Al-Imam Mohammad Bin Saud Islamic University
College of Computer and Information Sciences,
Department of Computer Science
CS 392 Software Engineering 2
November 4, 2021

Workshop #1: Quality Assessment for Local Banking Apps

Al Rajhi Bank

Student's Name	Student's E-mail	Student's ID
Areej Turky Alotaibi	atsalotaibi03@sm.imamu.edu.sa	440023303
Raghad Adel Alshabana	raaalshabana@sm.imamu.edu.sa	440021235
Slima Mohammed bnous	smbjlal@sm.imamu.edu.sa	439049380
Shoroog Saad Alarifi	sssalarifi@sm.imamu.edu.sa	440022128

Supervisor: Lamees Alhazzaa

Table of Contents

1. Introduction

Financial institutions are rapidly adopting digital technology. The banking business is increasingly dependent on technology and many types of banking applications, which must run smoothly to give a positive customer experience.

Banking systems must have the right quality assurance strategy in place as they handle sensitive customer data and large volumes of transactions 24 hours a day, seven days a week. Before putting these banking products on the market, therefore, it is important to evaluate the quality for us to develop and verify the entire quality of the application.

As Android system, there're more than 572,838 users use "Alrajhi-Bank", users put all their trust in banking systems since it has access and control to what they deposited. Any small mistake will lead to high risk and life-threatening.
Al-Rajhi Bank system as a Saudi bank connects users' accounts with the "Absher" system. Potential responsibility to the bank since it connects to critical information system.

"Using a Coding Standard is one of the best ways to ensure high-quality code".

The "Code Quality" as a term is all subjective, depending on the teams, organizations. Etc. purpose. But in the end, they must have equivalent results, and that is what's called standard...

1.1 The purpose

-Improving the code

-Improving the programmers

1.2 The goals

The main objective of this report is to provide results and analyses to assess the quality of mobile banking applications running on the Android operating system, and specifically, our focus will be on the Al Rajhi Bank application. We are interested in this report on the importance of evaluation criteria adopted in studies in the quality of mobile applications for individual users...

We aim to provide standard and clean code quality attributes recommendations to AlRajhi developers to maintain the quality of their code. For sure as a team, we aim to improve our programming knowledge and this comes from observing, teaching, and learning more professional codes.

The structure of the report consists in presenting how we extracted the data that we will analyze, a description of the results obtained with recommendations, and our view of the entire quality of the application.

2. Methodology

2.1 Methodology of Extracting the Source Code:

The decision of choosing Al-Rajhi Bank was not Random. After downloading all four apps, we use “APK Extractor” as an Android application then use “Online APK Decompiler”; The Source Code and all other components are clearly defined to us. One of our priorities as a team in choosing an app is that the app must be written in Java language so that Alinma bank was ignored (written in Kotlin). After observing classes and methods, the choice was made up to Al-Rajhi Bank, since some issues were clear to us early.

2.2 Methodology of Extracting the MobSF:

We used a Mobile Security Framework tool for several reasons, including that it is all-in-one and supports uploading APK files, and provides an easy interface for the results, so it was a good choice to evaluate the security of the application after the tool was installed, the APK file of the Al Rajhi application was uploaded then the interface appeared, We focused on the highest-risk security findings also extract some scores related with the overall result, and analyzes discussed in below.

3. Discussions

3.1 MobSF Analysis Result :

In AlRajhi bank some permissions may be dangerous for Malicious applications

- Access coarse location sources, such as the mobile network database, to determine an approximate phone location, where available, Malicious applications can use this to determine approximately where you are and may consume additional battery power.
- Allows an application to take pictures and videos with the camera, this allows the application to collect images that the camera is seeing at any time.
- Allows an application to read/write from/to external storage, Any App can read data written to External Storage, so it is insecure Data Storage.

