service that simplifies their customers' financial lives and gives them access to a wide range of transactions and services at their convenience.

This report will outline the studies and the quality assessments of Alinma Bank app in order to provide profound insights and improve the overall quality of the app. This well be achieved by performing a static analysis on the Android version of the app by using the MobSF tool, and automating some of the source code quality analysis through the use of Codacy.

Going through this workshop taught us that even critical apps that were developed by big companies can have flaws. Maintenance is a very important part in the life cycle of the app development process because as we saw these minor problems that we found could cause the app a big failure that could have been avoided with maintenance and proper documentation.

Alinma Static Analysis Results

Starting with a walkthrough of how we retrieved application's data (codes, docs, designs, etc.). By using the MobSF tool, we download a docker and uploaded an APK file of the Alinma banking app. The results given by the tool were: Security score, tracker detection, size of the file, latest app version and the different analysis options. Such analysis options were network, manifest, code, binary and security analysis which was not available due to security measures. After using the tool, many analysis titles were found. We have assessed each one separately.

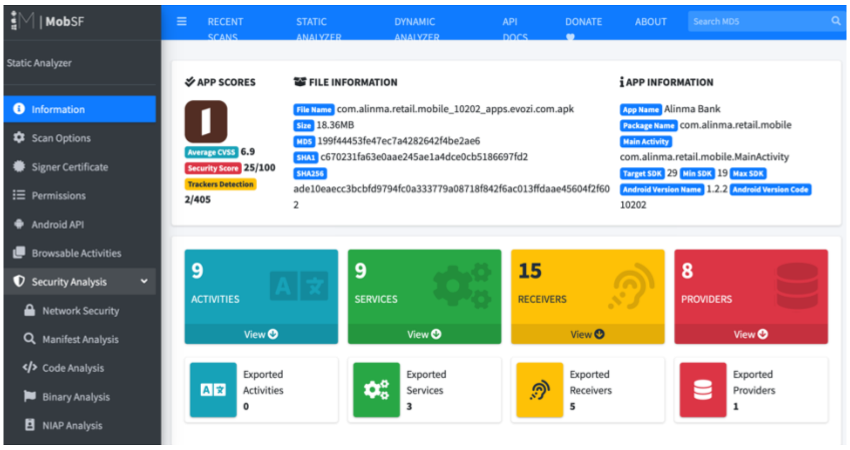
Going into detail of how we downloaded the tool, A docker was downloaded and some command lines were used in the terminal to help run the tool.

The two commands below were used to run the tool:

```
1.docker pull opensecurity/mobile-security-framework-mobsf
```

```
2.docker run -it --rm -p 8000:8000 opensecurity/mobile-security-framework-mobsf:latest
```

After the commands were run, an IP address was given that was associated for the MobSF tool. An APK file was download via google play and uploaded unto the tool. After a while of processing the results page was finally loaded.



APP SCORES

Average CVSS 6.9
Security Score 25/100
Trackers Detection 2/405

FILE INFORMATION

File Name com.alinma.retail.mobile_10202_apps.evozi.com.apk
Size 18.36MB
MD5 199f44453fe47ec7a4282642f4be2ae6
SHA1 c670231fa63e0aae245ae1a4dce0cb5186697fd2
SHA256 ade10eaecc3bcbfd9794fc0a333779a08718f842f6ac013ffdaae45604f2f602

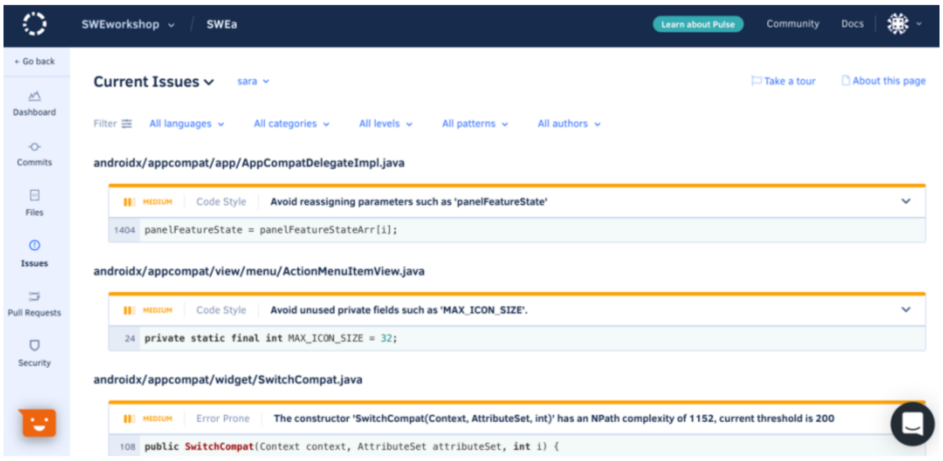
APP INFORMATION

App Name Alinma Bank
Package Name com.alinma.retail.mobile
Main Activity com.alinma.retail.mobile.MainActivity
Target SDK 29 Min SDK 19 Max SDK
Android Version Name 1.2.2 Android Version Code 10202

9 ACTIVITIES View
9 SERVICES View
15 RECEIVERS View
8 PROVIDERS View

Exported Activities 0
Exported Services 3
Exported Receivers 5
Exported Providers 1

For the clean code analysis, we used Codacy to run the code analysis with the help of GitHub. A repository of the Alinma java code was uploaded to GitHub and linked to Codacy. After running the tool, issues of the code were available for preview.



androidx/appcompat/app/AppCompatDelegateImpl.java

MEDIUM | Code Style | Avoid reassigning parameters such as 'panelFeatureState'

```
1404 panelFeatureState = panelFeatureStateArr[i];
```

androidx/appcompat/view/menu/ActionMenuItemView.java

MEDIUM | Code Style | Avoid unused private fields such as 'MAX_ICON_SIZE'.

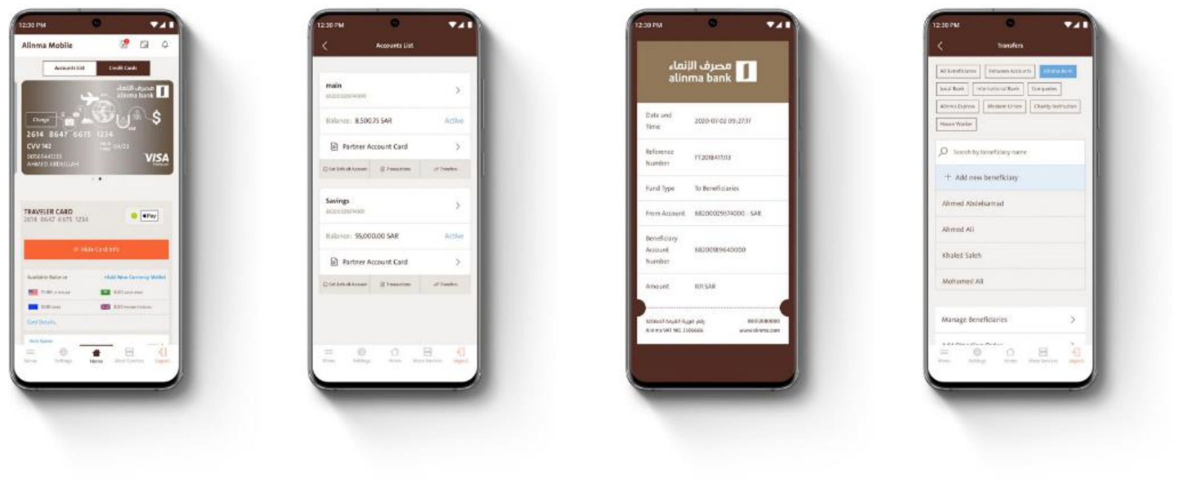
```
24 private static final int MAX_ICON_SIZE = 32;
```

androidx/appcompat/widget/SwitchCompat.java

MEDIUM | Error Prone | The constructor 'SwitchCompat(Context, AttributeSet, int)' has an NPath complexity of 1152, current threshold is 200

```
108 public SwitchCompat(Context context, AttributeSet attributeSet, int i) {
```

Before going in details about the issues we found, we would like to do some design analysis for the app pointing out some of the design issues in it.



Alinma bank application provides a set of features that allow users to securely conduct financial transactions and manage their accounts – anytime, anywhere.
The interface of the application is modern and reflects their entity, the colors, fonts, buttons, layout structure and photograph style are consistent with a compatible resolution size to android devices. A simple GUI that makes it easy to use, but we should point out that there is no guidance as a critical system like bank must be clear how to use it. Account cards are visible so clients can immediately see all their accounts. Massive functionalities in the homepage are unnecessary.
The app has some issues that makes it not efficient enough to users to perform their tasks quickly through the easiest process, such as when logging in or purchases received the OTP SMS is delayed too long, every weekend they undergo maintenance without a previous announcement. Also, it takes a long time to log in to the app beside the slow navigation between the interfaces.

Security Issues

The following part is regarding the different analysis results given by the MobSF tool which were regarding the security part of the app. We have taken examined each segment thoroughly. We stated the problems within each title and some solutions that could be done to enhance the app. The Segments include: Analysis of the Application permissions, network security, manifest analysis and so on.

1. Application Permissions

Via android static analysis report for Alinma bank we will discuss permissions as it is:

<u>**Dangerous:**</u>
android.permission.WRITE_EXTERNAL_STORAGE: This permission is allowing an application to write and modify, read on external memory, it is dangerous permission, but at first the user is asked if he allows access or not. Since it is rated dangerous it means that the user will be asked permission the best practices used to request dangerous permissions.
android.permission.CAMERA: it is dangerous because this permission allows application to take pictures and videos with camera, collect images, and seeing at any time, in some cases, user privacy may to break through this feature by malicious Applications, but at first the user is asked if he allows access or not Since it is rated dangerous it means that the user will be asked permission the best practices used to request dangerous permissions.
android.permission.ACCESS_COARSE_LOCATION: In this permission, when you access the coarse location sources such as the database of the mobile phone network, then phone location is determined approx., and malicious applications can use this to determine your Location approx.
android.permission.ACCESS_FINE_LOCATION: When this permission is available, the phone will be located globally via GPS, which may allow malicious applications to determine your location well, also consumes an additional battery power.
This feature is a unique feature of mobile applications and bringing it to an application is important and causes a lot of facilitation and contextual experience for users, but it can be exploited in the wrong way and is used maliciously to determine your location and saves and is exploited by malicious users and in order to preserve privacy and user location data, location permissions are requested ,Since it is rated dangerous it means that the user will be asked permission the best practices used to request dangerous permissions.

<u>**Normal:**</u>
android.permission.USE_FINGERPRINT: It is a good feature and bring to application that a distinct experience for users, but this constant was deprecated in API level 28. Application in this case Applications should request USE_BIOMETRIC instead.
android.permission.RECEIVE_BOOT_COMPLETED: this permission is allowing the app to have itself started as soon as the system has finished booting. This can make it take longer to start the phone and allow the app to slow down the overall phone by always running, which means that the problem of slowing down the device is created from allowing this permission in the application, Because the processor is under loading pressure apps.
android.permission.INTERNET (full Internet access) this is permission it normal it does not need to asked user if allows or not it just Allows an application to create network sockets, And the same with the rest of the normal permissions unless there is a problem.

