Advances in Science and Technology
Research Journal
Volume 10, No. 31, Sept. 2016, pages 247–254
DOI: 10.12913/22998624/64058

Research Article

COMPARISON OF APP INVENTOR 2 AND JAVA IN CREATING PERSONAL APPLICATIONS FOR ANDROID ON EXAMPLE OF A NOTEPAD

Radosław Kowalczyk[1], Łukasz Turczyński[1], Kamil Żyła[2]

[1] Student of Lublin University of Technology, Nadbystrzycka 36b, 20-618 Lublin, Poland, e-mail: radikowalczyk@gmail.com, lukasz2442@gmail.com

[2] Institute of Computer Science, Electrical Engineering and Computer Science Faculty, Lublin University of Technology, Nadbystrzycka 36b, 20-618 Lublin, Poland, e-mail: k.zyla@pollub.pl

Received: 2016.05.26
Accepted: 2016.07.05
Published: 2016.09.01

ABSTRACT

Nowadays, there is a common tendency to seek simpler and faster solutions that could be used in a process of software development. At the same time two other trends can be observed - rapid increase of mobile applications popularity and introducing IT concepts to non-IT specialists. This is why App Inventor 2 is one of the tools that software development community is interested in. The goal of this paper is to verify the possibility of using App Inventor as a tool for creating personal applications and compare it with possibilities given by Java, which is a native environment for Android platform. Comparison was based on application for managing personal text notes which was created in both solutions. The application had the same layout as well as analogical code, and it was the subject of performance tests followed by a survey. Performance test revealed that both implementations provided efficiency which is good enough for everyday use and small size personal applications. Survey participants preferred application built in Java due to its better responsiveness and visual appearance. Concluding, current shape of App Inventor, makes it useless for professionals, but useful for non-IT specialists for creating personal applications.

Keywords: Java, App Inventor, Android, efficiency comparison, application usability comparison.

INTRODUCTION

App Inventor was initially developed by Google which, at the beginning of 2012, transferred it to Massachusetts Institute of Technology – its current administrator [12, 15]. App Inventor tries to make Android applications development easier and faster, and involve not only people who are computer science experts [8]. Its homepage [2] claims that 4.7 mln registered users from 195 countries created 14.9 mln mobile applications (till the time of writing this paper).

On the other hand, classical method of creating mobile applications for Android platform is based on special version of Java and Android SDK. Almost unlimited capabilities of Java for Android lead to efficient and nice-looking solutions. This proven and flexible development platform is supported by many tools (e.g. Android Studio and Eclipse with ADT) and vast community of programmers. Although, it requires object-oriented programming skills, that not everyone is able to possess [1, 5, 11].

When it comes to App Inventor, it provides simpler syntax and development environment [16], although it offers significantly limited functionality. Applications are compiled to "*.apk", the same as in case of the classical method (Java for Android) and can be added to Google Play [13]. In opposite to Java, there is one integrated development environment, that allows to create applications through the web – project files and compilation is moved from a developer's PC to the web.

Lowering entry barrier and letting non-IT experts to program is a factor that makes App Inventor interesting. Another reason to write this paper

was a small number of works investigating performance issues and the results of simplifications as well as limited functionality. Most of publications focuses on aspects concerning didactics and fast prototyping [7, 9, 14, 17].

Due to the abovementioned reasons and high popularity of App Inventor, the authors decided to verify whether App Inventor is able to provide functionality sufficient to produce everyday applications for personal use. Another goal was to check how it compares to native development environment like Java for Android.

The following research hypotheses were formulated:

- H1. Performance of application produced using App Inventor is as good as in case of Java for Android.
- H2. Look and feel of application produced using App Inventor could be as good as in case of Java for Android.

Look and feel should be considered as a concept combining responsiveness of a user interface, convenience of navigation, as well as nice and aesthetical appearance, that are also elements of the usability concept [3, 4].

