From App Inventor to Java: Introducing Object-oriented Programming to Middle School Students Through Experiential Learning

Dr. Farzana Rahman, Florida International University

Farzana Rahman is a faculty of School of Computing and Information Sciences at Florida International University (FIU). Before joining FIU, she was an assistant professor at James Madison University (JMU). She is the director of the first REU program hosted by JMU during summer 2017. She designed and delivered courses on mobile development that involved critical research challenges of mobile computing area. She has mentored over 10 undergraduate students through research projects and honor thesis, the majority in the areas of mobile computing and mHealth. Her efforts over the last several years have led to several papers published in top ACM and IEEE conferences with undergraduate co-authors. Her field of interest encompasses Security, Trust and Privacy in Pervasive Computing, Internet of Things (IoT), Mobile Computing, CS education, and Mobile Healthcare Privacy. She has ben very active in broadening participation of women and underrepresented minority in computer science. She has also been working as an active member of various international conference technical program and journal review boards. She Additionally, she has served as Co Chair of IEEE PerIoT 2017 and 2018, IEEE eIoT 2017, IEEE STPSA 2014, 2015, 2016, and ACM CAPWIC 2017 conferences. She is also the winner of the ABI sponsored Fall 2014 Systers PIO award. She is the founder of CS4VA initiative, a one-day CS outreach program dedicated to build a Virginia statewide network for creating a more diverse CS K-12 pipeline.

From App Inventor to Java: Introducing Object Oriented Programming to Middle School Students through Experiential Learning

1. INTRODUCTION

Computing in today's modern world has become one of the most important skills. Hence, strategic computing education efforts are needed to prepare the next generation so they can be successful in the workplace as well as in higher education pursuits. These efforts need to equip students with computational thinking skills so they can solve problems in different aspects of their life. The growing trend in introducing computing to K-12 curriculum is one of many such efforts that have revolutionized K-12 education in recent years.

However, research shows that K-12 students find computer programming significantly harder comparing it with other academic fields [1]. There is enough evidence that shows many novice programmers at K-12 level, experience difficulties with learning programming concepts and applying those concepts in solving problems. When it comes to learning Object-Oriented Programming (OOP) languages like Java or C++, it challenges students to master programmatic overhead before programming itself. Finally, researchers also assert that traditional programming courses fail to connect computing concepts with young students' diverse interests [2, 3].

To ease the process of learning programming and making it engaging and accessible to a broader population many visual programming tools, especially block-based languages, have been developed [17]. In the category of block-based languages, MIT App Inventor (AI) has been used by educators, developers, and/or hobbyists, to develop mobile applications for personal use, recreation, learning, or social good [13]. Additionally, academics have successfully used AI in their courses to introduce programming to K-12 students for last several years [4].

However, more recently, CS education researchers are beginning to recognize the need to apply the learning sciences to develop age- and grade-appropriate curricula and pedagogies for building computational competencies among children. One effective approach to build learning competencies among young children is through *Experiential Learning* [6], which is the process of learning from experience, a methodology in which educators engage with students in direct experience to increase knowledge, develop skills, and clarify values. The abilities gained through experiential learning (remember, understand, apply, analyze, evaluate and create) are inspired from Bloom's Taxonomy [5] which will be the skills of a literate twenty-first-century citizen. Hence, our goal in this project was to develop a programming curriculum for middle school students that explores the benefits of using experiential learning through a common set of examples and a project to support the transition from a visual to textual programming language, in our case an object-oriented language, Java.

In this paper, we present our experience in designing a 2-week block course curriculum that teaches OOP fundamentals, through App Inventor 2 (AI2) and Java, to middle school students. The curriculum emphasizes computing concepts like data types and its usage, programming structures like sequential and conditional execution, repetition, procedures, etc. Once students are comfortable with programming concepts through AI2, we present our design of a computing curriculum that uses Java as the object-oriented programming language. The design of Java curriculum draws several lessons from AI2 curriculum and builds on top of programming concepts that students are already familiar with. The App Inventor to Java approach, along with the use of experiential learning, enables students to learn OOP concepts and stay motivated. Our hypothesis

was that using experiential learning to transfer knowledge from App Inventor would improve students' achievement in learning OOP concepts in Java.

Paper organization: The rest of the paper is organized as follows – in Section 2, we present relevant related works, pros and cons of using visual and textual programming language, and why we choose App Inventor and Java as the representatives for our curriculum. Section 3 presents an overview the curriculum and how we integrated experiential learning into it. Section 4 and 5 explains the details of the curriculum for AI2 and Java respectively. Section 6 briefly presents our findings from the evaluation. Finally, section 7 presents conclusions and discussions on future similar curricula offerings.