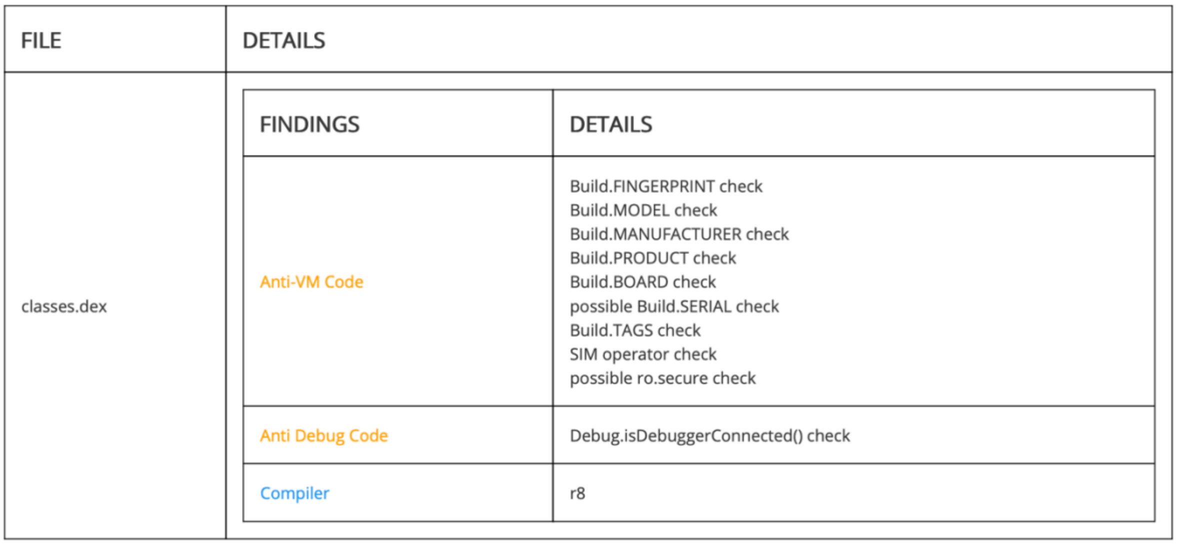
android.permission.ACCESS_NETWORK_STATE: Allows an application to view the status of all networks if it ((wifi ,3g,4g) also you can check if the network is connected or not.
android.permission.FLASHLIGHT: (control flashlight) allows the application to control the flashlight without activating the camera hardware.
com.sec.android.provider.badge.permission.READ:(Show notification count on app) Show notification count or badge on application launch icon for Samsung phones.
com.sec.android.provider.badge.permission.WRITE: (Show notification count on app) Show notification count or badge on application launch icon for Samsung phones.
com.htc.launcher.permission.READ_SETTINGS: (Show notification count on app) Show notification count or badge on application launch icon for htc phones
com.htc.launcher.permission.UPDATE_SHORTCUT: (Show notification count on app) Show notification count or badge on application launch icon for htc phones.
com.sonyericsson.home.permission.BROADCAST_BADGE (Show notification count on app) Show notification count or badge on application launch icon for sony phones.
com.sonymobile.home.permission.PROVIDER_INSERT_BADGE:(Show notification count on app) Show notification count or badge on application launch icon for sony phones.
com.anddoes.launcher.permission.UPDATE_COUNT:(Show notification count on app) Show notification count or badge on application launch icon for apex.
com.majeur.launcher.permission.UPDATE_BADGE :(Show notification count on app) Show notification count or badge on application launch icon for solid.
com.huawei.android.launcher.permission.CHANGE_BADGE:(Show notification count on app) Show notification count or badge on application launch icon for huawei phones
com.huawei.android.launcher.permission.READ_SETTINGS :(Show notification count on app) Show notification count or badge on application launch icon for huawei phones
com.huawei.android.launcher.permission.WRITE_SETTINGS:(Show notification count on app) Show notification count or badge on application launch icon for huawei phones
android.permission.READ_APP_BADGE :(show app notification) Allows an application to show app icon badges
com.oppo.launcher.permission.READ_SETTINGS: (Show notification count on app) Show notification count or badge on application launch icon for oppo phones.
com.oppo.launcher.permission.WRITE_SETTINGS: (Show notification count on app) Show notification count or badge on application launch icon for oppo phones.
android.permission.WAKE_LOCK: (prevent phone from sleeping) Allows an application to prevent the phone from going to sleep.
android.permission.VIBRATE : (control vibrator) Allows the application to control the vibrator.

Unknown:
There are unknown permissions and they were written in the report, but it is not clear what they are and what they do, such that **com.amazon.device.messaging.permission.RECEIVE com.alinma.retail.mobile.permission.RECEIVE_ADM_MESSAGE , me.everything.badger.permission.BADGE_COUNT_READ , me.everything.badger.permission.BADGE_COUNT_WRITE, com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVIC.**

Signature:
c2dm.permission. RECEIVE: Permission for cloud to device messaging, it is received data from Internet allows apps to accept cloud to device messages sent by the app's service. (Using this service will incur data usage. Malicious apps could cause excess data usage.).

2. APKID Analysis

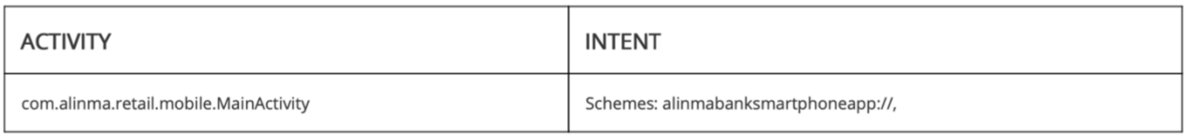
This analysis was not familiar to any team member, so it won't be discussed. This is a sample of it.



FILE	DETAILS	
	FINDINGS	DETAILS
classes.dex	Anti-VM Code	Build.FINGERPRINT check Build.MODEL check Build.MANUFACTURER check Build.PRODUCT check Build.BOARD check possible Build.SERIAL check Build.TAGS check SIM operator check possible ro.secure check
	Anti Debug Code	Debug.isDebuggerConnected() check
	Compiler	r8

3. Browsable Activities

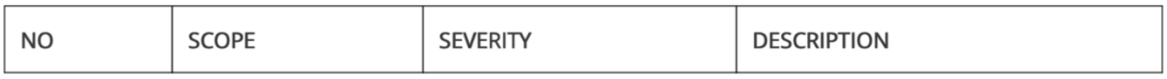
This title also was unfamiliar to any team member, so it won't be discussed. This is a sample of it.



ACTIVITY	INTENT
com.alinma.retail.mobile.MainActivity	Schemes: alinmabanksmartphoneapp://,

4. Network Security

The network analysis was not provided, most likely for security purposes. This is a sample of it.



NO	SCOPE	SEVERITY	DESCRIPTION

5. Manifest Analysis

In the manifest Analysis section of the report, several issues between medium to high severity appeared, some of it was understood by the team and there are some out of the team's knowledge, but it will be discussed for the cause of explaining the reported issues.

Starting with **Clear text traffic,** which is a high severity issue were the app intends to use cleartext network traffic, such as cleartext HTTP, FTP stacks, DownloadManager, and MediaPlayer. The default value for apps that target API level 27 or lower is "true". Apps that target API level 28 or higher default to "false". The key reason for avoiding cleartext traffic is the lack of confidentiality, authenticity, and protections against tampering; a network attacker can eavesdrop on transmitted data and also modify it without being detected.
Followed by Medium severity issue **Application Data can be Backed up flag is missing**, the flag [android: allowBackup] should be set to false. By default, it is set to true and allows anyone to backup your application data via adb. It allows users who have enabled USB debugging to copy application data off of the device.

There was a number of issues (with a high severity rate) regarding a various Broadcast Receivers used in the app, they either were not Protected or Protected by a permission, but the protection level of the permission should be checked.

When we say **Broadcast Receiver is not Protected** it means the Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Broadcast Receiver is explicitly exported. And a **Broadcast Receiver is Protected by a permission, but the protection level of the permission should be checked** indicates that A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analyzed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission.
We also encountered high severity issues with **Services that were not Protected with an existent of intent-filter** meaning a Service is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. The presence of intent-filter indicates that the Service is explicitly exported.

Finally, a **Content Provider is not Protected** is a high severity issue, the Content Provider is found to be shared with other apps.

6. Code Analysis

We discover weaknesses in source code that it leads to vulnerabilities depending on the code analysis report, some of these issues just warning and the others are highly severe.

The App logs information for some reasons to create log files on a mobile device, such as keeping track of crashes, errors, and usage statistics. However, logging sensitive data may expose the data to attackers or malicious applications, and it may also violate user confidentiality.

When you write an application you should make sure that it does not have cryptographic algorithms and protocols that have significant known weaknesses or are otherwise insufficient for modern security requirements. Algorithms that were previously regarded as secure may become insecure over time; as a result, it's critical to review current best practices and update configurations as needed. This App uses the encryption mode CBC with PKCS5/PKCS7 padding and this configuration is vulnerable to padding oracle attacks and also the app has MD5 is a weak hash known to have hash collisions that occur when an attempt to find two input strings of a hash function that produce the same hash result. Algorithms with known weaknesses should be replaced and need to read about algorithm choice for more secure alternatives.

Every Android-compatible device supports Files saved to external storage that are world-readable. Usernames, passwords, keys, and other sensitive information may be hardcoded in files. Data written to External Storage can be read by any app. So, the external storage can be used by an attacker to allow for arbitrary control of the application in some cases. Finally, when storing sensitive data—data that shouldn't be accessible from any other app—use internal storage, preferences, or a database. Internal storage provides the advantage of keeping data hidden from users.

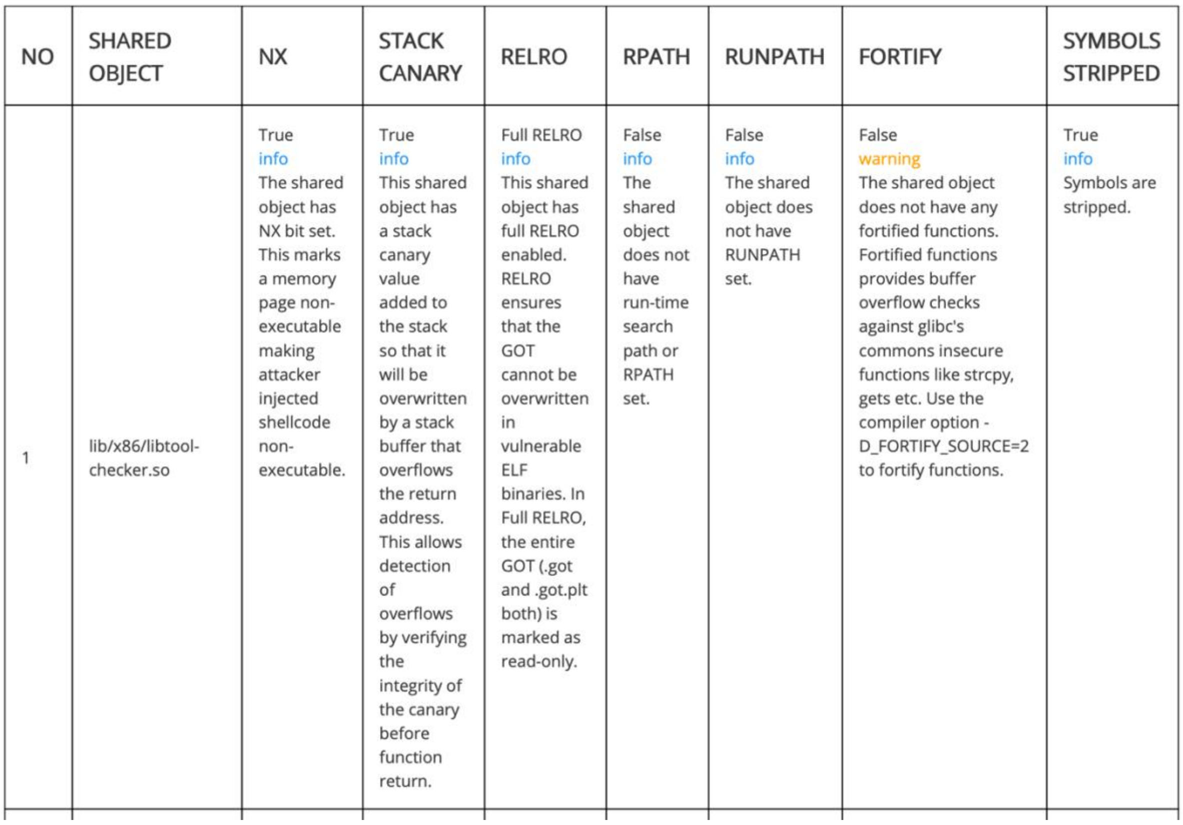
Since a rooted device is much more at risk of being compromised, it's crucial to be aware of it. And this App may have root detection capabilities. Also, this App may request root (Super User) privileges that allow users to control the root settings.

The app uses SQLite Database and executes the raw SQL query. A SQL injection attack involves integrating SQL commands into input data, mimicking the syntax of a predefined SQL command. A successful SQL injection attack allows the attacker to read or write to the database and possibly execute administrative commands, depending on the permissions granted by the server. Untrusted user input in raw SQL queries can cause SQL Injection. Also, sensitive information should be encrypted and written to the database. To prevent this, make sure that input validation and parameterized queries including prepared statements.

This App copies data to clipboard. The clipboard is accessible system-wide and is therefore shared by apps. Malicious apps can take advantage of this sharing to access sensitive data stored in the clipboard. Be aware sensitive data should not be copied to clipboard.