In order to verify the formulated hypotheses, the authors designed and developed an exemplary application (later called "Notepad") for everyday personal use, that allowed to manage personal notes. App Inventor and Java implementation shared similar layout and functionality. Then, the application was the subject of performance test (speed of performing leading operation) to verify H1 and the survey to verify H2. In general, the purpose of the test and survey was to check whether App Inventor is able to produce application, that will be good enough in comparison to Java (keeping in mind its purpose – being simple and for personal everyday use) and whether there will be significant differences in quality.

INTRODUCTION TO THE "NOTEPAD" APPLICATION – DEVELOPER'S PERSPECTIVE

Application design

Dissimilarity of App Inventor and Java caused that authors had to choose a subset of functionality that could be obtained in both technologies in a similar way, which was also important due to the performance test. Therefore, application allows to display, add, edit and delete notes that are stored in offline database typical for each solution.

The application is consisted of four screens:

1) Screen for displaying the list of notes (mockup presented in Figure 1) – screen presents the list of notes. After long click on a note, it could be deleted or edited on another screen. On the top of the screen there are two buttons – for displaying application credits and opening another screen for adding new note.
2) Screen for adding new note (mockup presented in Figure 2) – screen contains three elements: text field for a note name (note caption which is visible on the list of notes), text field

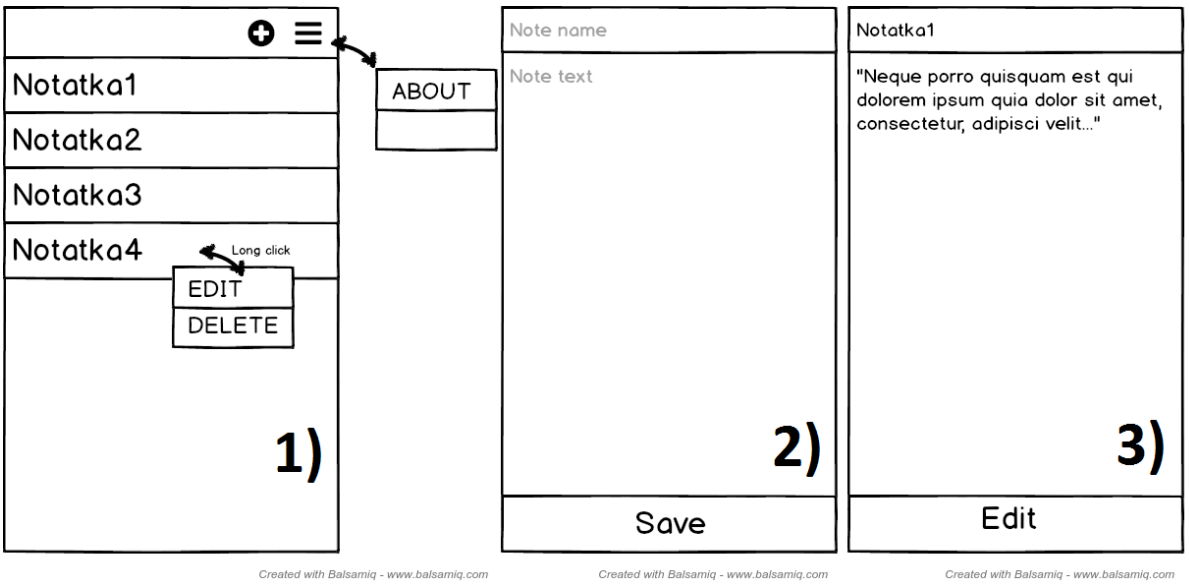

Fig. 1. Mockups of the Notepad application: 1) screen displaying the list of notes; 2) screen for adding new note; 3) screen for editing note

for the text of note, and button saving it to a database.

3) Screen for editing a note (mockup presented in Figure 3) – screen shares layout with the screen for note addition. The only difference is button name and purpose of saving changes to an existing note.
4) Screen for displaying a note – screen shares layout with the screen for note addition.

The authors chose the subset of layout elements common for App Inventor and Java in order to ensure equal chances for both solutions during the survey concerning look and feel of developed application. The only difference was context menu appearing after long click, which is such a basic layout element that it could not be omitted. Unfortunately, App Inventor did not provide such a functionality.