2. RELATED WORK ON VISUAL AND TEXTUAL PROGRAMMING LANGUAGE

In recent years, industry and academic community have made significant efforts in developing tools and applications to introduce computing education to the young generation. In this category, block based programming languages, also known as Visual Programming Language (VPL) [17], have drawn much attention. VPL is usually recommended to be used by young learners since it provides a graphical output which helps to maintain a positive impression of programming in the long run compared to Textual Programming Languages (TPL) [7].

Alice [12], Scratch [14] and App Inventor [13] are one of the most successful VPLs due to their widespread use by the educational community, especially at the K-12 level. There have been several studies comparing the user experience of students using VPL verses TPL. Most of these studies found that the participants not only felt more confident to modify VPL code, they also had a more positive experience than when using the TPL [8]. Furthermore, a recent study had experimented teaching VPL in middle school then teaching TPL in high school. They found that the students' relational thinking significantly improved when they had the VPL background [9], which contributed towards more self-confidence and motivation in students. Even though VPL is not considered a professional programming language, it has been reported to provide personal and intuitive experience [10]. Moskal et al. [21] developed an introductory college curriculum using Alice and compared students who took their course to students who took only the CS1 course. There are also many efforts to increase the number of K-12 students exposed to computing such as the CS10K project [25], the new AP CS Principles Course [23], and a new effort with code.org [22] that has gotten millions of students trying an hour of programming by using their one-hour tutorials. Many projects target middle school including CS Unplugged [24] and some projects are focused on a specific discipline like integrating math with computing at the K-12 level [26].

In [15], a summer camp curriculum was presented where the students were taught app development using App Inventor and later introduced to Java. However, this camp's focus was to teach app development (not programming only) in a particular platform to high school students only. Even though they have found the camp to be successful in using APP Inventor and transitioning to Java, their curriculum followed traditional learning techniques and most of their students already had prior Java experience which played a major role in student's learning experience. The primary difference between this and our curriculum is our approach of presenting a visual language followed by a strategic transition to OOP, in this case Java.

In [11], a curriculum was presented where middle school students use Alice at the beginning and later they take Python in higher classes to learn computing concepts in the context of a text-based programming language. Following the work of [11], in our course, we choose to use App Inventor as the VPL and later switch to Java. However, our curriculum is different than [11] for 3 reasons – 1) instead of using traditional teaching pedagogy, we used experiential learning strategy to

introduce programming to middle school kids, 2) instead of an entire semester long course, our curriculum is designed to be delivered as 14-day block course covered over a period of two semesters and 3) we used AI2 and Java instead of Scratch and Python, respectively. Particular reasons to choose these two specific languages are briefly presented in the following subsections.

2.1 App Inventor

App Inventor 2 (AI2) is a visual programming environment for creating mobile applications for Android OS. The AI2 environment is supported for the three known (Windows, iOS, and Linux) operating systems, and the resulting apps can be installed on any smart mobile device running Android OS. It also comes bundled with a smartphone emulator for live testing and debugging. AI2 also provides databases, geolocation, audio and video contents and other features. The concepts of lists and lists of lists are similar to arrays and multi-dimensional arrays, respectively. Despite of numerous benefits, AI2 does have some limitations. It does not allow creation of new components (new classes) and does not allow control over the priorities of events. In addition, in large projects with a lot of functions and variables, it is hard for a student to maintain them, because there are no hierarchies, except the option to minimize the procedures. AI2 does not currently offer a means of sharing projects or reusing parts of code across projects. This is extremely inefficient, and prevents students from reusing code in multiple projects and sharing with others. From a programming point of view, there is no opportunity to define a local variable except in 'for each' and 'for range' loops or global variable.

However, one of the major pedagogical advantages of working with AI2 is that students can develop their applications as well as do live testing. It allows creation of creative mobile applications while still engaging with complex computational concepts, including procedural and data abstraction, conditional and logical thinking, and debugging. The benefits of using AI2 are primarily in its use as an easy introduction to programming, because no extra coding knowledge is needed to program easily demonstrable results. It allows students to focus on the logic for programming rather than the strict syntax of a textual programming language.