7. **Shared library Binary Analysis**

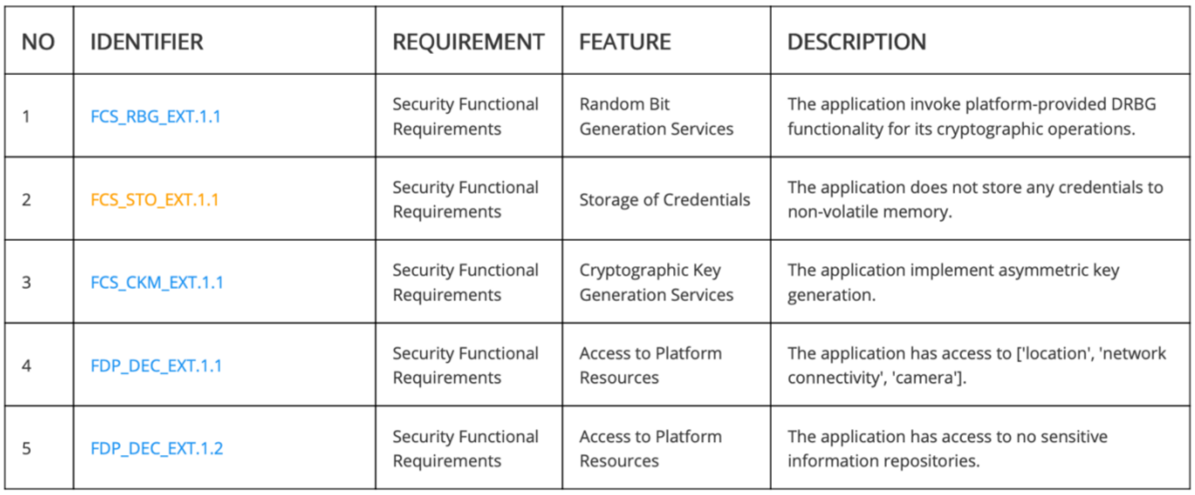
This title also was unfamiliar to any team member, so it won't be discussed. This is a sample of it.



NO	SHARED OBJECT	NX	STACK CANARY	RELRO	RPATH	RUNPATH	FORTIFY	SYMBOLS STRIPPED
1	lib/x86/libtool-checker.so	True info The shared object has NX bit set. This marks a memory page non-executable making attacker injected shellcode non-executable.	True info This shared object has a stack canary value added to the stack so that it will be overwritten by a stack buffer that overflows the return address. This allows detection of overflows by verifying the integrity of the canary before function return.	Full RELRO info This shared object has full RELRO enabled. RELRO ensures that the GOT cannot be overwritten in vulnerable ELF binaries. In Full RELRO, the entire GOT (.got and .got.plt both) is marked as read-only.	False info The shared object does not have run-time search path or RPATH set.	False info The shared object does not have RUNPATH set.	False warning The shared object does not have any fortified functions. Fortified functions provides buffer overflow checks against glibc's commons insecure functions like strcpy, gets etc. Use the compiler option -D_FORTIFY_SOURCE=2 to fortify functions.	True info Symbols are stripped.

8. **NIAP Analysis v1.3**

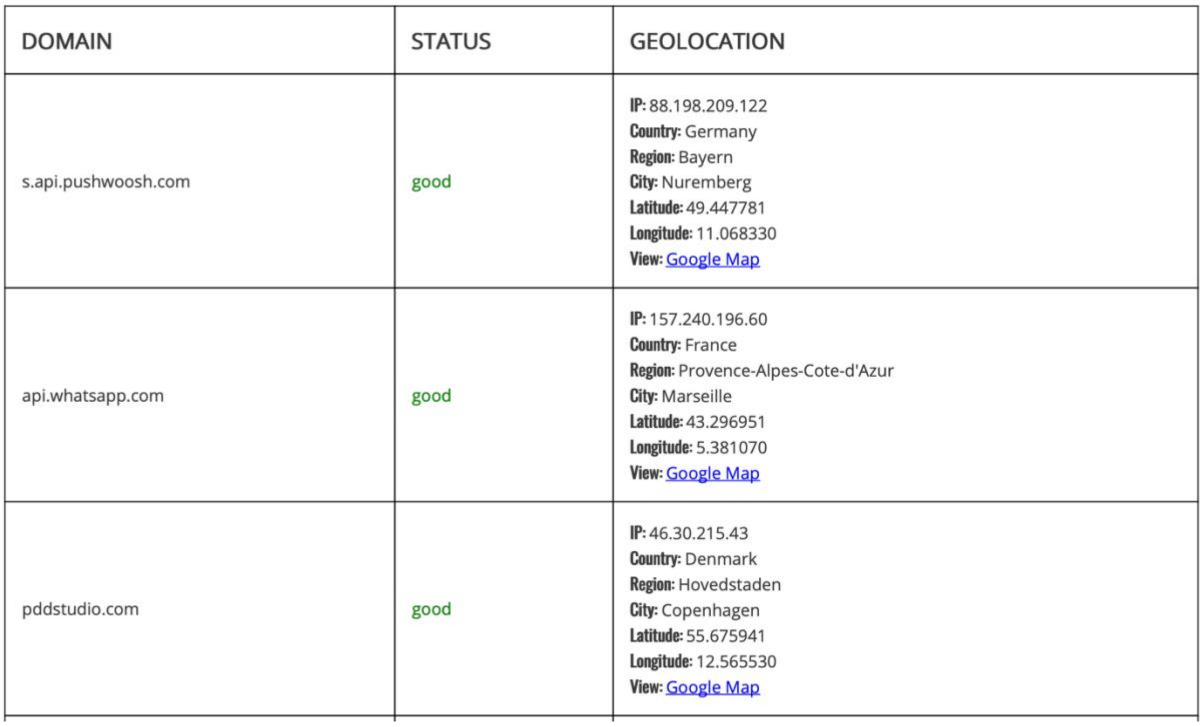
This title also was unfamiliar to any team member, so it won't be discussed. This is a sample of it.



NO	IDENTIFIER	REQUIREMENT	FEATURE	DESCRIPTION
1	FCS_RBG_EXT.1.1	Security Functional Requirements	Random Bit Generation Services	The application invoke platform-provided DRBG functionality for its cryptographic operations.
2	FCS_STO_EXT.1.1	Security Functional Requirements	Storage of Credentials	The application does not store any credentials to non-volatile memory.
3	FCS_CKM_EXT.1.1	Security Functional Requirements	Cryptographic Key Generation Services	The application implement asymmetric key generation.
4	FDP_DEC_EXT.1.1	Security Functional Requirements	Access to Platform Resources	The application has access to ['location', 'network connectivity', 'camera'].
5	FDP_DEC_EXT.1.2	Security Functional Requirements	Access to Platform Resources	The application has access to no sensitive information repositories.

9. **Domain Malware Check**

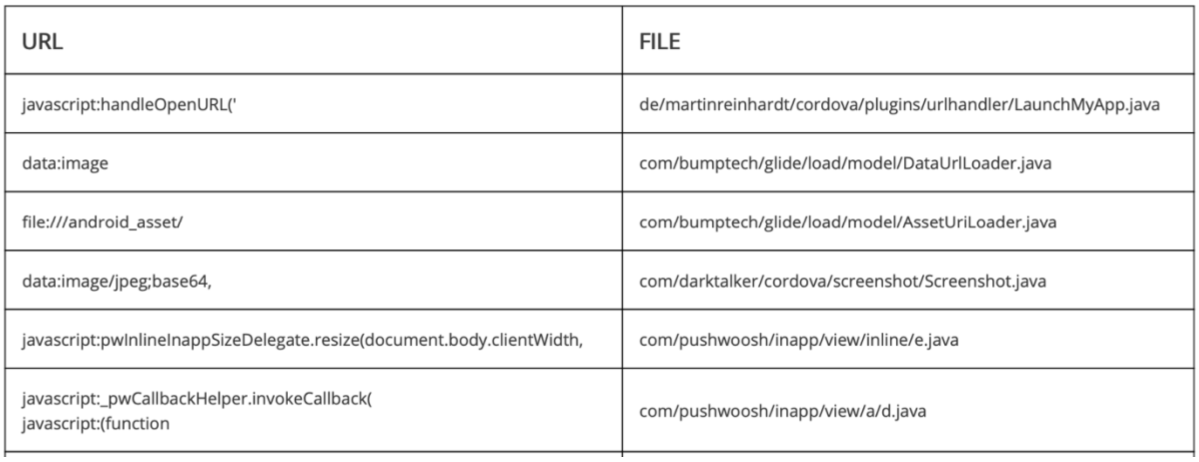
This title also was unfamiliar to any team member, so it won't be discussed. This is a sample of it.



DOMAIN	STATUS	GEOLOCATION
s.api.pushwoosh.com	good	**IP:** 88.198.209.122 **Country:** Germany **Region:** Bayern **City:** Nuremberg **Latitude:** 49.447781 **Longitude:** 11.068330 **View:** Google Map
api.whatsapp.com	good	**IP:** 157.240.196.60 **Country:** France **Region:** Provence-Alpes-Cote-d'Azur **City:** Marseille **Latitude:** 43.296951 **Longitude:** 5.381070 **View:** Google Map
pddstudio.com	good	**IP:** 46.30.215.43 **Country:** Denmark **Region:** Hovedstaden **City:** Copenhagen **Latitude:** 55.675941 **Longitude:** 12.565530 **View:** Google Map

10. **URLs**

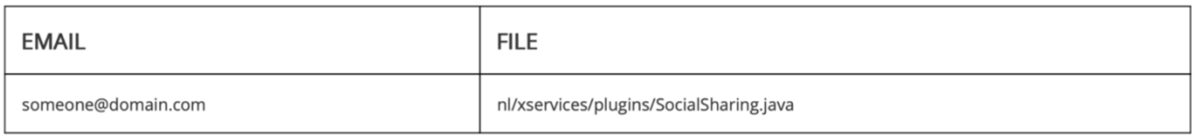
This title also was unfamiliar to any team member, so it won't be discussed. This is a sample of it.



URL	FILE
javascript:handleOpenURL('	de/martinreinhardt/cordova/plugins/urlhandler/LaunchMyApp.java
data:image	com/bumptech/glide/load/model/DataUrlLoader.java
file:///android_asset/	com/bumptech/glide/load/model/AssetUriLoader.java
data:image/jpeg;base64,	com/darktalker/cordova/screenshot/Screenshot.java
javascript:pwInlineInappSizeDelegate.resize(document.body.clientWidth,	com/pushwoosh/inapp/view/inline/e.java
javascript:_pwCallbackHelper.invokeCallback(javascript:(function	com/pushwoosh/inapp/view/a/d.java

11. Emails

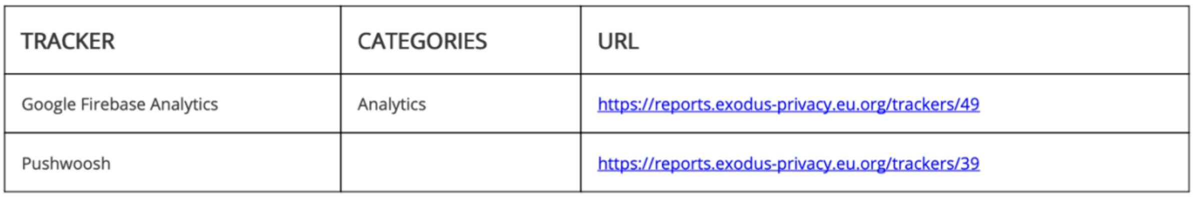
This title also was unfamiliar to any team member, so it won't be discussed. This is a sample of it.



EMAIL	FILE
someone@domain.com	nl/xservices/plugins/SocialSharing.java

12. Trackers

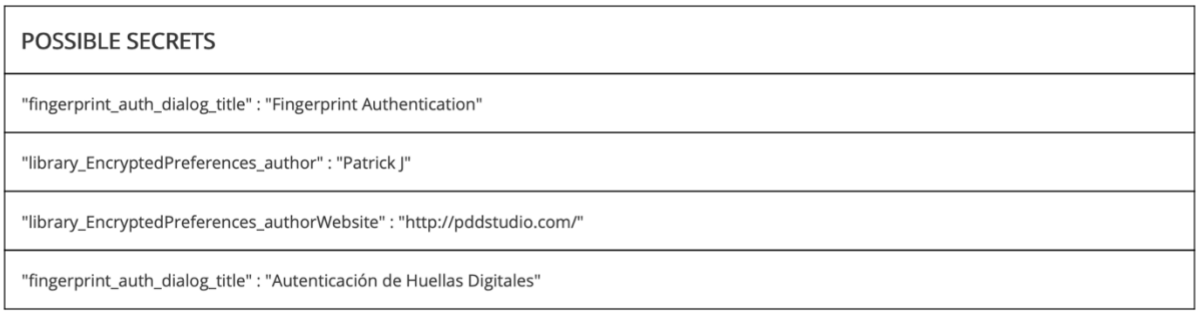
This title also was unfamiliar to any team member, so it won't be discussed. This is a sample of it.



TRACKER	CATEGORIES	URL
Google Firebase Analytics	Analytics	https://reports.exodus-privacy.eu.org/trackers/49
Pushwoosh		https://reports.exodus-privacy.eu.org/trackers/39

13. HardCoded Secrets

This title also was unfamiliar to any team member, so it won't be discussed. This is a sample of it.



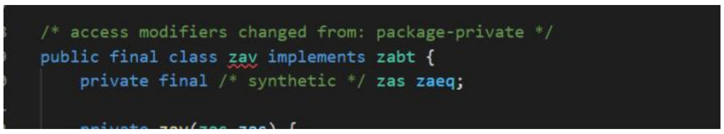
POSSIBLE SECRETS
"fingerprint_auth_dialog_title" : "Fingerprint Authentication"
"library_EncryptedPreferences_author" : "Patrick J"
"library_EncryptedPreferences_authorWebsite" : "http://pddstudio.com/"
"fingerprint_auth_dialog_title" : "Autenticación de Huellas Digitales"

Clean Code Analysis

With the help of Codacy and some manual work, code issues were brought up. We have compared some code patterns related to the clean code by using the clean code handbook.