Here are some lines from the code that writes to external storage:

```
            if (readableMap.hasKey(a6)) {
                str = readableMap.getString(a6);
                if (!isFileNameValid(str)) {
                    promise.reject(new Exception(C01921.a(33621)));
                    return;
                }
            } else {
                str = C01921.a(33622) + UUID.randomUUID().toString();
            }
            if (readableMap.hasKey(a8)) {
                if (C01921.a(33623).equals(Environment.getExternalStorageState())) {
                    file2 = new File(Environment.getExternalStorageDirectory(), readableMap.getString(a8));
                } else {
                    file2 = new File(this.mReactContext.getFilesDir(), readableMap.getString(a8));
                }
                if (file2.exists() || file2.mkdirs()) {
                    file = new File(file2, str + C01921.a(33625));
                } else {
                    promise.reject(new Exception(C01921.a(33624)));
                    return;
                }
            } else {
                file = getTempFile(str);
```

Figure 1

```
    @ReactMethod(isBlockingSynchronousMethod = true)
    public double getFreeDiskStorageOldSync() {
        long j;
        long j2;
        try {
            StatFs statFs = new StatFs(Environment.getExternalStorageDirectory().getAbsolutePath());
            if (Build.VERSION.SDK_INT < 18) {
                j2 = (long) statFs.getAvailableBlocks();
                j = (long) statFs.getBlockSize();
            } else {
                j2 = statFs.getAvailableBlocksLong();
                j = statFs.getBlockSizeLong();
            }
            return BigInteger.valueOf(j2).multiply(BigInteger.valueOf(j)).doubleValue();
        } catch (Exception unused) {
            return -1.0d;
        }
    }
```

Figure 2

- Allows an application to read all the contact (address) data stored on your phone. Malicious applications can use this to send your data to other people.

- Allows an application to read the user's profile data and to access the audio record path, so malicious applications can access this information
- Allows an application to request installing packages, Malicious applications can use this to try and trick users into installing additional malicious packages.
- Allows an application to show system-alert windows. Malicious applications can take over the entire screen of the phone.

Of course, they put these permissions for the benefit of the user, but I think most of the risk is higher than the benefit gained.
They were supposed to study these permissions before placing them to users and take into account the status of these permissions for a critical sensitive application,
so is the benefit of these permissions worth the risk?
From our point of view, we think that there is an alternative solution, we know often we do not benefit from these all the time except in rare cases so only then do we request the required data from a user.

There are also other issues not related to permissions:

- Activities in operating systems must define how they are initiated and interconnected by “Launch Mode of Activity” we found that the LMOA using (single-task/single-instance) not the standard, and it is possible for other applications to read the contents of the calling Intent.

  So, it is required to use the "standard" launch mode attribute when sensitive information is included in the Intent.

- Some services and broadcast Receivers are 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 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 permission and interact with the component.
But if it is set to a **signature**, only applications signed with the same certificate can obtain permission.

We recommend making sure that the permission is set signature**.**

- Clear text traffic is Enabled for App, the app intends to use cleartext network traffic, 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.
  Unfortunately, it is by default will be true that's mean it is activated, but there are several solutions to it by adding inside the code ether just add the cleartextTrafficPermitted to true or adding Network Security Config xml file in manifest

- Insertion of Sensitive Information into Log File, Sensitive information should never be logged.

3.1.1 discuss the results and how to solve these issues

Our first concern was the strength of protection in the application, Al-Rajhi app Security percentage that came out of a MobSF tool is :

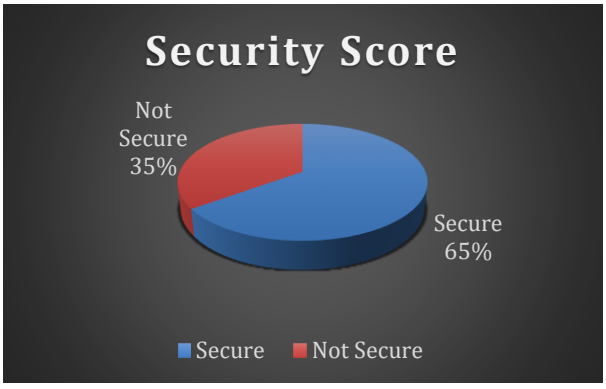

Figure 3

To measure system vulnerabilities and assess the severity of security vulnerabilities we extracted the average score of Common Vulnerability Scoring System **CVSS** :