Application implementation

The process of application implementation confirmed that Java for Android, as a dedicated development environment, has superior capabilities comparing to App Inventor. Unfortunately, the latter one lacks functionality which is considered basic for Android platform. Nevertheless, in case of simple applications for everyday personal use, App Inventor usually provides sufficient functionality or another way ("bypass") that allows to obtain similar goal/effect.

When it comes to developing the layout of the application, App Inventor provides small number of simple visual component. The authors, in opposite to Java, had problems with achieving all visual effects (layout in general) designed on mockups. The biggest issue was lacking longClick event issued on the list of notes and displaying context menu. It was bypassed by adding "Delete button" at the bottom of the screen for editing notes; choosing a note from the list redirects to the mentioned screen. Main menu had to be constructed manually, without any additional facilitations, by adding buttons to the horizontal layout. Moreover, it is not possible to programmatically create layout elements (e.g. creating a button depending on user action, instead of showing/hiding it is not possible; button has to be created at the application start and then showed or hidden). At last, the look of generated user interface is more primitive in App Inventor (missing advanced settings concerning gradients, edges, etc.). Some of the mentioned issues could be observed in Figures 2 and 3.

When it comes to implementation, Java for Android requires fair object-oriented programming skills, even in case of simple applications for everyday use. On the other hand, App Inventor provides simplified development environment and syntax that is easier to grasp for non-IT specialists. Listing 1 and Figure 4 is an example of the same functionality (addition of 100 notes to a database in a key-value manner) developed in both solutions.

Application in Java easily exceeded few hundreds lines of code distributed among dozen

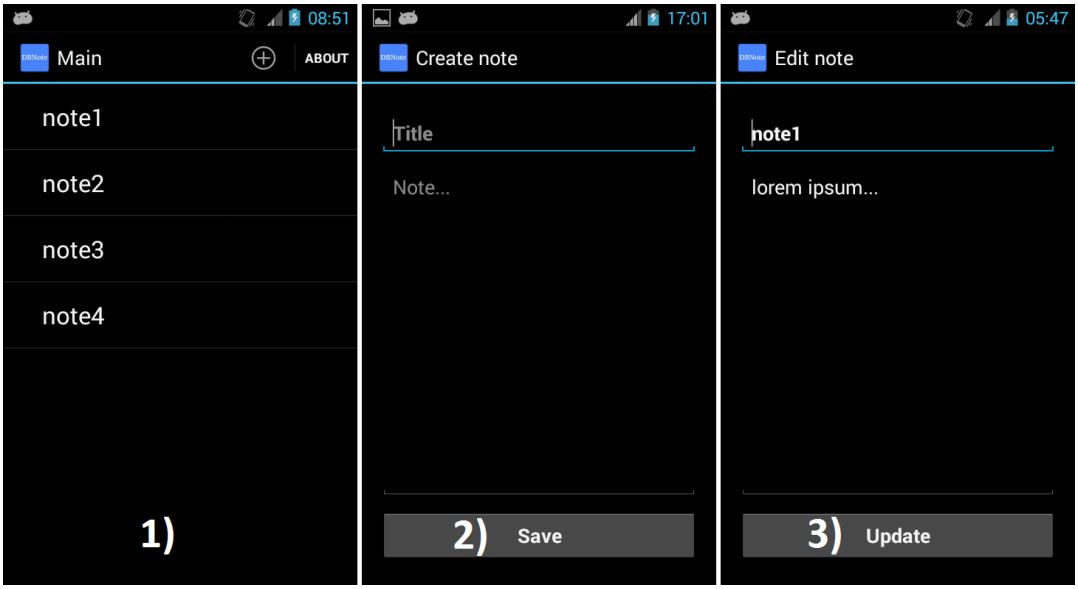

Fig. 2. Layout of the Notepad application in Java: 1) screen displaying the list of notes; 2) screen for adding new note; 3) screen for editing note