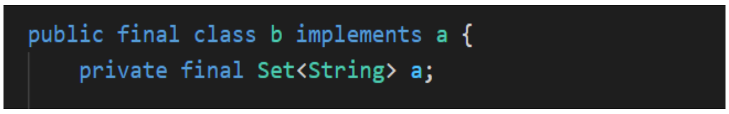
Meaningful Names

Names are everywhere in the software, classes, functions and variables. So, they have a great importance in the continuity of the code and modification and development by other developers. Choosing good names takes time but saves more than it takes, one of the importance of a good name is that it be easy, clear, and understandable to Everyone who reads your code (including you).
there are some restrictions on names in many respects:



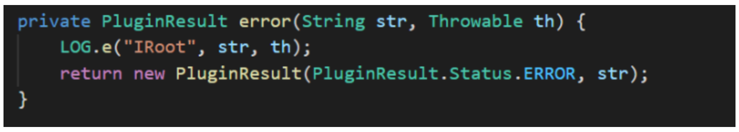
```
/* access modifiers changed from: package-private */
public final class zav implements zabt {
    private final /* synthetic */ zas zaeq;
```

Class names should be nouns or noun phrases that denote something because it is impossible for these non-significant characters to be translated by other programmers and they do not have to.



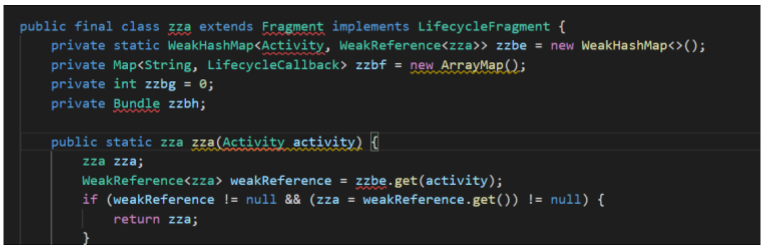
```
public final class b implements a {
    private final Set<String> a;
```

The problem of a single letter causes confusion into what is the reason for choosing this letter over others why you don't choose k or f? and what does it mean? and there is no logical reason.



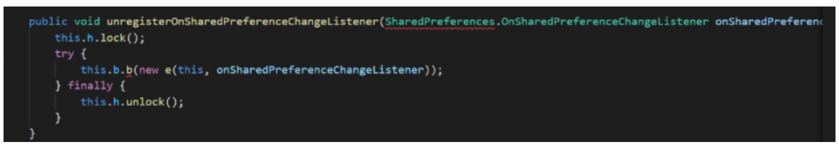
```
private PluginResult error(String str, Throwable th) {
    LOG.e("IRoot", str, th);
    return new PluginResult(PluginResult.Status.ERROR, str);
}
```

The method name must be verb and verb phrases in her error it just words you can be prefixed with is or set, according to the JavaBean standard.4



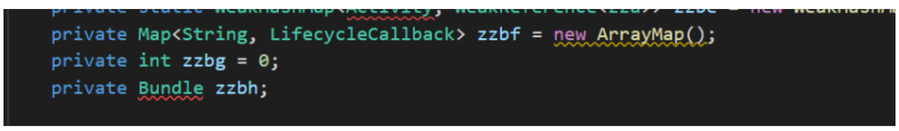
```
public final class zza extends Fragment implements LifecycleFragment {
    private static WeakHashMap<Activity, WeakReference<zza>> zzbe = new WeakHashMap<>();
    private Map<String, LifecycleCallback> zzbf = new ArrayMap();
    private int zzbg = 0;
    private Bundle zzbh;

    public static zza zza(Activity activity) {
        zza zza;
        WeakReference<zza> weakReference = zzbe.get(activity);
        if (weakReference != null && (zza = weakReference.get()) != null) {
            return zza;
        }
```

This method name is the same on enclosing class which it is not-constructor and should be avoided using the same word for two purposes, also the name is meaningless because its letters.



```
public void unregisterOnSharedPreferenceChangeListener(SharedPreferences.OnSharedPreferenceChangeListener onSharedPreferenc
    this.h.lock();
    try {
        this.b.b(new e(this, onSharedPreferenceChangeListener));
    } finally {
        this.h.unlock();
    }
}
```

This name of method is very long. In contrast, the reader is a programmer and understands the technical terms and does not need to clarify the name pulling each name from the scope of the problem is so much it is better to abbreviate it and make it understandable.



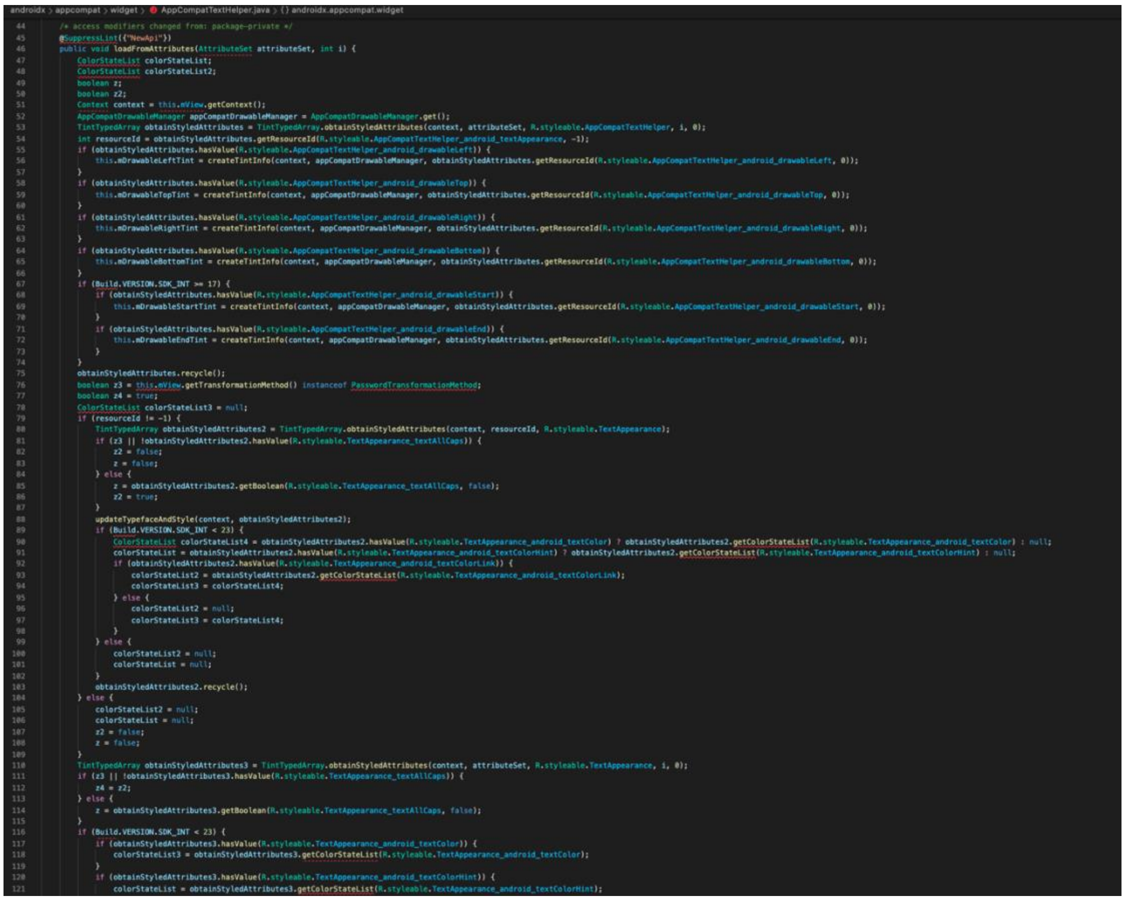
```
private static WeakHashMap<Activity, WeakReference<zza>> zzbe = new WeakHashM
private Map<String, LifecycleCallback> zzbf = new ArrayMap();
private int zzbg = 0;
private Bundle zzbh;
```

These names are similar and differ slightly, and this can cause wasting a lot of time to discover the difference between them in the future, also It is possible that the developer selects an object by name without seeing your comments because there is no clear difference.

Functions

Functions are the first line of organization in any program. Writing them well saves you a lot of trouble.

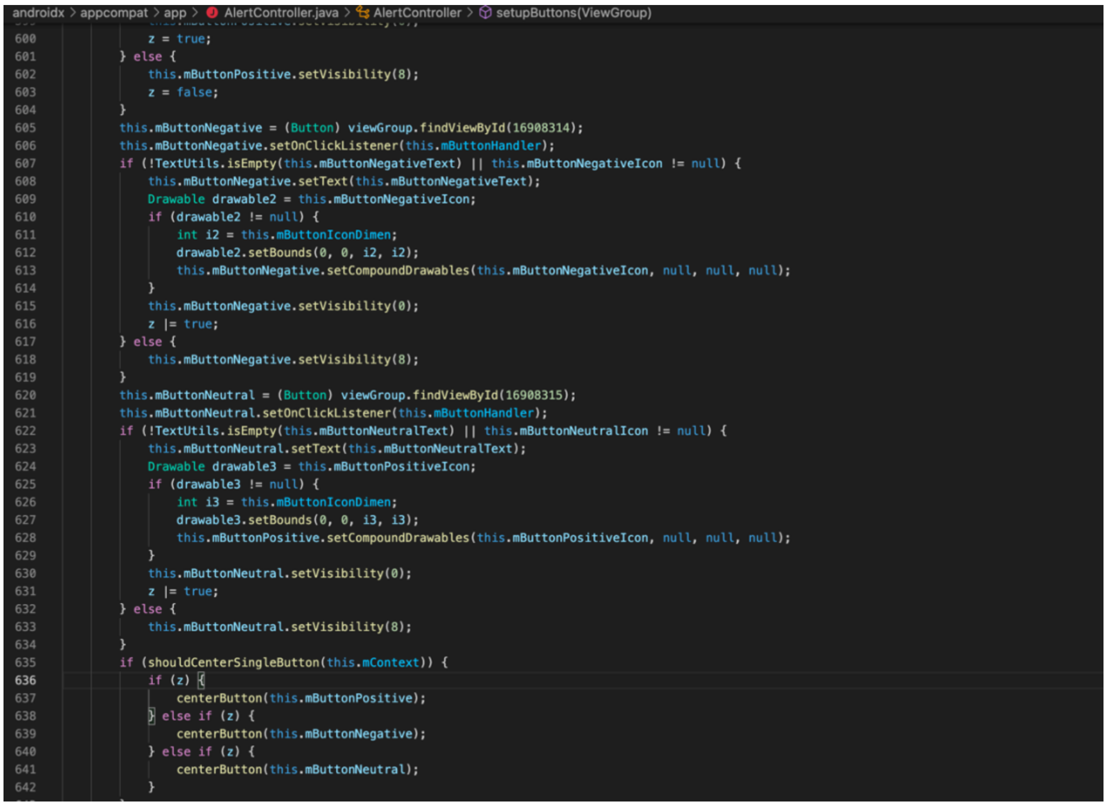
When methods are excessively long this usually indicates that the method is doing more than its name/signature might suggest which also means a violation of the high cohesion design principle. As will as They become challenging for others to digest since excessive scrolling causes readers to lose focus. with just a few simple method extractions, some renaming, and a little restructuring by removing any copy/pasted code lead us to closer to a small and one level of abstraction function.
This is very long method. thus, we could not contain it in one picture, it consists of 130 lines!



```
androidx > appcompat > widget > AppCompatTextHelper.java > {} androidx.appcompat.widget
```

```java
/* access modifiers changed from: package-private */
@SuppressLint({"NewApi"})
public void loadFromAttributes(AttributeSet attributeSet, int i) {
    ColorStateList colorStateList;
    ColorStateList colorStateList2;
    boolean z;
    boolean z2;
    Context context = this.mView.getContext();
    AppCompatDrawableManager appCompatDrawableManager = AppCompatDrawableManager.get();
    TintTypedArray obtainStyledAttributes = TintTypedArray.obtainStyledAttributes(context, attributeSet, R.styleable.AppCompatTextHelper, i, 0);
    int resourceId = obtainStyledAttributes.getResourceId(R.styleable.AppCompatTextHelper_android_textAppearance, -1);
    if (obtainStyledAttributes.hasValue(R.styleable.AppCompatTextHelper_android_drawableLeft)) {
        this.mDrawableLeftTint = createTintInfo(context, appCompatDrawableManager, obtainStyledAttributes.getResourceId(R.styleable.AppCompatTextHelper_android_drawableLeft, 0));
    }
    if (obtainStyledAttributes.hasValue(R.styleable.AppCompatTextHelper_android_drawableTop)) {
        this.mDrawableTopTint = createTintInfo(context, appCompatDrawableManager, obtainStyledAttributes.getResourceId(R.styleable.AppCompatTextHelper_android_drawableTop, 0));
    }
    if (obtainStyledAttributes.hasValue(R.styleable.AppCompatTextHelper_android_drawableRight)) {
        this.mDrawableRightTint = createTintInfo(context, appCompatDrawableManager, obtainStyledAttributes.getResourceId(R.styleable.AppCompatTextHelper_android_drawableRight, 0));
    }
    if (obtainStyledAttributes.hasValue(R.styleable.AppCompatTextHelper_android_drawableBottom)) {
        this.mDrawableBottomTint = createTintInfo(context, appCompatDrawableManager, obtainStyledAttributes.getResourceId(R.styleable.AppCompatTextHelper_android_drawableBottom, 0));
    }
    if (Build.VERSION.SDK_INT >= 17) {
        if (obtainStyledAttributes.hasValue(R.styleable.AppCompatTextHelper_android_drawableStart)) {
            this.mDrawableStartTint = createTintInfo(context, appCompatDrawableManager, obtainStyledAttributes.getResourceId(R.styleable.AppCompatTextHelper_android_drawableStart, 0));
        }
        if (obtainStyledAttributes.hasValue(R.styleable.AppCompatTextHelper_android_drawableEnd)) {
            this.mDrawableEndTint = createTintInfo(context, appCompatDrawableManager, obtainStyledAttributes.getResourceId(R.styleable.AppCompatTextHelper_android_drawableEnd, 0));
        }
    }
    obtainStyledAttributes.recycle();
    boolean z3 = this.mView.getTransformationMethod() instanceof PasswordTransformationMethod;
    boolean z4 = true;
    ColorStateList colorStateList3 = null;
    if (resourceId != -1) {
        TintTypedArray obtainStyledAttributes2 = TintTypedArray.obtainStyledAttributes(context, resourceId, R.styleable.TextAppearance);
        if (z3 || !obtainStyledAttributes2.hasValue(R.styleable.TextAppearance_textAllCaps)) {
            z2 = false;
            z = false;
        } else {
            z = obtainStyledAttributes2.getBoolean(R.styleable.TextAppearance_textAllCaps, false);
            z2 = true;
        }
        updateTypefaceAndStyle(context, obtainStyledAttributes2);
        if (Build.VERSION.SDK_INT < 23) {
            ColorStateList colorStateList4 = obtainStyledAttributes2.hasValue(R.styleable.TextAppearance_android_textColor) ? obtainStyledAttributes2.getColorStateList(R.styleable.TextAppearance_android_textColor) : null;
            colorStateList = obtainStyledAttributes2.hasValue(R.styleable.TextAppearance_android_textColorHint) ? obtainStyledAttributes2.getColorStateList(R.styleable.TextAppearance_android_textColorHint) : null;
            if (obtainStyledAttributes2.hasValue(R.styleable.TextAppearance_android_textColorLink)) {
                colorStateList2 = obtainStyledAttributes2.getColorStateList(R.styleable.TextAppearance_android_textColorLink);
                colorStateList3 = colorStateList4;
            } else {
                colorStateList2 = null;
                colorStateList3 = colorStateList4;
            }
        } else {
            colorStateList2 = null;
            colorStateList = null;
        }
        obtainStyledAttributes2.recycle();
    } else {
        colorStateList2 = null;
        colorStateList = null;
        z2 = false;
        z = false;
    }
    TintTypedArray obtainStyledAttributes3 = TintTypedArray.obtainStyledAttributes(context, attributeSet, R.styleable.TextAppearance, i, 0);
    if (z3 || !obtainStyledAttributes3.hasValue(R.styleable.TextAppearance_textAllCaps)) {
        z4 = z2;
    } else {
        z = obtainStyledAttributes3.getBoolean(R.styleable.TextAppearance_textAllCaps, false);
    }
    if (Build.VERSION.SDK_INT < 23) {
        if (obtainStyledAttributes3.hasValue(R.styleable.TextAppearance_android_textColor)) {
            colorStateList3 = obtainStyledAttributes3.getColorStateList(R.styleable.TextAppearance_android_textColor);
        }
        if (obtainStyledAttributes3.hasValue(R.styleable.TextAppearance_android_textColorHint)) {
            colorStateList = obtainStyledAttributes3.getColorStateList(R.styleable.TextAppearance_android_textColorHint);
```