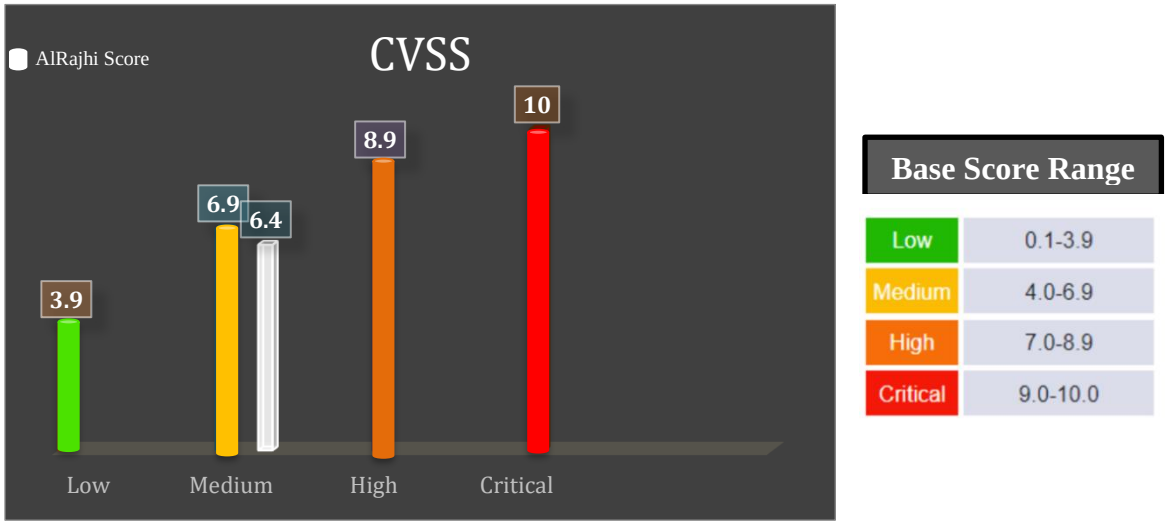

Figure 4

The extent of software vulnerabilities and the severity of system vulnerabilities is rather high, with the aforementioned score of 6.4 considered close to the onset of high risk, which starts with 7

3.2 PMD Analysis Result

The results of the tests and controls evaluated in this code review provided a number of relevant findings regarding the application reviewed by these tools: PMD and check style. A general overview is depicted in tables, which show the findings and their impact on the categories included in the analysis.

AvoidFileStream :

The unresolvable compatibility issue is the requirement
in FileInputStream and FileOutputStream finalizer methods to call close.
The proposed reference-based cleaners can not implement anything calling close() because it would require holding a reference to the FIS/FOS which would prevent the stream from becoming phantom referenced.
Since it is unknown/unknowable how many FIS/FOS subclasses might rely on overriding close or finalize the compatibility issue is severe and Some platforms, in particular, allow a file to be opened for writing by only one FileOutputStream at a time. In such situations, the constructors in this class will fail if the file involved is already open.
AvoidFileStream is a HIGH-risk problem in the category of Performance because it is can have an impact on a technical and business So They must be resolved because losing or not fully accessing files can cause successive losses and problems. The solutions proposed by the PMD tool are processes that help prevent Garbage Collection pauses, no finalization.