2.2 Java as OOP Language

Table 1. Setup of Block courses in 7 modules and 4 experiential learning stages

Course	Module
Block 1 Course App Inventor	Variables and Data Types in AI2 Module
	I/O and Operators in AI2 Module
	Conditionals in AI2 Module
	Repetitions in AI2 Module
	Procedures in AI2 Module
	Lists in AI2 Module
	Objects in AI2 Module
Block 2 Course OOP (Java)	Variables and Data Types in Java Module
	I/O, Operators, Libraries in java Module
	Decisions in Java Module
	Repetitions and Nesting in Java Module
	Methods and Parameters in Java Module
	Arrays in Java Module
	Objects in Java Module

Historically, Java has been one of the most widely used object-oriented programming languages in the world [20]. We choose to introduce Java, instead of any other OOP, due to its wide appeal in

student community and its inherent analogy to various structures used in App Inventor. AI2 provides an effective object-oriented framework for designing visual interfaces and event-driven control structures. It presents application development as two main activities – analyzing what components (objects) make up the app and coding the actions (behavior) that these components (objects) take during the application's execution. This model provides a good introduction to the object-oriented paradigm and provides a good foundation for acquiring more advanced computing skills in future semesters.

3. CURRICULUM OVERVIEW

For our curriculum, we designed two block courses – a) *Block 1 – App Inventor* and b) *Block 2 – OOP using Java*. Each block course consisted of 28 hours of instructional time, divided between 7 days (7 Saturdays). We offered Block 1 for the first time in Spring 2014 where 30 middle school students (from grade 7 and 8) participated. Since these block courses were not offered as part of any regular school curriculum, students met every Saturday for 4 hours with the instructors. Block 2 course was offered to the same set of students during Summer 2014 following the same Saturday schedule. Apart from the in-class 4 hours of instructional time, students were not provided with any additional readings, assignments or tutorials to be completed at home. However, parents reported that many students continued to explore advanced concepts at home due to their own curiosity and interest.

We divided the curriculum of both courses in 7 modules so we could dedicate each Saturday to teach one of the modules. For each module of Block 1, our approach was to develop a small mobile application using AI2 to deliver the foundational concepts of programming, and show the equivalent implementation in Java later in Block 2 course. Our reasoning for this strategy is that the students would be able to form a mental mapping to the equivalent Java representation. We believe that the familiarity with the same components eased their transition from a visual language environment to a much harder OOP environment.

3.1 Integration of Experiential Learning

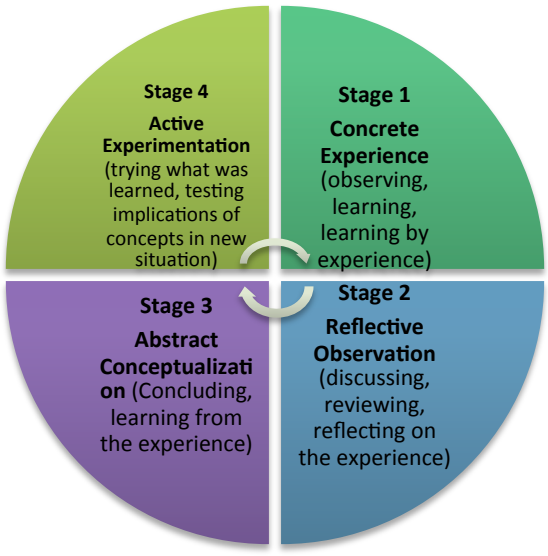

Figure 1. Kolb's Experiential Learning Cycle

David Kolb first proposed that experiential learning is multidimensional process following a cyclical model of different stages [Fig 1]. One major goal of our curriculum was to move away

from traditional learning setting and use experiential learning strategies where students try to solve authentic problems with an inquiry approach. We wanted the students to develop problem solving and self-directed learning abilities while they also remain motivated to learn increasingly challenging topics. To incorporate experiential learning in our curriculum, each module begins from concrete experience to reflective observation, then to abstract conceptualization to active experimentation. In other words, the first stage is where the learner actively experiences an activity. The second stage is when the learner consciously reflects back on that experience. The third stage is where the learner attempts to conceptualize a theory or a model of what is observed. The fourth stage is where the learner is trying to plan how to test a model or theory or plan for a forthcoming experience. In both of our courses, for each module, we tried to follow the same learning cycle very closely by dividing each day's instructional hour (4 hours) into 4 stages: 1) Concrete Experience: Mini Lecture and demo, 2) Reflective Observation: Demo practice individually by students, 3) Abstract Conceptualization: Team discussion, and 4) Active Experimentation: Team work to complete a module of the project [see Fig 1].

3.2 Learning Objectives

In our curriculum, through the 2 block courses, we wanted the students to learn the foundational concepts of object-oriented programming language, in this case Java. Even though AI2 is introduced first, we wanted students to internalize various structures of programming so later it is easier for them to transition to Java. The major learning objectives of our curriculum are:

1) Understand fundamentals of programming such as input/output, variables, data types, operators, expressions, conditional (branching), iterative execution, methods, etc.