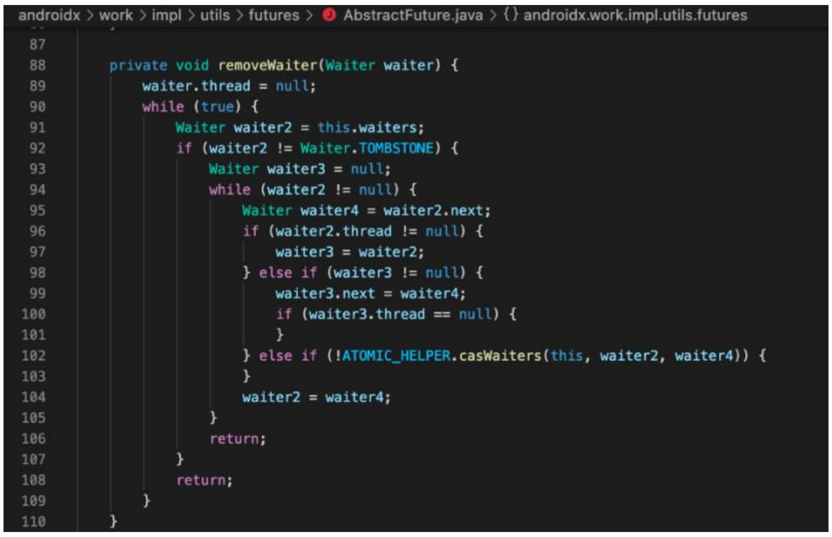
Duplication is a problem because it bloats the code and will require four-fold modification should the algorithm ever have to change. It is also a four-fold opportunity for an error of omission. And as we saw in the code, there were a lot of duplicated blocks and some of them could have been avoided.
Duplicated blocks from line 605-619 and from 620-634.



androidx > appcompat > app > AlertController.java > AlertController > setupButtons(ViewGroup)

```
600             z = true;
601         } else {
602             this.mButtonPositive.setVisibility(8);
603             z = false;
604         }
605         this.mButtonNegative = (Button) viewGroup.findViewById(16908314);
606         this.mButtonNegative.setOnClickListener(this.mButtonHandler);
607         if (!TextUtils.isEmpty(this.mButtonNegativeText) || this.mButtonNegativeIcon != null) {
608             this.mButtonNegative.setText(this.mButtonNegativeText);
609             Drawable drawable2 = this.mButtonNegativeIcon;
610             if (drawable2 != null) {
611                 int i2 = this.mButtonIconDimen;
612                 drawable2.setBounds(0, 0, i2, i2);
613                 this.mButtonNegative.setCompoundDrawables(this.mButtonNegativeIcon, null, null, null);
614             }
615             this.mButtonNegative.setVisibility(0);
616             z |= true;
617         } else {
618             this.mButtonNegative.setVisibility(8);
619         }
620         this.mButtonNeutral = (Button) viewGroup.findViewById(16908315);
621         this.mButtonNeutral.setOnClickListener(this.mButtonHandler);
622         if (!TextUtils.isEmpty(this.mButtonNeutralText) || this.mButtonNeutralIcon != null) {
623             this.mButtonNeutral.setText(this.mButtonNeutralText);
624             Drawable drawable3 = this.mButtonPositiveIcon;
625             if (drawable3 != null) {
626                 int i3 = this.mButtonIconDimen;
627                 drawable3.setBounds(0, 0, i3, i3);
628                 this.mButtonPositive.setCompoundDrawables(this.mButtonPositiveIcon, null, null, null);
629             }
630             this.mButtonNeutral.setVisibility(0);
631             z |= true;
632         } else {
633             this.mButtonNeutral.setVisibility(8);
634         }
635         if (shouldCenterSingleButton(this.mContext)) {
636             if (z) {
637                 centerButton(this.mButtonPositive);
638             } else if (z) {
639                 centerButton(this.mButtonNegative);
640             } else if (z) {
641                 centerButton(this.mButtonNeutral);
642             }
```

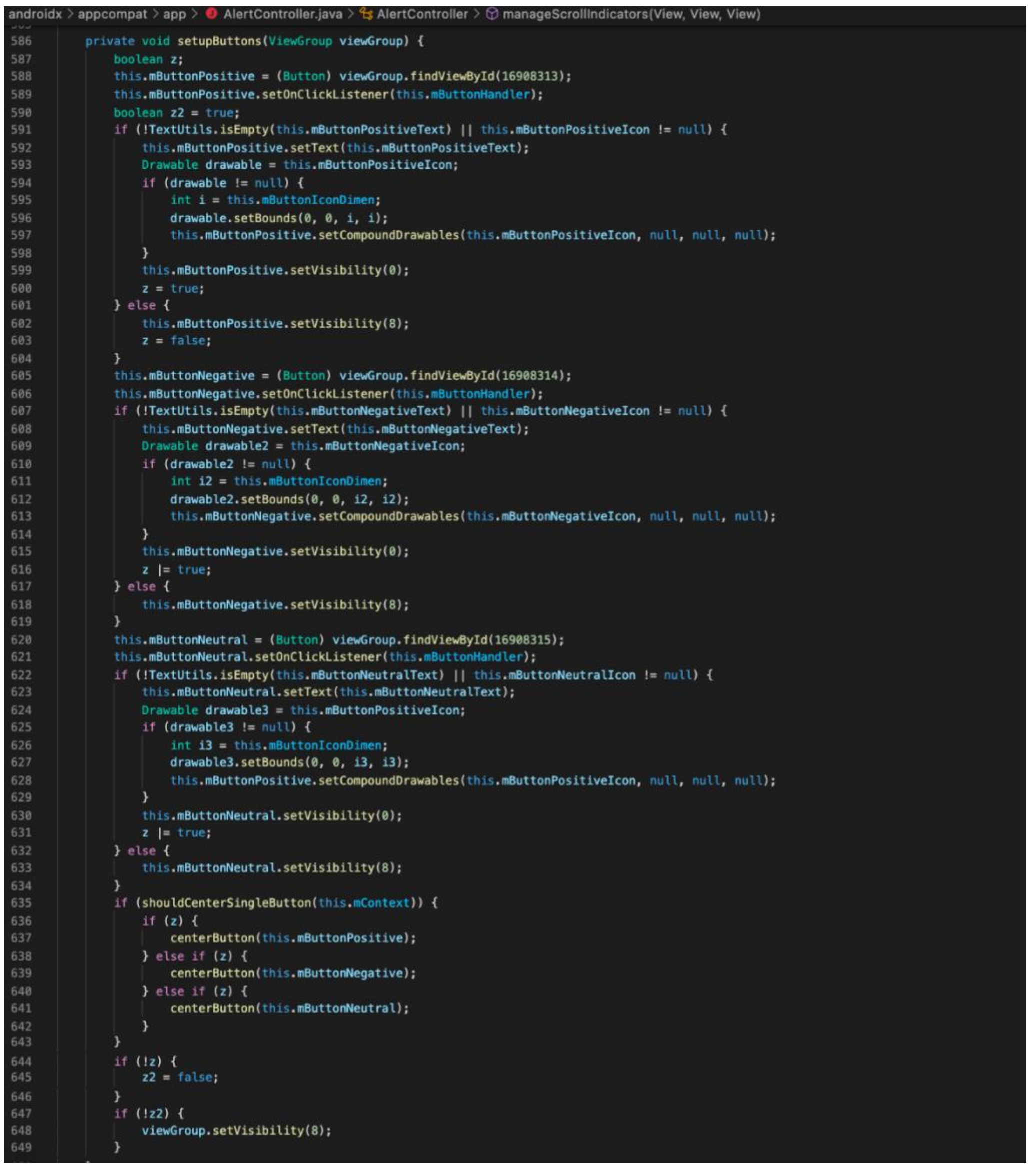
Using a branching statement as the last part of a loop may be a bug, and/or is confusing. Ensure that the usage is not a bug, or consider using another approach. And an Empty if statement blocks were encountered a lot what is the point where a condition is checked but nothing is done about it. This an example combining the two problems, empty if statement block shown at line 100 and 102 and the branching statement is shown in line 108.



androidx > work > impl > utils > futures > AbstractFuture.java > {} androidx.work.impl.utils.futures

```
87
88      private void removeWaiter(Waiter waiter) {
89          waiter.thread = null;
90          while (true) {
91              Waiter waiter2 = this.waiters;
92              if (waiter2 != Waiter.TOMBSTONE) {
93                  Waiter waiter3 = null;
94                  while (waiter2 != null) {
95                      Waiter waiter4 = waiter2.next;
96                      if (waiter2.thread != null) {
97                          waiter3 = waiter2;
98                      } else if (waiter3 != null) {
99                          waiter3.next = waiter4;
100                         if (waiter3.thread == null) {
101                         }
102                     } else if (!ATOMIC_HELPER.casWaiters(this, waiter2, waiter4)) {
103                     }
104                     waiter2 = waiter4;
105                 }
106                 return;
107             }
108             return;
109         }
110     }
```