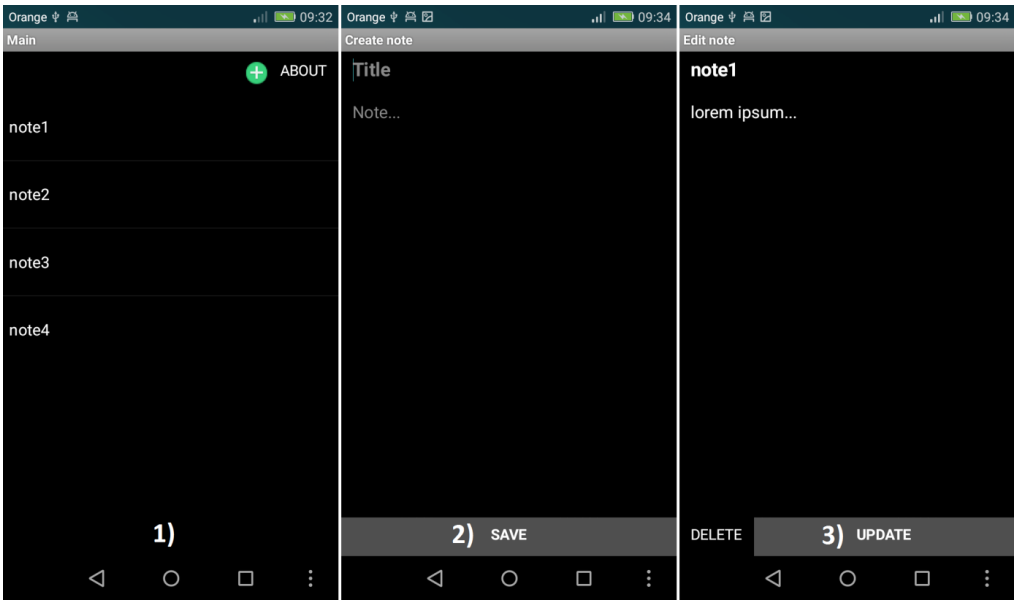

Fig. 3. Layout of the Notepad application in App Inventor – 1) screen displaying the list of notes; 2) screen for adding new note; 3) screen for editing note

files, which could be seen as a proof of bigger effort needed to develop the application. Nevertheless, it cannot be forgotten that noticeable amount of that code is generated automatically by an integrated development environment. Moreover, in opposite to App Inventor, developer could utilize many editors and easily reuse the application code.

Basing on the given example, it can be assumed that implementation of applications in App Inventor would rather be concise and easy to read, which is also supported by the presence of color patterns and searching mechanism, that facilitate finding particular elements of implementation. Unfortunately, it is true only for rather small/simple applications. Developing larger ones require special policy concerning model creation and organization (distribution of the model elements across the workbench becomes vital). Very often model becomes too big to manage it conveniently e.g. instructions are wider than work area of a screen and zooming out makes captions illegible, as well as there is no project tree that allows to jump directly to the particular element instead of scrolling large parts of the model. At last, App Inventor projects could be opened by only one editor, and that editor does not allow convenient code reusability.