2) Understand OOP fundamentals in Java, e.g. defining classes, using methods and libraries, etc.

3) Write a computer program to solve specific problems.

Some concepts in AI2 like event driven programming and concurrent execution require a lot of background in terms of OOP and threading. We believe that these topics are too advanced for middle school students and hence we do not cover them in our curriculum. Additionally, due to inherent complexity, we decided not to cover polymorphism or inheritance, with the hope that students will get exposed to those critical concepts in high school.

4. BLOCK 1: App Inventor Curriculum

Here we describe various modules of App Inventor curriculum. We divided the students in self-organizing teams (3-4 people) where each team had to develop the same mobile application through teamwork. The application closely mimics a student record management system, which would ask a user to enter each student's name, grade, DOB and gender and through the *textbox* component of AI2. Finally, the collected data will be used to display the names of all students, their grade and age, as of current date and their gender. The application will also maintain a List of students, which can be utilized to select and view particular student's information. To accelerate the implementation of this application, we created a design diagram beforehand. The design of the application was broken into 7 parts where each part's required concepts would align with 1 module of the course. The developed application's screenshot is shown in Figure 2.

4.1 Variables and Data Types Module

Our first module focused on delivering the concept of data types or variables in AI2. We presented the concept of variable creation and use of variables. Variables for the three AI2 data types: text (or string), number, and Boolean was also explained. Several variables were created, renamed, and

given initial values through a demo application in AI2. A more detailed explanation of Boolean operators and Boolean variables were also presented. The comparison operator blocks for both textual and numerical data were demonstrated. To incorporate experiential learning, here is how we designed the first module, which is followed in all modules of both courses.

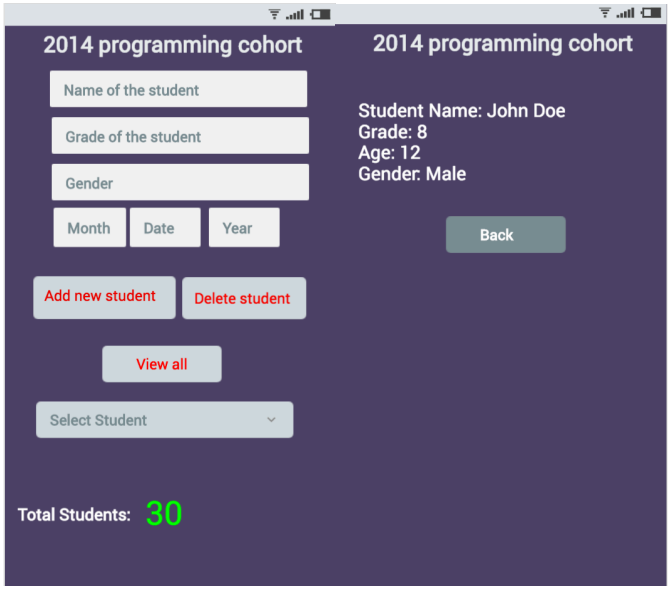

Figure 2. Screenshot of the mobile application developed in AI2

Concrete Experience: We first delivered a mini lecture on how variables store data and what is their purpose, which took approximately 15 min. It was followed by a demo of a widely used app named "Hello Purr" [18] having a button object and then programmed the button so when it is clicked a "meow" sound would play. Variable update was demonstrated with a label with a text on top it, displaying how many times the button was clicked. This demo took additional 15 min.

Reflective Observation: The students were then asked to repeat the steps of the Hello Purr app individually during next 30 min.

Abstract Conceptualization: This was followed by a 30 min team discussion on how many and what types of variables are needed to complete their project. Without advanced programming concepts (like loops or lists), students quickly realized that they would need fairly large number of variables since there were 30 students. Instructors helped each team during this time so they could form a collective correct solution.

Active Experimentation: The next 2 hours were dedicated to develop their solution in AI2 in teams. This allowed them to complete a small part of the project using variables to store student data (all hard coded in AI2). Students were encouraged to switch the role of driver and navigators [16] every 30 min.