ClassNamingConventions :

the system allows calling all classes from almost all places of the code. For this, we are should properly name the classes and put the file which contains the class into a particular directory. Another goal of such conventions is to avoid conflicts between classes of different extensions. Through the observation on the application of Al-Rajhi, the classes names were just used alphabetical letters that do not represent the content of the code in that class, and this is what made studying the code and browsing the classes somewhat complicated. And that is a HIGH-risk problem in the category Code Style.
Also, the code content another problem from
category NamingConventions is MethodNamingConventions that is not followed to the naming laws like (Pascal case)

DataflowAnomalyAnalysis :

In a data flow anomaly, it is an abnormal situation to successfully assign two values to a variable without using the first value. Similarly, it is abnormal to use a value of a variable before assigning value to the variable. The presence of a data flow anomaly in a program

does not necessarily mean that execution of the program will fail, it simply means that the program may fail or Error-Prone.

UnusedImports :
The more the code is refined and limited to tasks without adding unused processes, the better the quality of the code, some unused calls were detected which is a violation of code style / UnnecessaryImport.

Vertical Openness Between Concepts :
All code is read left to right and top to bottom. Each line represents an expression or a clause, and each group of lines represents a complete thought.
Those thoughts should be separated from each other with blank lines vertically, which is not provided in alrajhi`s code, blank lines that separate the package declaration, the import(s), and each of the functions looked like a muddle. As I scan down alrajhi`s code, my eye is looking for the first line that follows a blank line. Personally, I recommend this extremely simple rule because it has a profound effect on the visual layout and readability of the code especially when you unfocused your eyes.

Horizontal Alignment :
In the list of declarations if you try to line up all the variable names in a set of declarations you are tempted to read down the list of variable names without looking at their types.
To make matters worse, most of the large company's developers use the automatic reformatting tools, likewise, alrajhi`s code does which is usually delimitating this kind of alignment. I do not recommend this kind of alignment because it emphasizes the wrong things and leads my eye away from the true intent. So, instead, Nowadays unaligned declarations and assignments are better because they point out an important deficiency.

Rule	**AvoidFileStream**		
Category	Performance		
Priority	HIGH	severity	Error
Massage	Avoid instantiating FileInputStream, FileOutputStream, FileReader, or FileWriter		
Description	The FileInputStream and FileOutputStream classes contains a finalizer method which will cause garbage collection pauses. See [JDK-8080225](https://bugs.openjdk.java.net/browse/JDK-8080225) for details. The FileReader and FileWriter constructors instantiate FileInputStream and FileOutputStream, again causing garbage collection issues while finalizer methods are called. * Use `Files.newInputStream(Paths.get(fileName))` instead of `new FileInputStream(fileName)`. * Use `Files.newOutputStream(Paths.get(fileName))` instead of `new FileOutputStream(fileName)`. * Use `Files.newBufferedReader(Paths.get(fileName))` instead of `new FileReader(fileName)`. * Use `Files.newBufferedWriter(Paths.get(fileName))` instead of `new FileWriter(fileName)`. Please note, that the `java.no API does not throw a `FileNotFoundException` anymore, instead it throws a `NoSuchFileException`. If your code dealt explicitly with a `FileNotFoundException`, then this needs to be adjusted. Both exceptions are subclasses of `IOException`, so catching that one covers both.		
Path	com.alrajhiretailapp/src/java/cl/json		
On element	d.java e.java	Line	line 137 line 159
example	// these instantiations cause garbage collection pauses, even if properly closed FileInputStream fis **= new FileInputStream(fileName);** FileOutputStream fos = new FileOutputStream(fileName);		

```
FileReader fr = new FileReader(fileName);
FileWriter fw = new FileWriter(fileName);

// the following instantiations help prevent Garbage Collection pauses, no finalization

try(InputStream is = Files.newInputStream(Paths.get(fileName))) {
}
try(OutputStream os = Files.newOutputStream(Paths.get(fileName))) {
}
try(BufferedReader br = Files.newBufferedReader(Paths.get(fileName), StandardCharsets.UTF_8)) {
}
try(BufferedWriter wr = Files.newBufferedWriter(Paths.get(fileName), StandardCharsets.UTF_8)) {
}
```