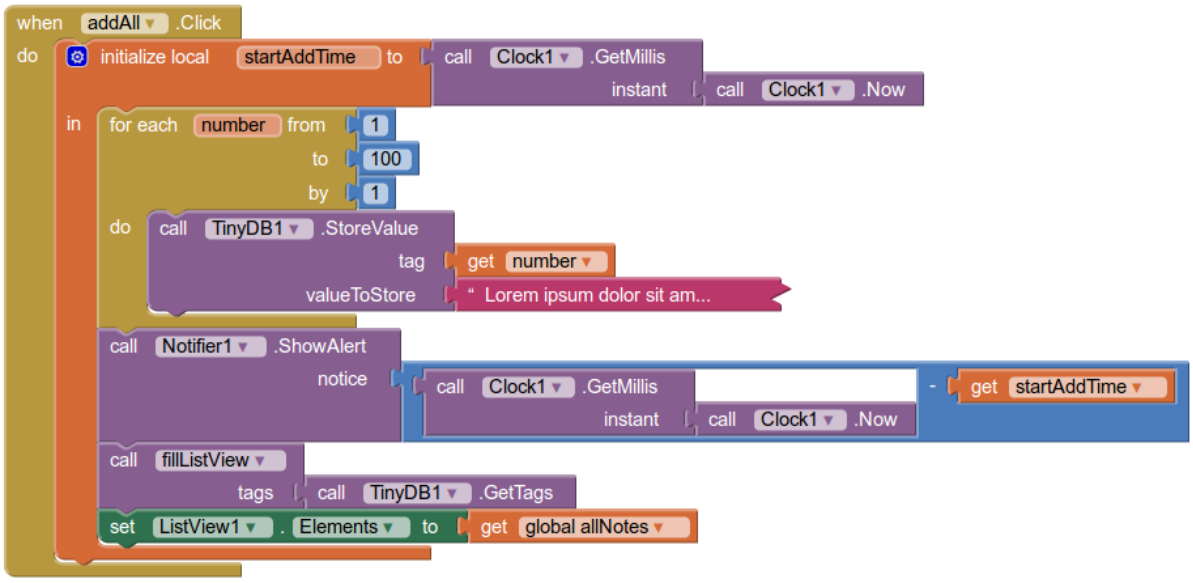

Fig. 4. Code/Model in App Inventor counting time of adding 100 notes

```
public class Tester {
 public static void addAllNotes(DBAdapter dbAdapter, Context context){
 long startTime = System.currentTimeMillis();
 for (Integer i = 0; i < 100; ++i) {
 dbAdapter.insertNote(i.toString(), "Lorem ipsum dolor ...");
 }
 long endTime = System.currentTimeMillis();
 Toast.makeText(context, "That took " + (endTime - startTime) + " milliseconds",
Toast.LENGTH_LONG).show();
 }
 // .....
}

public long insertNote(String title, String content) {
 ContentValues newValues = new ContentValues();
 newValues.put(KEY_TITLE, title);
 newValues.put(KEY_CONTENT, content);
 return db.insert(DB_TABLE, null, newValues);
}
```

Listing 1. Code in Java for Android counting time of adding 100 notes

The authors also noticed that, despite shorter implementation, "*.apk" file obtained from App Inventor is much larger (1.4MB) than in case of Java for Android (1MB). It is quite a big difference as the application is rather small. The authors checked implementation of App Inventor components (available at [6]), and the most probable reason is lack of optimization of the code generator.

PERFORMANCE TESTS

The leading operation for the Notepad is saving and deleting notes, thus the authors assumed that it could be used as a representative marker for the whole application. Therefore, in order to briefly compare the performance of the obtained executables, authors measured the:

- time needed to add 100 notes to database,
- time needed to delete 100 notes from database.

Each note consisted of 676 characters. Both tests were repeated 30 times for both implementations. Notes were added to empty database, and removed from database containing exactly 100 notes.

Timer was started before adding the first note and stopped after adding the last one. Listing 2.1 and figure 2.4 present the procedure of time measurement for notes addition, which is analogical as in case of notes deletion. Notes were managed in a key-value manner using default offline database: App Inventor – TinyDB; Java for Android – sqlite. Such databases, despite differences, were chosen on purpose, to check behavior of each solution correlated with its typical (default) database. All tests were performed on the device running Android OS 4.3.1, equipped with 2-core CPU (800MHz) and 768MB RAM.

Those tests indirectly indicate the ability of Java for Android compiler and App Inventor code generator to produce efficient code. Moreover, the goal of the tests is to determine if the efficiency of application produced by App Inventor is good enough in everyday use comparing to Java.

RESULTS OF TESTS

When it comes to addition of 100 notes App Inventor is slightly faster (of 477.4 ms in average) than Java for Android (Fig. 5). Similarly, when it comes to deletion of 100 notes – the difference is 810,5 ms in average (Fig. 6). It is most likely caused by differences between databases, because Java for Android is often much faster, e.g. [17]. Nevertheless, such small difference in performance is not sufficient to state that any solution is significantly worse in everyday use, because its