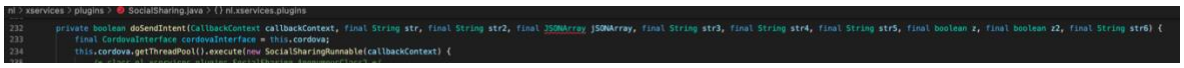
The NPath complexity was encountered a lot in the code, which indicates the number of acyclic execution paths through a method. Meaning that the NPath counts the number of full paths from the beginning to the end of the block of the method. That metric grows exponentially, as it multiplies the complexity of statements in the same block. Below you can see an example method of it.



androidx > appcompat > app > AlertController.java > AlertController > manageScrollIndicators(View, View, View)

```java
    private void setupButtons(ViewGroup viewGroup) {
        boolean z;
        this.mButtonPositive = (Button) viewGroup.findViewById(16908313);
        this.mButtonPositive.setOnClickListener(this.mButtonHandler);
        boolean z2 = true;
        if (!TextUtils.isEmpty(this.mButtonPositiveText) || this.mButtonPositiveIcon != null) {
            this.mButtonPositive.setText(this.mButtonPositiveText);
            Drawable drawable = this.mButtonPositiveIcon;
            if (drawable != null) {
                int i = this.mButtonIconDimen;
                drawable.setBounds(0, 0, i, i);
                this.mButtonPositive.setCompoundDrawables(this.mButtonPositiveIcon, null, null, null);
            }
            this.mButtonPositive.setVisibility(0);
            z = true;
        } else {
            this.mButtonPositive.setVisibility(8);
            z = false;
        }
        this.mButtonNegative = (Button) viewGroup.findViewById(16908314);
        this.mButtonNegative.setOnClickListener(this.mButtonHandler);
        if (!TextUtils.isEmpty(this.mButtonNegativeText) || this.mButtonNegativeIcon != null) {
            this.mButtonNegative.setText(this.mButtonNegativeText);
            Drawable drawable2 = this.mButtonNegativeIcon;
            if (drawable2 != null) {
                int i2 = this.mButtonIconDimen;
                drawable2.setBounds(0, 0, i2, i2);
                this.mButtonNegative.setCompoundDrawables(this.mButtonNegativeIcon, null, null, null);
            }
            this.mButtonNegative.setVisibility(0);
            z |= true;
        } else {
            this.mButtonNegative.setVisibility(8);
        }
        this.mButtonNeutral = (Button) viewGroup.findViewById(16908315);
        this.mButtonNeutral.setOnClickListener(this.mButtonHandler);
        if (!TextUtils.isEmpty(this.mButtonNeutralText) || this.mButtonNeutralIcon != null) {
            this.mButtonNeutral.setText(this.mButtonNeutralText);
            Drawable drawable3 = this.mButtonPositiveIcon;
            if (drawable3 != null) {
                int i3 = this.mButtonIconDimen;
                drawable3.setBounds(0, 0, i3, i3);
                this.mButtonPositive.setCompoundDrawables(this.mButtonPositiveIcon, null, null, null);
            }
            this.mButtonNeutral.setVisibility(0);
            z |= true;
        } else {
            this.mButtonNeutral.setVisibility(8);
        }
        if (shouldCenterSingleButton(this.mContext)) {
            if (z) {
                centerButton(this.mButtonPositive);
            } else if (z) {
                centerButton(this.mButtonNegative);
            } else if (z) {
                centerButton(this.mButtonNeutral);
            }
        }
        if (!z) {
            z2 = false;
        }
        if (!z2) {
            viewGroup.setVisibility(8);
        }
```

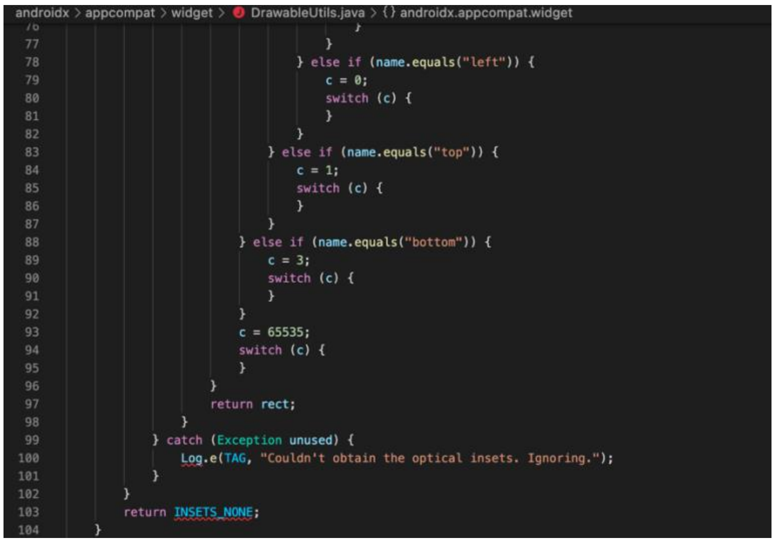
Methods with numerous parameters are a challenge to maintain, especially if most of them share the same datatype, and they are even harder from a testing point of view. Imagine the difficulty of writing all the test cases to ensure that all the various combinations of arguments work properly. If there are no arguments, this is trivial. If there's one argument, it's not too hard. With two arguments the problem gets a bit more challenging. With more than two arguments, testing every combination of appropriate values can be daunting.



nl > xservices > plugins > SocialSharing.java > {} nl.xservices.plugins

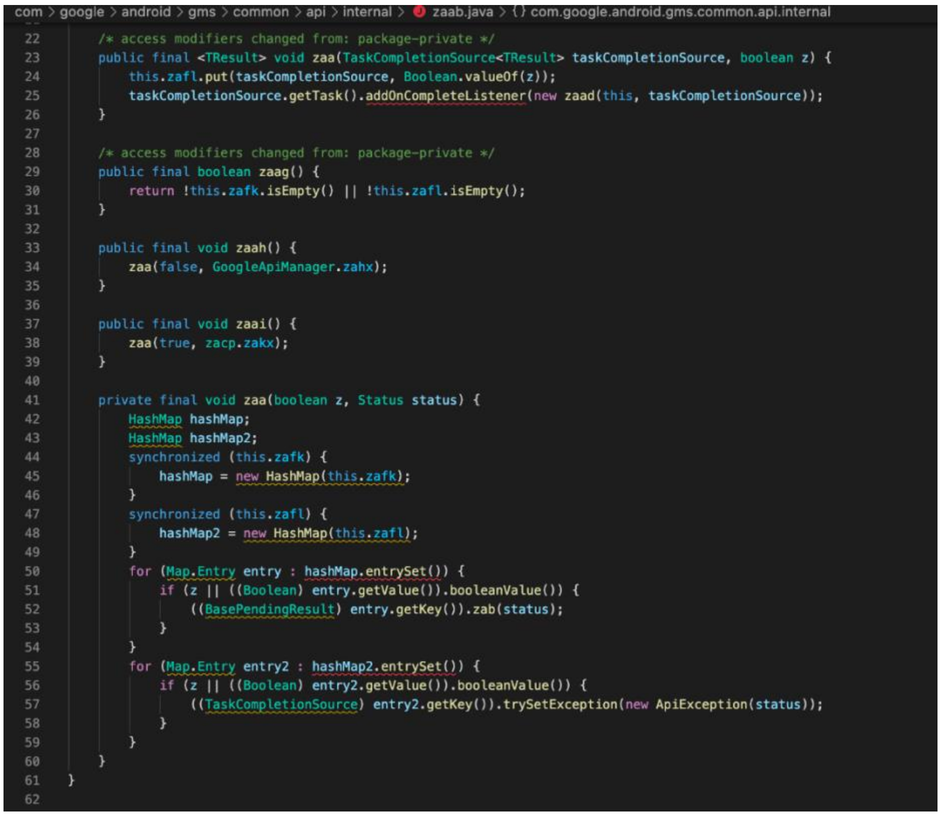
```java
    private boolean doSendIntent(CallbackContext callbackContext, final String str, final String str2, final JSONArray jSONArray, final String str3, final String str4, final String str5, final boolean z, final boolean z2, final String str6) {
        final CordovaInterface cordovaInterface = this.cordova;
        this.cordova.getThreadPool().execute(new SocialSharingRunnable(callbackContext) {
```

An Empty switch statements serve no purpose and should be removed. See below for the example.


androidx > appcompat > widget > DrawableUtils.java > {} androidx.appcompat.widget

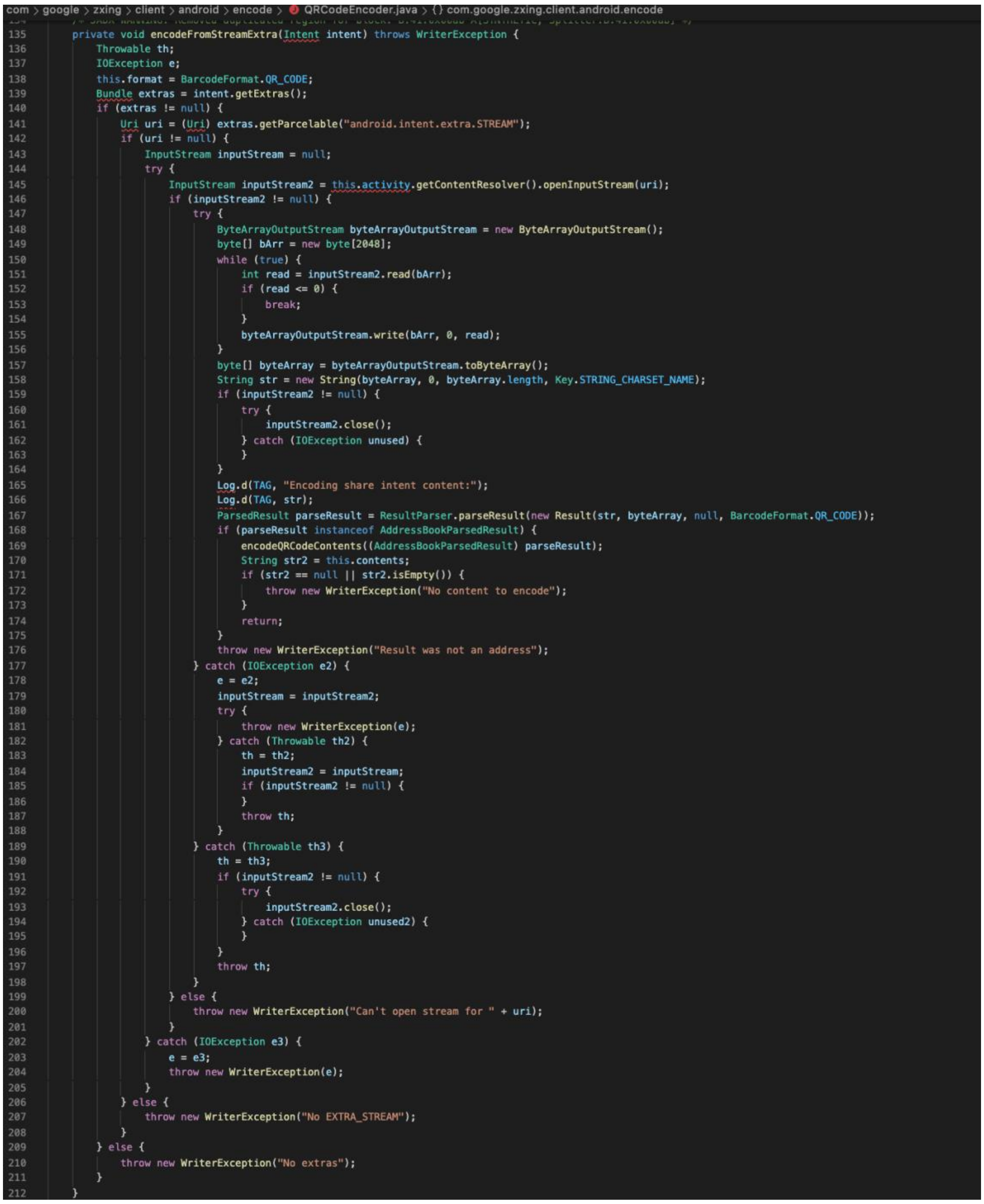
```java
                                        }
                                    }
                                } else if (name.equals("left")) {
                                    c = 0;
                                    switch (c) {
                                    }
                                }
                            } else if (name.equals("top")) {
                                c = 1;
                                switch (c) {
                                }
                            }
                        } else if (name.equals("bottom")) {
                            c = 3;
                            switch (c) {
                            }
                        }
                        c = 65535;
                        switch (c) {
                        }
                    }
                    return rect;
                }
            } catch (Exception unused) {
                Log.e(TAG, "Couldn't obtain the optical insets. Ignoring.");
            }
        }
        return INSETS_NONE;
    }
```