Rule	ClassNamingConventions	tool	PMD
Category	Code Style		
Priority	HIGH	severity	Error
Massage	The class name 'a' doesn't match '[A-Z][a-zA-Z0-9]*'		
Description	Configurable naming conventions for type declarations. This rule reports type declarations that do not match the regex that applies to their specific kind (e.g. enum or interface). Each regex can be configured through properties. By default, this rule uses the standard Apex naming convention (Pascal case).		
Path	com.alrajhiretailapp/src/java/cl/json		

On element			
	a.java	Line	17 , 23, 24 , 38 ,
	b.java		14, 21
	c.java		14-23
	d.java		17,33,36
	e.java		17.42
	f.java		17
	g.java		17
	h.java		16
	i.java		18
	k.java		16
	l.java		16
	m.java		16
	n.java		16
	o.java		15
	p.java		12
	r.java		25
	s.java		55 -218

example	public class FooClass { } // This is in the pascal case, so it's ok
	
	public class for class { } // This will be reported unless you change the regex

Rule	**MethodNamingConventions**			
Category	Code Style			
Priority	HIGH		severity	Error
Massage	The instance method name 'B' doesn't match '[a-z][a-zA-Z0-9]*' The {0} name ''{1}'' doesn''t match ''{2}''			
Description	Configurable naming conventions for method declarations. This rule reports method declarations that do not match the regex that applies to their specific kind (e.g. static method, or test method). Each regex can be configured through properties. By default, this rule uses the standard Apex naming convention (Camel case).			
Path	com.alrajhiretailapp/src/java/com/aurelhubert/ahbottomnavigation			
On element	VerticalScrollingBehavior.java	Line	35-39-41-43	
	AHBottomNavigationFABBehavior.java		45-55-82-86-90-94-98-103-212- 118-139-154-162 190-197-204-208	

example	public class Foo {
	public void instanceMethod() { } // This is in camel case, so it's ok
	
	public void INSTANCE_METHOD() { } // This will be reported unless you change the regex			

Rule	DataflowAnomalyAnalysis	tool	PMD
Category	Error Prone		
Priority	LOW	severity	Info
Massage	Found 'DD'-anomaly for variable 'bl' Found 'DU'-anomaly for variable 'string2'		
Description	The dataflow analysis tracks local definitions, undefinitions, and references to variables on different paths on the data flow. From that information, there can be found various problems. 1. DU - Anomaly: A recently defined variable is undefined. These anomalies may appear in the normal source text. 2. DD - Anomaly: A recently defined variable is redefined. This is ominous but doesn't have to be a bug. This rule is deprecated. Use {% rule "java/bestpractices/UnusedAssignment" %} in category bestpractices instead.		
Path	com.alrajhiretailapp/src/java/com/huawei/hms/adapter/ui		
On element	UpdateAdapter.java	Line	63 – 66 – 77
	AHBottomNavigationBehavior.java		218
	AvailableAdapter.java		78

example	public void foo() {
	  int buz = 5;
	  buz = 6; // redefinition of buz -> dd-anomaly
	  foo(buz);
	  buz = 2;
	} // buz is undefined when leaving scope -> du-anomaly

Rule	**UnusedImports**	**tool**	**PMD**
Category	**Best practices**		
Priority	LOW	severity	Warning
Massage	Unused import ''{0}''		
Description	R Reports import statements that are not used within the file. This also reports duplicate imports, and imports from the same package. The simplest fix is just to delete those imports. This rule is deprecated since PMD 6.34.0. Use the rule {% rule "java/code style/UnnecessaryImport" %} from category code style instead.		
Path	net.sourceforge.pmd.lang.java.rule.bestpractices.UnusedImportsRule		
On element	raspSDKcallback.java		

example	import java.io.File; // not referenced or required
	import java.util.*; // not referenced or required
	
	public class Foo {}		

severity	Warning			tool	Check style
Path	com.alrajhiretailapp/src/java				
On element	yz.java	line	21	Description	'ctor def modifier' has incorrect indentation level 4, expected level should be 2.
	ae3.java		59		'for' child has incorrect indentation level 12, expected level should be 6.
	mau.java		28		'extends' has incorrect indentation level 4, expected level should be 8.
	d.java		72		'else' child has incorrect indentation level 12, expected level should be 6.