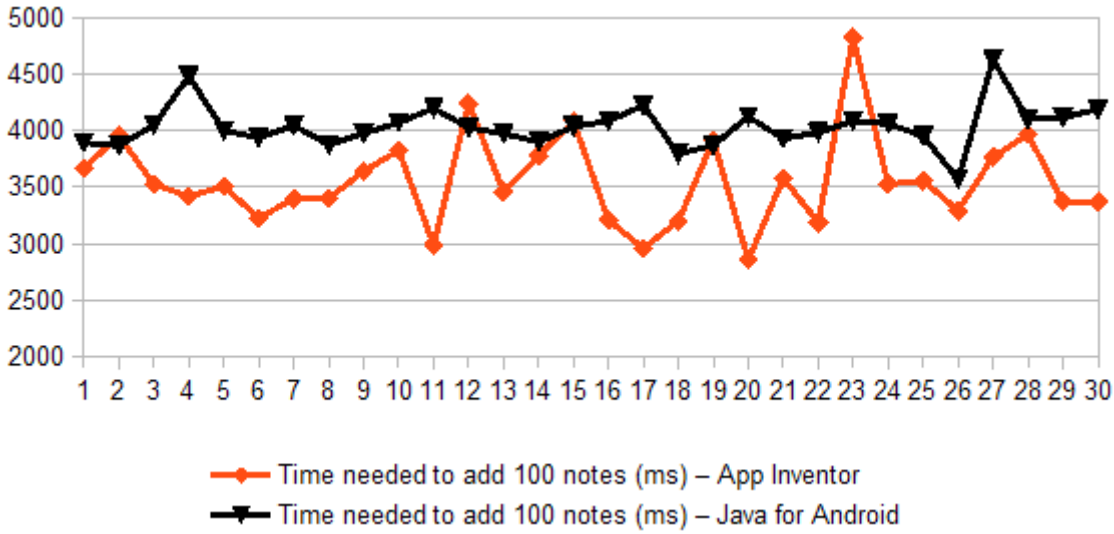

Fig. 5. Time of notes addition

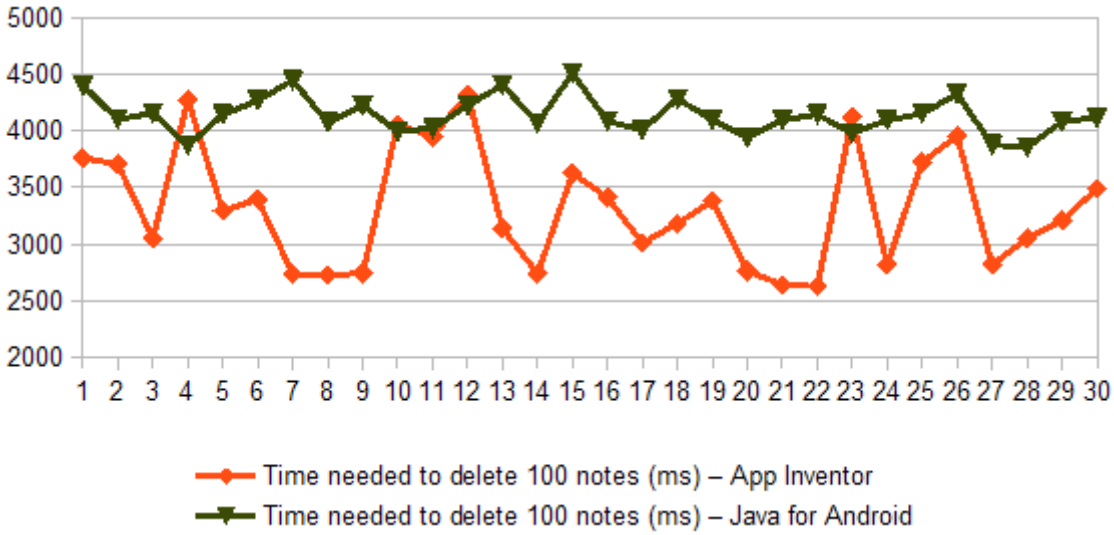

Fig. 6. Time of notes deletion

Table 1. Times measured during performance tests (ms)

Parameter	App Inventor		Java for Android	
	Notes addition	Notes deletion	Notes addition	Notes deletion
Arithmetic mean	3 556.6	3 324.0	4 034.0	4 134.5
Standard deviation	409.3	523.5	194.5	167.5
Median	3 515.5	3 252.5	4 0300	4 099.5
Average deviation from median	299.7	439.8	131.4	124.8

value was aggregated during 100 iterations, and user is working with one note from time to time. At last, the speed of application produced from Java code is more stable than in case of App Inventor – standard deviation and average deviation from median are much lower for Java for Android (see Table 1).