4.2 I/O and Operators Module

The easiest way to introduce the notion of I/O in AI2 is through the use of textbox. The concept of this component was presented to the participants as the primary method of collecting numeric or textual information from the user. First 30 min of this module was spent on explaining what I/O is from the perspective of a program and demonstrating a textbox's two events, *GotFocus* and *LostFocus*. The following 30 min was spent on demonstrating step-by-step instructions how to use a textbox in AI2. This sample app asks the user about what course they took last semester and the corresponding grade, which needed to be entered through the textbox. The program would then

concatenate that information into a single sentence that stated their course name and grade using two labels to display the information. Students spent the next 30 min repeating the instructions for the sample program individually. The following 30 min was spent on team discussion to form a collective solution on what type of I/O functionality is required for their project and how they can program those. The rest of the in-class time was spent on project development within teams. Each team worked on asking the user to enter student information through textbox and using labels to display them. This allowed another feature of the project to be completed.

4.3 Conditionals Module

This module focused on introducing the notion of conditionals and branches in programs. The structure of both the *if/then-do* and the *if/then-do/else-d*o blocks, provided by the AI2 were explained. We also explained the role of Boolean variables used in the decision-making processes of a program. All these were explained in the first 30 min, which was followed by a demonstration of a sample program implementing decisions in AI2. The sample program simply sets different number (text) on a label based on a global variable value. We changed the global variable value to demonstrate how different if-branches are getting executed, which results in a different number displayed on the label. Like in our previous modules, the students first repeated the steps of this sample program, which was followed by a team discussion. Students spent the final 2.5 hours implementing the decision part of their project, where we asked them to display "Male" or "Female" on labels based on the gender variable, either M or F, respectively.

4.4 Repetitions Module

This module focused on iterations or loops in AI2. Three type of looping control blocks were explained using the *built-in* Control Blocks of AI2. The structure and the parameters used with the *for range* loop, *while do* loop, and "infinite while loop" were presented and explained. The *for range* loop block's *start*, *end*, and *step* parameters were explained. This was followed by the reflective observation stage with a demonstration of a simple application utilizing the concepts of loops. Due to time limitation, in this class, students were asked to repeat the steps after the instructor during the application demonstration. The demo app simply allowed the students to practice 3 types of loops where one ball (red color) bounced continuously till the user hits a key, another ball (blue color) bounces for a specific number of times entered by the user and the last ball (green color) would keep bouncing forever. During abstract conceptualization stage, each team discussed how they could utilize repetition in their project to use the same textboxes to take input of multiple students' information . Majority of the team struggled during this phase that was expected and the instructor tried to help them in constructing their solution, which required little over 2.5 hours, in teams, for them to develop. Some teams needed help during this phase to debug unexpected repetition behaviors encountered due to inaccurate logics and bugs.

4.5 Procedures Module

The goal of this module was to teach students about procedures in AI2 and their functionality. We explained the two types of procedure blocks in AI2 - one that returned a value and one that did not. It was shown that a procedure may or may not include arguments. During practice time, the demo used by the students had an arithmetic equation to calculate the area and volume of a 3D circle, whose radius was entered as a user input through textbox. This was followed by a team discussion on what procedures might be useful for their project, which in their case was computing the age from the entered DOB for each student. The students needed little over 2 hours, in teams, to incorporate this procedure in their project.

4.6 Lists Module

This module focused on the built-in *List* functionality of AI2. Through a small demo application, we first explained how to make a list, select list item, insert an item, replace item, append item, remove item and find out the length of list. The sample program was designed on top of a tutorial provided in the App Inventor web site that created a rudimentary "Quiz" program where the questions and answers were "hard coded" into the app. This was done by using a list for the questions and another list for the answers. After team discussion, the students incorporated the knowledge learned in this module in their project. This allowed them to create 4 List corresponding to the names, age, gender and grade of all students present in the class. This allowed students to complete their project, except few UI issues which students fixed in the following class.

4.7 Objects Module

Our last module focused on delivering the concepts of objects through AI2 demo application. We explained that everything they could drag from the palette to the viewer was an object (e.g. buttons, labels, textboxes, puzzle pieces and etc.). We also explained that object has properties which tells information about the objects, provides values and settings that describes the specific instance of the object. We reminded the notion of procedures and explained that some objects have procedures too, which allows them to perform certain action, like a button object have *Click*, *GotFocus*, *LostFocus*, etc. Even though students seemed to understand the concepts of object, it was hard for them to internalize why object's concept was necessary for them to learn app development. In this module, we asked the students to reflect on what types of objects they have used so far for their project. The rest of the class time was dedicated for team work where they created a table of objects used in their project, corresponding properties of each of those objects and procedures of each object and their functionality.

5. BLOCK 2: OOP (Java) Curriculum

Here, we present the concepts of OOP covered through Java and the links established with AI2 curriculum. For this module, we asked students to use DrJava IDE [19] to write their code. We exercised all stages of experiential learning during each module of Block 2 course, using the same approach we followed in Block 1. We divided the students in same teams (formed during Block 1) where each team had to develop the same application, however, this time using Java.