Methods should have a descriptive unambiguous name that describes what the function does. A long descriptive name is better than a short enigmatic name. A long descriptive name is better than a long descriptive comment. The picture below contains several methods with inscrutable names.


com > google > android > gms > common > api > internal > zaab.java > {} com.google.android.gms.common.api.internal

```java
    /* access modifiers changed from: package-private */
    public final <TResult> void zaa(TaskCompletionSource<TResult> taskCompletionSource, boolean z) {
        this.zafl.put(taskCompletionSource, Boolean.valueOf(z));
        taskCompletionSource.getTask().addOnCompleteListener(new zaad(this, taskCompletionSource));
    }

    /* access modifiers changed from: package-private */
    public final boolean zaag() {
        return !this.zafk.isEmpty() || !this.zafl.isEmpty();
    }

    public final void zaah() {
        zaa(false, GoogleApiManager.zahx);
    }

    public final void zaai() {
        zaa(true, zacp.zakx);
    }

    private final void zaa(boolean z, Status status) {
        HashMap hashMap;
        HashMap hashMap2;
        synchronized (this.zafk) {
            hashMap = new HashMap(this.zafk);
        }
        synchronized (this.zafl) {
            hashMap2 = new HashMap(this.zafl);
        }
        for (Map.Entry entry : hashMap.entrySet()) {
            if (z || ((Boolean) entry.getValue()).booleanValue()) {
                ((BasePendingResult) entry.getKey()).zab(status);
            }
        }
        for (Map.Entry entry2 : hashMap2.entrySet()) {
            if (z || ((Boolean) entry2.getValue()).booleanValue()) {
                ((TaskCompletionSource) entry2.getKey()).trySetException(new ApiException(status));
            }
        }
    }
}
```

As we mentioned earlier Functions should do one thing and one thig only. Error handling is one thing. Thus, a function that handles errors should do nothing else. This implies that if the keyword `try` exists in a function, it should be the very first word in the function and that there should be nothing after the `catch/finally` blocks. But in the example here we see that the function is doing other thing beside Error handling.


com › google › zxing › client › android › encode › QRCodeEncoder.java › {} com.google.zxing.client.android.encode

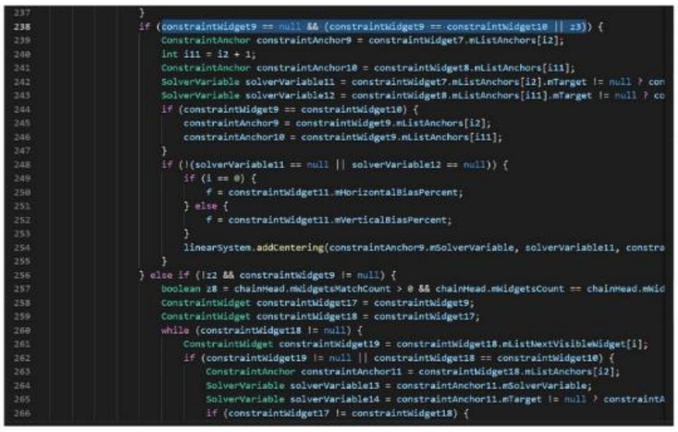
```java
private void encodeFromStreamExtra(Intent intent) throws WriterException {
    Throwable th;
    IOException e;
    this.format = BarcodeFormat.QR_CODE;
    Bundle extras = intent.getExtras();
    if (extras != null) {
        Uri uri = (Uri) extras.getParcelable("android.intent.extra.STREAM");
        if (uri != null) {
            InputStream inputStream = null;
            try {
                InputStream inputStream2 = this.activity.getContentResolver().openInputStream(uri);
                if (inputStream2 != null) {
                    try {
                        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
                        byte[] bArr = new byte[2048];
                        while (true) {
                            int read = inputStream2.read(bArr);
                            if (read <= 0) {
                                break;
                            }
                            byteArrayOutputStream.write(bArr, 0, read);
                        }
                        byte[] byteArray = byteArrayOutputStream.toByteArray();
                        String str = new String(byteArray, 0, byteArray.length, Key.STRING_CHARSET_NAME);
                        if (inputStream2 != null) {
                            try {
                                inputStream2.close();
                            } catch (IOException unused) {
                            }
                        }
                        Log.d(TAG, "Encoding share intent content:");
                        Log.d(TAG, str);
                        ParsedResult parseResult = ResultParser.parseResult(new Result(str, byteArray, null, BarcodeFormat.QR_CODE));
                        if (parseResult instanceof AddressBookParsedResult) {
                            encodeQRCodeContents((AddressBookParsedResult) parseResult);
                            String str2 = this.contents;
                            if (str2 == null || str2.isEmpty()) {
                                throw new WriterException("No content to encode");
                            }
                            return;
                        }
                        throw new WriterException("Result was not an address");
                    } catch (IOException e2) {
                        e = e2;
                        inputStream = inputStream2;
                        try {
                            throw new WriterException(e);
                        } catch (Throwable th2) {
                            th = th2;
                            inputStream2 = inputStream;
                            if (inputStream2 != null) {
                            }
                            throw th;
                        }
                    } catch (Throwable th3) {
                        th = th3;
                        if (inputStream2 != null) {
                            try {
                                inputStream2.close();
                            } catch (IOException unused2) {
                            }
                        }
                        throw th;
                    }
                } else {
                    throw new WriterException("Can't open stream for " + uri);
                }
            } catch (IOException e3) {
                e = e3;
                throw new WriterException(e);
            }
        } else {
            throw new WriterException("No EXTRA_STREAM");
        }
    } else {
        throw new WriterException("No extras");
    }
}
```

After seeking looking for issues and rules violations, we should highlight the programmers' good practices that been followed when writing the code. Per to our observations, the whole code's structure satisfied the low coupling design principle.
Always keep in mind when writing functions, that they will come out long and complicated. They will have lots of indenting and nested loops. They will have long argument lists. The names will be arbitrary, and there will be a duplicated code. But we also should have a suite of unit tests that cover every one of those clumsy lines of code.
So then we should refactored refine that code, splitting out functions, changing names, eliminating duplication. And sometimes shrinking the methods and reordering them. Even Sometimes break out whole classes, all the while keeping the tests passing.
In the end, we will wind up with functions that follows the rules we discussed. And there is more of it, but always keep in mind we do not write them that way to start. we do not think anyone could.

Error Handling

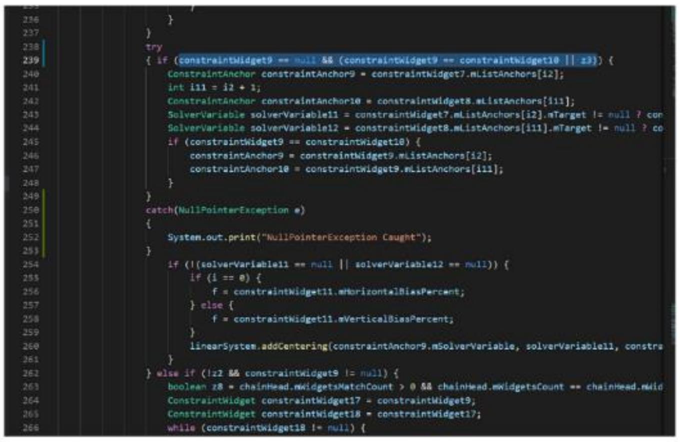
Error handling is just one of those things that we all must do when we program. Devices can fail and input can be unexpected. In short, things can go wrong, and it is our responsibility as programmers to ensure that our code does what it is supposed to do when they do. Error handling is necessary, but it is incorrect if it obscures reasoning.

In some techniques for handling and reporting errors were limited. You either set an error flag or returned an error code that the caller could check. The caller must check for errors immediately after the call, this is easy to forget. For this reason, it is better to throw an exception when you encounter an error. The calling code is cleaner.



```java
}
if (constraintWidget9 == null && (constraintWidget9 == constraintWidget10 || z3)) {
    ConstraintAnchor constraintAnchor9 = constraintWidget7.mListAnchors[i2];
    int i11 = i2 + 1;
    ConstraintAnchor constraintAnchor10 = constraintWidget8.mListAnchors[i11];
    SolverVariable solverVariable11 = constraintWidget7.mListAnchors[i2].mTarget != null ? con
    SolverVariable solverVariable12 = constraintWidget8.mListAnchors[i11].mTarget != null ? co
    if (constraintWidget9 == constraintWidget10) {
        constraintAnchor9 = constraintWidget9.mListAnchors[i2];
        constraintAnchor10 = constraintWidget9.mListAnchors[i11];
    }
    if (!(solverVariable11 == null || solverVariable12 == null)) {
        if (i == 0) {
            f = constraintWidget11.mHorizontalBiasPercent;
        } else {
            f = constraintWidget11.mVerticalBiasPercent;
        }
        linearSystem.addCentering(constraintAnchor9.mSolverVariable, solverVariable11, constra
    }
} else if (!z2 && constraintWidget9 != null) {
    boolean z8 = chainHead.mWidgetsMatchCount > 0 && chainHead.mWidgetsCount == chainHead.mWid
    ConstraintWidget constraintWidget17 = constraintWidget9;
    ConstraintWidget constraintWidget18 = constraintWidget17;
    while (constraintWidget18 != null) {
        ConstraintWidget constraintWidget19 = constraintWidget18.mListNextVisibleWidget[i];
        if (constraintWidget19 != null || constraintWidget18 == constraintWidget10) {
            ConstraintAnchor constraintAnchor11 = constraintWidget18.mListAnchors[i2];
            SolverVariable solverVariable13 = constraintAnchor11.mSolverVariable;
            SolverVariable solverVariable14 = constraintAnchor11.mTarget != null ? constraintA
            if (constraintWidget17 != constraintWidget18) {
```

Without try catch my appear NullPointerExceptio:

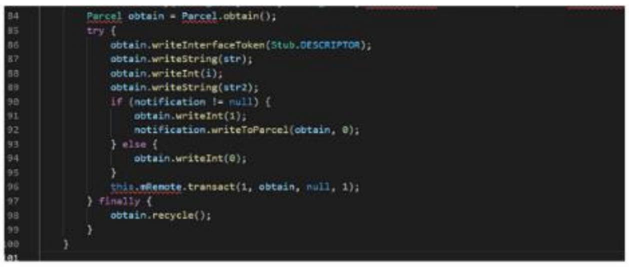


```java
        }
    }
    try
    { if (constraintWidget9 == null && (constraintWidget9 == constraintWidget10 || z3)) {
        ConstraintAnchor constraintAnchor9 = constraintWidget7.mListAnchors[i2];
        int i11 = i2 + 1;
        ConstraintAnchor constraintAnchor10 = constraintWidget8.mListAnchors[i11];
        SolverVariable solverVariable11 = constraintWidget7.mListAnchors[i2].mTarget != null ? con
        SolverVariable solverVariable12 = constraintWidget8.mListAnchors[i11].mTarget != null ? co
        if (constraintWidget9 == constraintWidget10) {
            constraintAnchor9 = constraintWidget9.mListAnchors[i2];
            constraintAnchor10 = constraintWidget9.mListAnchors[i11];
        }
    }
    catch(NullPointerException e)
    {
        System.out.print("NullPointerException Caught");
    }
        if (!(solverVariable11 == null || solverVariable12 == null)) {
            if (i == 0) {
                f = constraintWidget11.mHorizontalBiasPercent;
            } else {
                f = constraintWidget11.mVerticalBiasPercent;
            }
            linearSystem.addCentering(constraintAnchor9.mSolverVariable, solverVariable11, constra
        }
    } else if (!z2 && constraintWidget9 != null) {
        boolean z8 = chainHead.mWidgetsMatchCount > 0 && chainHead.mWidgetsCount == chainHead.mWid
        ConstraintWidget constraintWidget17 = constraintWidget9;
        ConstraintWidget constraintWidget18 = constraintWidget17;
        while (constraintWidget18 != null) {
```

With try catch handling:

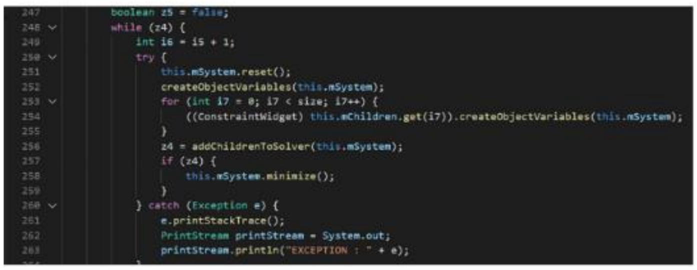
Notice how much cleaner it is. This isn’t just a matter of aesthetics. The code is better because two concerns that were tangled, the algorithm for device shutdown and error handling, are now separated.