SURVEY

In order to compare look and feel of both implementations, short survey was conducted. Before survey, respondents had some time to work with each application. They have not been told, which application was developed using App Inventor

either Java for Android. They were distinguished only by numbers: number 1 – Java for Android; number 2 – App Inventor.

The survey consisted of the following questions:

- Is the application 1 readable for you?
- Is the application 2 readable for you?
- Are you fully satisfied by the GUI (Graphical User Interface) of application 1?
- Are you fully satisfied by the GUI of application 2?
- Is the application 1 sufficiently responsive for you?
- Is the application 2 sufficiently responsive for you?
- Which application (1 or 2) do you prefer?

RESULTS OF SERVEY

Authors tried to choose respondents without strong computer science background and finally gathered 30 surveys. The age structure of the respondents is presented in Table 2.

The results of the survey are presented in Figure 7. All respondents claimed that application implemented in Java for Android was readable, as well as sufficiently responsive. Most of them (26 persons) stated that GUI is nice looking and fully satisfying. App Inventor gathered noticeable worse opinions. Half of the respondents claimed that GUI is not fully satisfying and told authors that it is missing some layout elements and gestures facilitating navigation. About 23% of respondents stated that application implemented in App Inventor is not sufficiently responsive, as well as not readable enough. Generally, respondents preferred application developed in Java for Android (23 to 7 persons).

The obtained results are not surprising. When it comes to user interface and App Inventor, worse opinions are most likely caused by limited set of gestures and poor functionality concerning visual aspects, like colors and shapes of edges, small palette of colors, etc.. In case of both implementations, the respondents during the survey did not claim that the number of saved notes influenced the efficiency of the application, however, their number was rather small. Moreover, some of them complained that switching to another screen lasted noticeably longer than in case of Java for Android, which impaired general satisfaction.

CONCLUSIONS

The goal of this paper was to check the possibility of using App Inventor as a tool for creating personal applications, by comparing it with

Table 2. Age structure of the respondents

Years old	<20	20–25	26–30	31–40	40<
Number of respondents	2	20	4	0	4
Percentage of respondents	6.7%	66.7%	13.3%	0%	13.3%