5.1 Variables and Data Types Module

Since students were mostly familiar with majority of the concepts of this module, the only new part was to learn the syntax difference in Java. We also demonstrated all these concepts through a simple Java program and asked the students to repeat after us. After team discussion, students started to use String variables to store names (by hard coding) of each student in their project. Almost all students were facing difficulty in this first module with syntax errors, which was expected. Instructors used this opportunity to teach them about various compile time and run-time errors, how to distinguish them, how to fix some common errors.

5.2 I/O, Operators and Libraries Module

This module demonstrated the students how to take input using *Scanner* and how to print using *System.out.print* function. We also briefly explained what are *libraries* in Java and how a programmer can benefit from using them. We re-introduced the notion of operators from Java's perspective, how to use them in the program and specially the purpose of *Mod* operator. A simple application on finding *Even/Odd* number using Mod operator was demonstrated to students. After

individual practice and team discussion, students started to write code in teams, so they could take input of students' information and store them in the String variable. They quickly realized that this was very inefficient which was later addressed in repetitions module.

5.3 Decisions Module

Transitioning to if-else structure in Java was pretty easy for majority of the students since they were familiar with similar concepts in AI2. Students also learned how to construct a nested if-else; using if statement to check more than one condition. We demonstrated and asked students to repeat after us, to write a similar program, where students used a integer variables (say 5) to display if the number entered was less then 0, grater than 25 or between 0 and 25. Later students integrated the concept of decisions in their project, within teams, to display if the gender of the student is Male or Female.

5.4 Repetitions and Nesting Module

A for loop in Java can be directly translated from a loop in AI2. To demonstrate the *for loop* and *do-while* loop in Java, we demonstrated a similar program used in Block 1 repetition module. The demo application asked the students to display a certain character for fixed number of times using for loop and while loop. We also explained *infinite while loop* and students displayed a character using *while(true)*, to see the outcome of an infinite loop. Later, students modified their project's code in teams to use same variables for all students to collect and display data using for loop, running the index from 0 to 29. However, students still did not learn how to store the data for all students permanently, which was addressed in a later module.

5.5 Methods and Parameters Module

Students with App Inventor background already experienced defining and calling functions. To explain the concept of value returning and void function, we demonstrated the same program used in Block 1 Procedure module. This time the demo program had two methods, both accepting the radius of a 3D circle as input (parameter). The students practiced writing code for *areaCalculate* method that directly printed the circle area and *volumeCalculate* method that returned the volume of the 3D circle back to main for printing. Later students worked in teams working on their project to implement methods that can compute their age, given their DOB (as month, day and year).

5.6 Arrays Module

Transitioning to the concept of Arrays in Java was much easier since students already had notion of lists from AI2. First a small demo application was shown to students that would use an array to store 10 numbers. Later, students worked in teams to modify their project so students' information could be saved in 4 parallel arrays of names, grade, gender and age.

5.7 Objects Module

Even though students had some experience with objects in AI2, transitioning to objects in Java seemed to be a critical concept for them. To ease their transition process, we first demonstrated a class named Student with the following instance variables *(String name, int grade)*. Later, we used this class to modify one the project codebase so the program would now use an array of Student object to store each student's information. We used a for loop to ask for each student's information from the user, instantiated a student object with the entered data and stored the new student object in the array. Later we used another loop to iterate through the array and display required student information. Since we expected that students would find these concepts complicated, we decided to demonstrate this whole process. Finally, we asked students to work in

teams to modify the student class to include other instance variables like DOB (divided in Mon, Day, Year) and gender. They also modified their project so it can now store students complete information through student Object in array and display them as required.

6. Evaluation

Our curriculum was offered to 30 students as an experiment to determine if visual programming could really help middle school students' transition to textual object-oriented language successfully. The primary goal of the course was to teach students programming fundamentals through experiential learning so they are better at internalizing various structures of programming.

For evaluation, we conducted a formal assessment at the end of the both courses to determine participants' knowledge of various programming structures. Pre and Post Surveys were also given for self-assessment of students' background in programming and their attitudes towards computer science. The details analysis of the survey results and assessment are out of the scope of the paper. However, we briefly highlight some interesting findings below:

The survey indicates that 88% of students rated their before course AI2 expertise as 'None', 6% rated their expertise 'Little', 6% rated it 'Some', and no students considered themselves an 'Expert'. No students rated their post-course expertise as 'None' or 'Little', 42% rated it 'Some', and 58% considered themselves an 'Expert'. 93% of the students rated their before course Java expertise as 'None' and 7% rated their expertise as 'little'.