To satisfy your tests, try writing tests that force exceptions and then adding behavior to your handler. When you run code in the try component of a try-catch-finally statement, you are stating that execution can stop at any time and continue at the catch. Whatever happens in the try, your catch must leave your program in a consistent condition, perform finally block that have important code in our program that needs to be executed irrespective of whether the exception is thrown. As a result, when writing code that may throw exceptions, it is a good way to start with a try-catch-finally statement.
Examples of try-catch-finally exception handling:



```
        Parcel obtain = Parcel.obtain();
        try {
            obtain.writeInterfaceToken(Stub.DESCRIPTOR);
            obtain.writeString(str);
            obtain.writeInt(i);
            obtain.writeString(str2);
            if (notification != null) {
                obtain.writeInt(1);
                notification.writeToParcel(obtain, 0);
            } else {
                obtain.writeInt(0);
            }
            this.mRemote.transact(1, obtain, null, 1);
        } finally {
            obtain.recycle();
        }
    }
```

try-finally

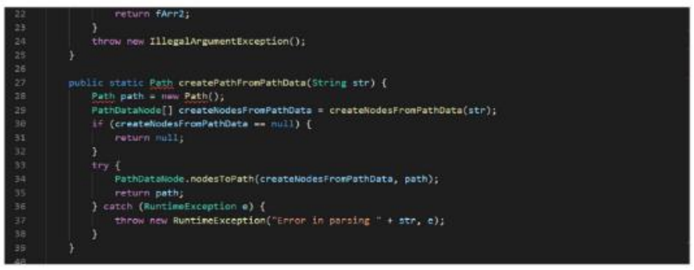


```
        boolean z5 = false;
        while (z4) {
            int i6 = i5 + 1;
            try {
                this.mSystem.reset();
                createObjectVariables(this.mSystem);
                for (int i7 = 0; i7 < size; i7++) {
                    ((ConstraintWidget) this.mChildren.get(i7)).createObjectVariables(this.mSystem);
                }
                z4 = addChildrenToSolver(this.mSystem);
                if (z4) {
                    this.mSystem.minimize();
                }
            } catch (Exception e) {
                e.printStackTrace();
                PrintStream printStream = System.out;
                printStream.println("EXCEPTION : " + e);
```

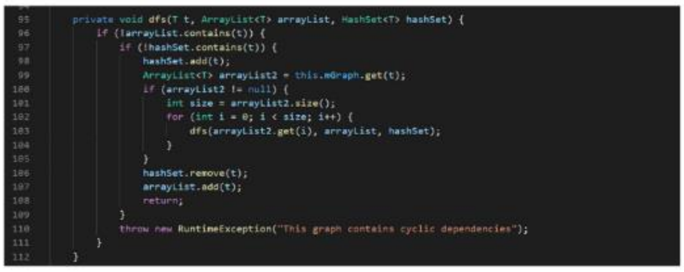
try-catch

Throwing a specific error/exception rather than a generic/general exception is a recommended practice in Java. However, to make your code cleaner Avoid throwing certain exception types. Rather than throw a raw RuntimeException, Throwable, Exception, or Error, use a subclassed exception or error instead.

These some examples in Alinma application source code:

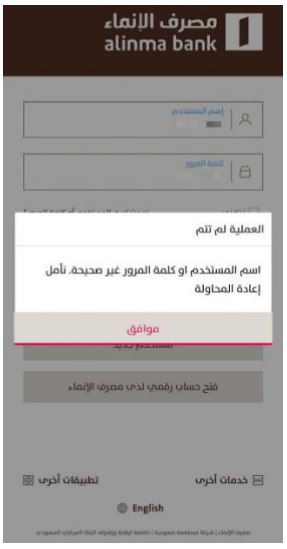


```
            return fArr2;
        }
        throw new IllegalArgumentException();
    }

    public static Path createPathFromPathData(String str) {
        Path path = new Path();
        PathDataNode[] createNodesFromPathData = createNodesFromPathData(str);
        if (createNodesFromPathData == null) {
            return null;
        }
        try {
            PathDataNode.nodesToPath(createNodesFromPathData, path);
            return path;
        } catch (RuntimeException e) {
            throw new RuntimeException("Error in parsing " + str, e);
        }
    }
```



```
    private void dfs(T t, ArrayList<T> arrayList, HashSet<T> hashSet) {
        if (!arrayList.contains(t)) {
            if (!hashSet.contains(t)) {
                hashSet.add(t);
                ArrayList<T> arrayList2 = this.mGraph.get(t);
                if (arrayList2 != null) {
                    int size = arrayList2.size();
                    for (int i = 0; i < size; i++) {
                        dfs(arrayList2.get(i), arrayList, hashSet);
                    }
                }
                hashSet.remove(t);
                arrayList.add(t);
                return;
            }
            throw new RuntimeException("This graph contains cyclic dependencies");
        }
    }
```

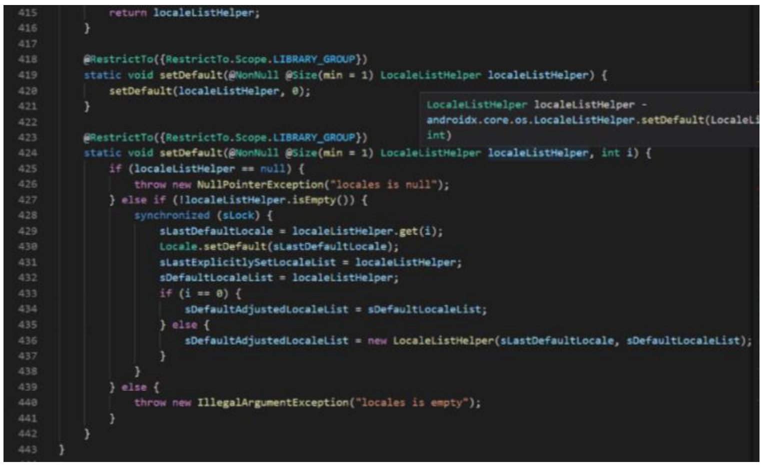
You should provide enough context for each exception you throw to determine the source and location of an error. Create informative error messages and pass them along with your exceptions. Mention the operation that failed and the type of failure. If you are logging in your application, pass along enough information to be able to log the error in your catch.



العملية لم تتم

اسم المستخدم او كلمة المرور غير صحيحة. نأمل إعادة المحاولة

Avoid throwing NullPointerExceptions manually. To avoid a method being called with a null parameter, you may consider using an IllegalArgumentException instead, making it clearly seen as a programmer-initiated exception. However, there are better ways to handle this:
other exceptions are standardly used for certain kinds of illegal arguments and states. If a caller passes null in some parameter for which null values are prohibited, convention dictates that NullPointerException be thrown rather than IllegalArgumentException.
To implement that, you are encouraged to use java.util.Objects.requireNonNull() This method is designed primarily for doing parameter validation in methods and constructors with multiple parameters.



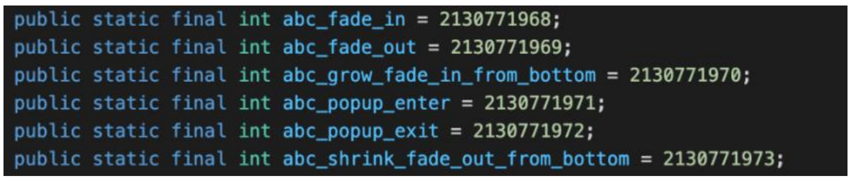
```
        return localeListHelper;
    }

    @RestrictTo({RestrictTo.Scope.LIBRARY_GROUP})
    static void setDefault(@NonNull @Size(min = 1) LocaleListHelper localeListHelper) {
        setDefault(localeListHelper, 0);
    }                                              LocaleListHelper localeListHelper -
                                                   androidx.core.os.LocaleListHelper.setDefault(LocaleLi
    @RestrictTo({RestrictTo.Scope.LIBRARY_GROUP})  int)
    static void setDefault(@NonNull @Size(min = 1) LocaleListHelper localeListHelper, int i) {
        if (localeListHelper == null) {
            throw new NullPointerException("locales is null");
        } else if (!localeListHelper.isEmpty()) {
            synchronized (sLock) {
                sLastDefaultLocale = localeListHelper.get(i);
                Locale.setDefault(sLastDefaultLocale);
                sLastExplicitlySetLocaleList = localeListHelper;
                sDefaultLocaleList = localeListHelper;
                if (i == 0) {
                    sDefaultAdjustedLocaleList = sDefaultLocaleList;
                } else {
                    sDefaultAdjustedLocaleList = new LocaleListHelper(sLastDefaultLocale, sDefaultLocaleList);
                }
            }
        } else {
            throw new IllegalArgumentException("locales is empty");
        }
    }
}
```

Formatting

Following a specific formatting method can really help the programmers when reading the code after sharing it with others, the code is then easier on the eyes.

Using proper indentation also falls under formatting but indentation should be taken into consideration what programming language we used. Indentation could also show were the code falls in the hierarchy of the code.

In this section of code, vertical alignment could've been followed where all the values would be in one vertical column.

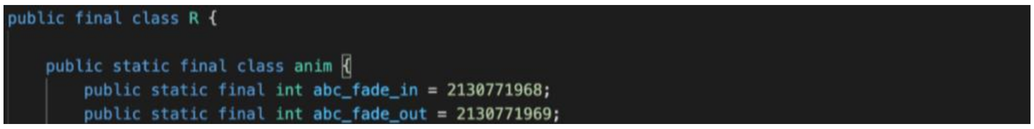


```
public static final int abc_fade_in = 2130771968;
public static final int abc_fade_out = 2130771969;
public static final int abc_grow_fade_in_from_bottom = 2130771970;
public static final int abc_popup_enter = 2130771971;
public static final int abc_popup_exit = 2130771972;
public static final int abc_shrink_fade_out_from_bottom = 2130771973;
```

Code Documentation Analysis

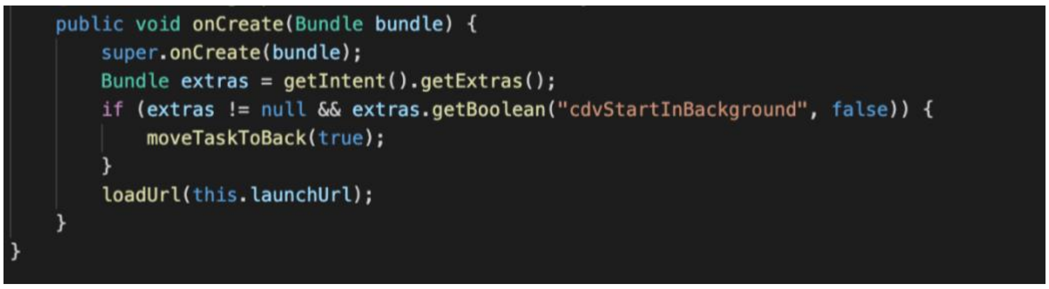
The purpose of code documentation is to compensate for the times we need to express ourselves in the code. Good comments are beneficiary in times where we do not understand the code written which would save us time. It is good to include informative comments, such as the parameter types and what is the return value type for easy understanding. Developers should have included these so that other team members have an easier time understating.

Below are some screenshots of some segments of the java code for the Alinma app. They show the lack of documentation that the app developers should have included. The lack of documentation could be a security precaution.



```
public final class R {

    public static final class anim {
        public static final int abc_fade_in = 2130771968;
        public static final int abc_fade_out = 2130771969;
```

They should've explained the purpose of this class, and the purpose of the assigned variables as comments.



```
    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        Bundle extras = getIntent().getExtras();
        if (extras != null && extras.getBoolean("cdvStartInBackground", false)) {
            moveTaskToBack(true);
        }
        loadUrl(this.launchUrl);
    }
}
```

The purpose of the method was not explained as a comment.
Overall, the java codes showed very poor documentation, there were rarely any comments explaining the purposes of the classes nor the methods. It did not follow the Javadoc style guide as there were not headers before the classes nor sufficient comments in the classes.

Conclusion

To conclude this analysis report of our chosen banking app, we learned that a lot of factors are included in the inspection and the evaluation of the app from the security analysis all the way to the documentation analysis of the code. Using the MobSF tool helped sum the different segments in the app, we were able to look at each segment individually to know which flaws were detected. The purpose of this is to show that even big branded and known apps and systems will have flaws, and we need to analyze in order to reduce the flaws and bring up the overall score of the system or app. Maintenance needs to be a big part of the app throughout the years to reduce these minor problems. This workshop assessed the application only, must work on the maintenance of the issues that have been discussed in our report to improve the overall quality and avoid failure to the application.

References

1. Arabian Business
   www.arabianbusiness.com

2. How to install and Use MobSF on macOS | Android Penetration Testing Suite
   www.youtube.com

3. How to install Docker on Mac OS in 3 minutes
   www.youtube.com

4. How to use MobSF
   www.youtube.com

5. Clean Code by Robert C. Martin