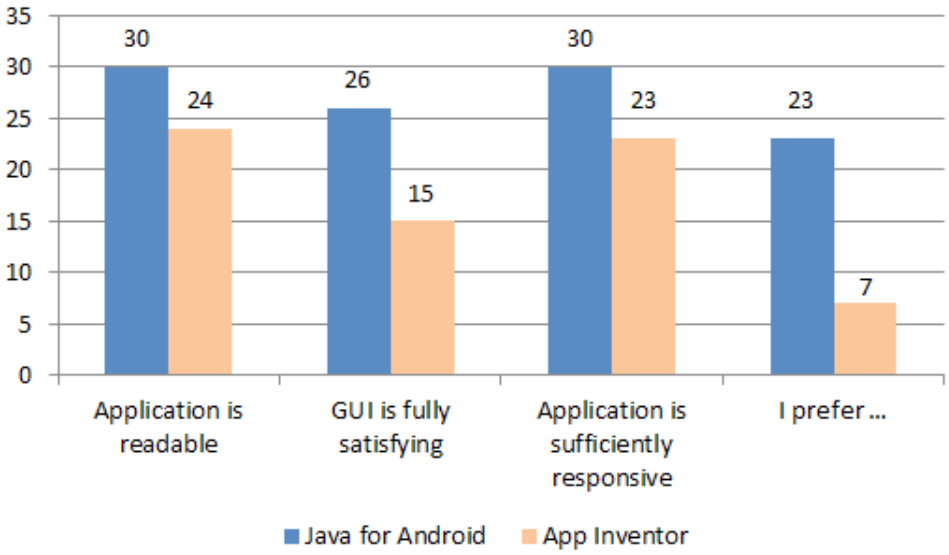

Fig. 7. Survey results

Java for Android. Application developed using App Inventor successfully met the requirements of displaying, saving, writing and deleting notes. However, it turned out that despite a large number of build-in objects, a lot of useful visual controls and events, that are available in Java and useful in creating Notepad, are missing (disadvantage common for Model-Driven Engineering solutions [10]). Moreover, necessity to implement whole functionality of the application screen in one view (container) aggregates a lot of building blocks in one place, which makes the code less clear, more vulnerable to errors and more difficult to maintain. Results of the survey, regarding appearance of both applications, confirmed deficiencies of App Inventor in creating complex and intuitive user interfaces.

App Inventor can be recommended for people without special programming skills, although for simple personal projects only. It is not suitable for creating business applications In this case, the advantage of Java, with regard to the capabilities and efficiency, is indisputable.

Summarizing, performance of both applications was good enough for everyday use, although App Inventor might be significantly slower in case of more complex applications (H1). Currently, look and feel of applications produced using App Inventor cannot be as good as in case of Java for Android (H2).

Concluding, App Inventor allows to develop uglier applications, although generally readable as well as sufficiently responsive, thus able to fulfill their purpose – being simple and for personal everyday use.

REFERENCES

1. Android developers homepage, developer.android.com [21.05.2016].
2. App Inventor homepage, http://appinventor.mit.edu [21.05.2016].
3. Borys M., Miłosz M., Mobile application usability testing in quasi-real conditions a case study of a mobile eye tracker. 8th International Conference on Human System Interactions. IEEE, 2015.
4. Borys M., Plechawska-Wójcik M., Usability and accessibility testing of mobile application interfaces. Nierówności społeczne a wzrost gospodarczy, 35, 2013, 63-77.
5. Conder S., Darcey L., Android wireless application development. Addison-Wesley, 2011.
6. Google code archive: App Inventor for Android, https://code.google.com/p/app-inventor-for-android/ [21.05.2016].
7. Grover S., Pea R., Using a discourse-intensive pedagogy and android's App Inventor for introducing computational concepts to middle school students. Proc. of the 44th ACM technical symposium on Computer science education. ACM, New York 2013.
8. Jordan L., Greyling P.: Practical Android Projects. Apress, 2011.
9. Karakus M., Uludag S., Guler E., Turner S. W., Ugur A., Teaching computing and programming fundamentals via App Inventor for Android. 2012 International Conference on Information Technology Based Higher Education and Training. IEEE, 2012.
10. Kęsik J., Żyła K., Współczesne Technologie Informatyczne. Technologie MDE w projektowaniu aplikacji internetowych. Politechnika Lubelska. Lublin, 2011.
11. Kopniak P., Programming interfaces of accelerometers for Smartphone type mobile devices. Pomiary automatyka kontrola, 12, 2011, 1477-1479.
12. Mitchell J., MIT launches Center for Mobile Learning with support from Google. readwrite, 2011. http://readwrite.com/2011/08/16/mit_launches_center_for_mobile_learning_with_suppo [22.05.2016].
13. Publishing Apps to Google Play (App Inventor 2), http://appinventor.mit.edu/explore/ai2/google-play.html [21.05.2016].
14. Roy K., Rousse W.C., DeMeritt D.B., Comparing the mobile novice programming environments: App Inventor for Android vs. GameSalad. Frontiers in Education Conference. IEEE, 2012.
15. Wolber D., App Inventor discontinued: the good, the bad and the ugly. App Inventor blog, 2011. https://appinventorblog.com/2011/08/09/app-inventor-discontinued-the-good-the-bad-and-the-ugly/ [21.05.2016].
16. Wolber D., Abelson H., Spertus E., Looney L., App Inventor: Create your own Android apps. O'Reilly Media, 2011.
17. Żyła K., Wydajność implementacji podstawowych metod całkowania w środowisku APP Inventor. Informatyka, automatyka, pomiary w gospodarce i ochronie środowiska, 1, 2015, 45-48.