severity	Warning		tool	Check style
Type	Formatting		Vertical Openness Between Concepts	
			Horizontal Alignment	
Path	com.alrajhiretailapp/src/java			
On element	Almost all classes	Description	Not adjusting horizontal alignment and vertical line spacing	

3.3 Manual Analysis Result

3.3.1 Formatting

Standard Code Structure

Following the usual standard code structure helps you read the code smoothly without any confusion or misunderstanding about missing brackets or semicolons.
as we are observing Alrajhi's code, obviously we know that it is correctly executed without any errors, but we had a lack of understanding about the unstandardized structure of curly brackets that were open and closed without any function in between simply like this "{ }" or with a separated line, here are questions we asked ourselves that will clarify the importance of standard structure
Did the code make an error because of those brackets? NO
Did you double-check the code's correctness? YES
Did you wonder once you see it? YES
and the last question is, does it good practice? the existence of those curly brackets did not add any benefit indeed it is added inconsistent and nonstandard style so it is not a good practice for sure.
In conclusion, the last thing we want to do is add unnecessary structure that may distract us we would recommend deleting these nonstandard brackets to make the code coherent standard structure.

```
    public boolean h() {
        return !this.f() || !((Boolean)this.d()).booleanValue();
        {
        }
    }

    public boolean i() {
        return this.f() && (Boolean)this.d() != false;
    }

    public boolean j() {
        return !this.f() || ((Boolean)this.d()).booleanValue();
        {
        }
    }
}
```

3.3.2 Functions

Small

The first rule of functions is that they should be small. The Second rule of functions is that they should be smaller than that.

The purpose of that is to be easy to do read which leads to easy to document and that's our purpose. But the hardness may locate in how small it should be? How can we measure the small ability? Robert C. Martin agrees that 4 is a maximum nom. Of line.

From observing, some functions exceed +100 lines of code which effect the easiness of reading, understanding, and documenting it. E.g. (See com.alrajhiretailapp/src/java/a10.java method c)

We suggest following some proven principles that lead to minimizing nom. Of line, deleting unnecessary variables, using a proper Switching Statement; the mentioned function was incredibly suffered from too much number of If statement. Using suitable Switch Statements will make a difference. There's also a special principle in Switch Statement you can follow for more proficiency. (See Clean Code/Functions/ Switch Statements).

For sure **Redundancy** is one of the violation processes for small functions. We were surprised as a team that there's too many duplicated methods/variables for no reason! Here 's an example of a method that has two for loops doing the same operation.

```
public static int a(CharSequence charSequence) {
    IllegalArgumentException illegalArgumentException;
    int n;
    int n2 = charSequence.length();
    int n3 = 0;
    for (n = 0; n < n2 && charSequence.charAt(n) < 'Â€'; ++n) {
    }
    int n4 = n2;
    while (n < n2) {
        char c6 = charSequence.charAt(n);
        if (c6 < '\u0800') {
            n4 += 127 - c6 >>> 31;
            ++n;
            continue;
        }
        int n5 = charSequence.length();
        while (n < n5) {
            char c7 = charSequence.charAt(n);
            if (c7 < '\u0800') {
                n3 += 127 - c7 >>> 31;
            } else {
                n3 += 2;
                if ('\ud800' <= c7 && c7 <= '\udfff') {
                    if (Character.codePointAt((CharSequence)charSequence, (int)n) >= 65536) {
                        ++n;
                    } else {
                        throw new d01(n, n5);
                    }
                }
            }
            ++n;
        }
        n4 += n3;
        break;
    }
    if (n4 >= n2) {
        return n4;
    }
```