Another set of Pre and Post Surveys were also given to account for students' attitudes towards taking computing courses in future, future interest in computer careers, and self-efficacy with regards to programming. Some of the key questions addressed in this survey include – for under-represented middle school students, can the approach applied in this course:

1. impact the choices regarding computing-related course work in the future?
2. alter perspectives on computing career choices?
3. enhance self-efficacy in programming?
4. provide better learning outcomes in programming?

For our survey, a 4-point Likert scale (Strongly Agree, Agree, Disagree, Strongly Disagree) was used. We calculated mean and standard deviation from the Likert items to produce a numeric value for each of the questions mentioned above, in both the pre- and post- survey. A partial summary of the results is presented in Table 1.

Table 1. Results (and standard deviations) from the pre- and post-survey

Topic	Pre (sd)	Post (sd)
Computing course interest	1.82 (0.61)	3.88 (0.52)
Computing career interest	1.90 (0.80)	3.75 (0.54)
Self-efficacy in programming	2.11 (0.90)	3.84 (0.70)
Programming Knowledge	1.25 (0.89)	3.48 (0.91)

It is worth stating that none of the findings in this survey should be casually generalized to other contexts. Also, the lack of responses to the post-survey makes it difficult to draw strong conclusions from it. It is likely that some students did not take the surveys seriously, but this is difficult to verify. Additionally, the survey questions, while typical of validated instruments, were not themselves validated. They were also kept short to avoid survey fatigue. This may explain why we observed contradictory responses in both in pre- and post- survey for "Self-efficacy" and "Programming knowledge". Finally, the offering of the pre-survey at the end of Block-1 course

and post-survey at the end of Block-2 course may also have impacted the study in unforeseen ways. Some students may have associated their post AI2 knowledge with OOP knowledge, which made them to select higher Likert items, contributing to the large standard deviation.

Qualitatively, the teacher observed a big difference in student's attitude towards programming after Block 1 course. Students seems more motivated and excited to learn more about programming, in this case, Java. However, few still expressed that they prefer AI2 compared to Java due its cool aspect of deploying a functional app in mobile with much less effort.

Based on instructor's observation of the students, few concepts like repetitions and objects were hard to understand. Students' energy toward learning Java was not always positive, many found it confusing, and as a result they were less motivated to finish their project. However, some aspects were positive, as the students understood the programming concept, and what is code, how to execute it, and other basic programming concepts. In open-ended questions, almost 60% of students reported that they understood most of the Java programming fundamentals since it was easy to relate the topics to AI2.

When asked, "Did any and/or both of these courses change the classes you are likely to take in the future?" the young learners answered overwhelmingly in the affirmative and some of their feedbacks were:

"The App Inventor helped me feel more confident towards computer programming, which made me intimidated before"

"I did few lessons in code.org and now that I had this class I feel like I will definitely take more classes in the future"

"I liked working in teams to create the mobile application. I am thinking to make a game that will shoot stars into the sky repeatedly"

"I could have never done it (make a mobile app) all my myself, but my teammates helped me figure out so many things and I would love to take more courses like this in future"

7. CONCLUSIONS AND DISCUSSION

In this paper, we shared our experience of designing and delivering two block courses on programming for middle school students. One unique aspect of our courses was we integrated experiential learning strategy by carefully crafting how the materials were presented, how students reflected on their knowledge, how they conceptualized through team discussion and finally used their knowledge by applying it in a new domain. These learning stages helped students to reach all 5 levels of Bloom's Taxonomy, which helps to be a competent learner.

Our qualitative findings show that students with AI2 background found programming in OOP, in this case Java, easier and more enjoyable. We also found that using similar examples in both App Inventor and Java effectively helped students perform better at transferring concepts. Additionally, our findings clearly points out that students found experiential learning (through team discussion, team work, demo) to be very effective. Finally we found that the "cool" aspect of developing a project (mobile application) made it a motivating and an enjoyable experience for students to learn fundamental programming concepts of OOP.

By analyzing the survey results and observing participants, we found that hands-on activities with minimal lecturing helped to speed the learning process. In future, we plan to offer these as regular courses in affiliation with local schools where a K-12 teacher can take the role of the instructor. Finally, one week for each course did not seem to be sufficient to expose participants to many

important concepts of either AI2 or OOP. Hence, running the two course over an entire semester or two consecutive semesters might allow instructors to cover more programming concepts which are required to solve problems in the real world.

8. REFERENCES

[1]. S. Kurkovsky, "Making computing attractive for non-majors: a course design," *Journal of Computer. Science Coll.* Vol.22, No. 3, pp. 90-97, Jan 2007.

[2]. A. Forte and M. Guzdial, M"otivation and nonmajors in computer science: identifying discrete audiences for introductory courses," *IEEE Transaction on Education*, Vol. 48, Ed. 2, pp. 248-253, 2005.

[3]. M, Guzdial and E. Soloway, "Teaching the Nintendo generation to program," *ACM Communication, Vol.* 45, Ed. 4, pp. 17-21, Apr 2002.

[4]. Computational Thinking through Mobile Computing. URL: https://nsfmobilect.wordpress.com/ [Last accessed: 17th March, 2018].

[5]. B. Bloom, M. Englehart, E. Furst, W. Hill, and D. Krathwohl, "Taxonomy of educational objectives: The classification of educational goals". *Handbook I: Cognitive domain*, NY, 1956.

[6]. A. Y. Kolb, and D. A. Kolb, "The learning way: Meta-cognitive aspects of experiential learning." *Simulation & Gaming,* Vol. 40, Ed. 3, pp. 297-327, 2009.

[7]. B. Kaucic, and T. Asic, "Improving introductory programming with Scratch?, " in *Proceedings of the 34th International Convention MIPRO*, 2011, pp. 1095-1100.

[8]. T. Booth, and S. Stumpf, "End-user experiences of visual and textual programming environments for Arduino," In *End- User Development*. Berlin Heidelberg: Springer, 2013, pp. 25-39.

[9]. M. J. Conway, "*Alice: Easy-to-learn 3D scripting for novices," School of Engineering and Applied Science, University of Virginia,* Charlottesville, VA. 1997.

[10]. Y. B. Kafai, and Q. Burke, "Computer programming goes back to school." *Edu. Week*, pp. 61-65.

[11]. N. Tabet, H. Gedawy, H. Alshikhabobakr, and S. Razak, "From Alice to Python. Introducing Text-based Programming in Middle Schools," in *Proceedings of the Annual Conference on Innovation and Technology in Computer Science Education* (ITiCSE '16), 2016, pp. 124-129.

[12]. Alice. Carnegie Melon University. URL: http://www.alice.org. [Last accessed: 17th March, 2018]

[13]. Android App Inventor. URL: http://appinventor.mit.edu [Last accessed: 17th March, 2018]

[14]. Scratch. URL: https://scratch.mit.edu/ [Last accessed: 17th March, 2018]

[15]. A. Wagner, J. Gray, J. Corley, and D. Wolber, "Using app inventor in a K-12 summer camp," in *Proceedings of Technical Symposium on Computer Science Education* (SIGCSE '13), 2013, pp. 521- 626.

[16]. N.Nagappan, L. Williams, M. Ferzli, E. Wiebe, K.Yang, C. Miller, and S. Balik, "Improving the CS1 experience with pair programming,"in *Proceedings of the ACM Technical Symposium on Computer Science Education (SIGCSE '03)*, 2003, pp. 359-362.

[17]. M. Guzdial, "Programming environments for novices," in *S. Fincher and M. Petre (Eds.), Computer Science Education Research. London,* UK: Taylor & Francis, pp. 127-154.

[18]. HelloPur, URL: http://appinventor.mit.edu/explore/content/hellopurr.html [Last accessed: 17th March, 2018].

[19]. DrJava. URL: http://drjava.sourceforge.net/ [Last accessed: 17th March, 2018].

[20]. M. Kölling,"The problem of teaching object-oriented programming, Part 1: Languages." *Journal of Object-oriented programming*, Vol. 11, Ed. 8, 1999, pp. 8-15.

[21]. B. Moskal, D. Lurie, and S. Cooper, "Evaluating the effectiveness of a new instructional approach," in *Proceedings of the* SIGCSE Bulletin, Vol. 36, Ed. 1, 2004, pp. 75–79.

[22]. Code.org. URL: http://code.org. [Last accessed: 17th March, 2018].

[23]. College Board. CS principles. http://csprinciples.org. [Last accessed: 17th March, 2018].

[24]. Computer Science Unplugged. http://csunplugged.org. [Last accessed: 17th March, 2018].

[25]. J. Cuny. Finding 10,000 teachers. *In CSTA Voice*, 5, 6, pages 1–2, Jan 2010.

[26]. E. Freudenthal, M. Roy, A. Ogrey, T. Magoc, and A. Siegel, "Mpct: Media propelled computational thinking," in *The Forty-first SIGCSE Technical Symposium on Computer Science Education*, 2010, pp. 37–41.