Figure 5-com.alrajhiretailapp/src/java/a01.java

The obvious wrong with this code is that it has 1st for loop waiting ‘n’ to equal ‘n2’, then starting from new ‘n’ they start a new while loop comparing ‘n’ with ‘n5’ which ‘n5 == n2’. Logically, programmatically this is not acceptable. No reason for duplication. Additionally.

Single Responsibility Principle

FUNCTIONS SHOULD DO ONE THING. THEY SHOULD DO IT WELL. THEY SHOULD DO IT ONLY.
So far, most of the functions do more than one level of abstraction. For a simple example, the below function does 2 levels of abstraction. For more enhanced, you could extract the If Statement into a function called "PareclWriteStrongBinder()" which's responsible to initialize the "Interface".
As a note, One Level of abstraction can be achieved if you can extract a part of your function and replace it with a function that is merely a restatement of its implementation without changing the function's job.

```
public static void c(Parcel parcel, IInterface iInterface) {
    if (iInterface == null) {
        parcel.writeStrongBinder(null);
        return;
    }
    parcel.writeStrongBinder(iInterface.asBinder());
}
```

Figure 6-com.alrajhiretailapp/src/java/a21.java method c (SRP)

Using Descriptive names

"You know you are working on clean code when each routine turns out to be pretty much what you expected." Half the battle to achieving that principle is choosing good names for small functions that do one thing.
Following the above principles will be easy to define a name for your function. The more your function is small and do one thing the more clearly to name it.
Unfortunately, Alrajhi's code has no single function with a clear name. They may have a security purpose but as coder reviewers, we denounce the way they define names, break the **explicit** of the code, especially if Alrajhi Bank has a huge group of programmers who will face a lot of difficulties to complete their group work.
For the bellow code, a sample of weird names is given to the function.

We highly recommend the simple and easy solution, defining a function do will lead your team to define an easy name to read, understand, and document.

```
public static /* synthetic */ WritableMap f(a a2) {
    return a2.h;
}

public static /* synthetic */ Promise g(a a2) {
    return a2.i;
}

public static /* synthetic */ void h(a a2) {
    a2.j();
}

private void j() {
    this.a = null;
    this.b = null;
    this.c = null;
    this.d = null;
    this.e = false;
    this.f = null;
    this.g = false;
    this.h = null;
    this.i = null;
}

private String k(File file) throws IOException {
    RandomAccessFile randomAccessFile = new RandomAccessFile(file, l.a((int)25116));
    byte[] arrby = new byte[(int)randomAccessFile.length()];
    randomAccessFile.readFully(arrby);
    return Base64.encodeToString((byte[])arrby, (int)0);
}

private PrintAttributes l() {
    if (Build.VERSION.SDK_INT < 19) {
        return null;
    }
    PrintAttributes.Builder builder = new PrintAttributes.Builder().setMediaSize(PrintAttributes.MediaSize.NA_LETTER);
    String string = l.a((int)25117);
    return builder.setResolution(new PrintAttributes.Resolution(string, string, 600, 600)).setMinMargins(PrintAttributes.Margins.NO_MARGINS).build();
}
```

Figure 7-com.alrajhiretailapp/src/java/a.java

Function Arguments

The principle states that the argument should be ideal. We mean by ideal is to have zero arguments, and it should be a comprehensive argument. We mean by that also that an argument

shouldn't include a specific class as a parameter, rather, having a superclass "String Buffer" or "Object class" will accept more subclass to its method.
The main purpose of this is to have an easy testing process. The more argument you have, the more complex test you will provide.

In total, Alrajhi's code may face problems in testing their code since the arguments are not comprehensive and have +2 arguments.
In the below example, there's an unused parameter which means the compiler will define unnecessary attributes each time the method calls without doing anything!.
Suggested solution for the below example: The "Iterable" class was used only for the iteration, where the "byte" class has probably the same functionality as The "Iterable" class in this situation. So using one of them will be enough to use. For sure "eq" attribute class should be deleted.

```
public void a(Iterable<byte[]> iterable, cs cs2, eq eq2) {
    for (byte[] arrby : iterable) {
        if (arrby.length < 14) continue;
        String string = new String(arrby, 0, 13);
        if (!l.a((int)11290).equals((Object)string)) continue;
        this.c((ur)new tr(arrby, 14), -1 + (arrby.length - 13), cs2);
    }
}
```

Figure 8-com.alrajhiretailapp/src/java/a10.java

We highly suggest the Alrajhi team focus on this point, since it will have a side effect including the damage of space/ time complexity.

In conclusion for this part, Functions should either do something or answer something, but not both. And following this rule will lead to following all previous attributes.

3.3.3 Comments

Comments

There are certainly times when code makes a poor explanation. Unfortunately, alrajhi code does so it is useful to provide basic information with a comment to support your code understanding, document it and makes it readable.

However, looking at alrajhi's code made us notice a number of classes that looked ambiguous with whole empty content, not even a small code, what makes things worst that it was not supported by any kind of comments or description to explain the intent behind the decision of the emptiness in those classes. In this case, the comments were necessary to let us know the intent of each class.
regardless of the "empty classes" let us talk about the main classes that for sure have content. (See com.alrajhiretailapp/src/java/a03.java).
as we mentioned above alrajhi's was poor code for explanation with no comments at all.
Assenting to Martin saying that the motivations of writing comments are bad code **"reference"**. When some modules look confusing and disorganized all we do is say that I will document that with comments which are NO, you better clean it than to comment it.
To be honest, alrajhi's code was a bad code without even comments describing that mess!
Our humble advice that alrajhi's code should work on the clearness of their code with few comments for documentation will be far superior. Rather than spend your time writing the comments that explain the mess you have made, spend it cleaning that mess.

3.3.4 Proper Data Structure

Using proper data structure will minimize Nom. Of line in the code, less time and space complexity and provide more tools to use.
From observing the code, it seems that there's a lot of array list usage. (See Figure3).
In java, using an array list means storing data sequentially. This means for any process e.g., Retrieving, searching, or deleting will cost O(n). instead, they can implement a "List" interface that has classes with the same process and the cost not exceeding O(1).
In Algorithms theory, we take care of complexity and computability. So, taking care of these details means the code has been written professionally in all aspects

4. Conclusion

This paper showed an evaluation of the quality of Al-Rajhi application from many aspects, including security and code analysis, and we discussed our point of view as software engineers and several recommendations that we believe will benefit developers and software engineers

As a result of what we have noticed that the application is completely acceptable, and as we mentioned above what it lacks in the basics in every aspect, we look forward to our evaluation and recommendations being the beginning of a change in its quality and development.

5. References

http://www.javadecompilers.com/
https://play.google.com/store/apps/details?id=com.ext.ui
http://en.wikipedia.org/wiki/Single_responsibility_principle
http://en.wikipedia.org/wiki/Single_responsibility_principle
https://www.codeproject.com/Tips/1000719/High-Cohesion-Low-Coupling-using-SOLID-Principles
http://www.androidtutorialshub.com/cleartext-http-traffic-not-permitted/
https://medium.com/android-news/android-activity-launch-mode-e0df1aa72242
https://ar.wikipedia.org/wiki/نظام_تسجيل_نقاط_الضعف_المشترك
https://pmd.github.io/latest/pmd_rules_java_performance.html#avoidfilestream
https://docs.oracle.com/javase/7/docs/api/java/io/FileInputStream.html

//TO BE MODIFIED LATTER