Using Mobile Phone Programming to Teach Java and Advanced Programming to Computer Scientists

Derek Riley
University of Wisconsin-Parkside
Kenosha, WI
rileyd@uwp.edu

ABSTRACT

In this work an approach employing the Android mobile phone platform in an upper division computer science course to teach Java programming and other advanced computer science topics is presented. Mobile phones are growing in influence in the computing market, but their strengths and popularity are rarely exploited in computer science classrooms. The aim of the course is to harness this enthusiasm to improve fluency in the Java language to afford an opportunity to learn how to work on large, complex projects and to enhance the students’ preparedness for the job market. The ideas presented in this work could be adapted for improving learning in many courses across the computing curriculum.

Categories and Subject Descriptors: K.3.2 [Computer and Information Science Education]: Curriculum, Computer science education

General Terms: Human Factors, Design

Keywords: Java, Android, Education, Course.

1. INTRODUCTION

The recent revolution in mobile computing has taken computing out of the office and into pockets around the world, and the market segment is projected to continue growth [9]. Having grown up using mobile phones, students find the transition to smart phones to be natural. They enthusiastically master smart phone user interfaces and applications, so it seems that students with an interest in programming will be highly motivated to learn how to program smart phones. Harnessing this enthusiasm in upper-division computer science courses can potentially improve the learning of core computing topics as well as improve the preparedness of students for the job market. In this work I present a course designed to teach the Java language and object oriented programming to upperclassmen using the Android platform.

There are several platforms in the mobile device community. The Android platform was chosen because it is open source, it has a large community of active developers, and its development can be done using any operating system, therefore eliminating the need to buy Apple computers (as is required for iOS development) [6]. Android is especially well-suited for use in the computer science classroom because it utilizes many features of the Java programming language and provides a plug-in for integration with Eclipse [2] including an emulation environment called the Android Virtual Device (AVD), which eliminates the need to have the actual mobile devices to do basic development [1]. However, many students already own and use Android-based mobile devices, so they can be easily used if students desire to pursue programming of more advanced features not implemented in the AVD.

The goal of the course presented in this paper is to use Android as a platform to teach upperclassmen computer science students the Java language. The students in CSCI 3033 at Middle Tennessee State University (MTSU) numbered twenty-eight: twenty-six were computer science majors and two were math majors. Many only had used C/C++ previously, and this course was their first exposure to learning a new programming language.

Android has been used previously as a platform in mobile device courses [7, 5, 10], but given the newness of the platform, no textbook currently exists to teach Java or any other upper-division courses using Android examples. Further, most Java language textbooks are intended for CS1/CS2 courses, so they are not easily adapted for an upper-division course. The concepts presented in this paper could be adapted for use in other upper level CS courses such as networking, event-driven programming, parallel programming, game design, graphics, databases, and more.

Writing simple applications from scratch using the Android platform requires proficiency in OO concepts, event-driven programming, and the Android architecture, so the learning curve for Android is steep enough to keep most students from learning it on their own. Further, only about two-thirds of the students in the MTSU course had taken upper-level courses such as operating systems, networking, or algorithms, so many topics were covered in the course in addition to the Android material to enable the students to utilize some of the advanced Android features they chose to explore while learning Java.

The contributions of this work are twofold. First, I present an extensible course to bridge the gap between C++ and Java using Android while demonstrating the course’s efficacy. The course is designed to overcome the aforemen-
tioned challenges by covering a breadth of computing topics to soften the Android learning curve and cater to a wide variety of computing backgrounds. The course also attempts to use relatable and realistic projects to motivate students without causing them to be overwhelmed by the challenges of learning a new language.

The second contribution is a proposed curriculum consisting of targeted individual and group projects designed to teach a variety of programming concepts for Android. Several examples of student group projects are presented to demonstrate student capabilities at this level. A post-course survey shows that students view the instructional approach favorably with regard to learning Java programming skills. The survey also indicates that students feel more prepared for what they are likely to encounter in their future careers as a result of this course.

2. THE COURSE

This section describes the goals and design of CSCI 3033. The course is is typically taken by upperclassmen enrollees who need to fulfill a language breadth requirement, but it is not required for any academic major or minor program of study. The C++ language is used as the primary language for most undergraduate Computer Science courses at MTSU; the only prerequisites for this course are CS1 and CS2.

2.1 Goals

The main goal of CSCI 3033 is to teach students how to become competent Java developers, which includes proficiency in OO programming topics, the use of libraries, and much more. Because of the potentially varied background of the students in the course, it was necessary to cover a wide variety of topics from the basics of the Java programming environment to more advanced topics such as multithreading and event-driven programming. Another pedagogical priority was success for all students regardless of their programming background, together with student exposure to large, real-world software challenges.

Key CSCI 3033 topics reinforced with programming assignments or examples included inheritance, polymorphism, interfaces, exception handling, streams, files, and collections. Additionally, it was expected that students would acquire a basic understanding of GUI programming and the core Android API. More challenging topics such as multithreading, event-driven programming, networking, XML, and graphics were incorporated as time permitted. Lastly, it was important for best software engineering practices to be a common thread of all course presentations and assignments.

Assessment of the course goals was accomplished by a midterm exam, several individual programming assignments, a final, cumulative project, and a post-semester survey. I posit that these tools were adequate for assessing the individual student performance.

2.2 Course Design

The course was split into five units according to the aforementioned goals: Java introduction, OO programming, Android introduction, advanced Android, and software engineering. Because of the variation in student background, special care was taken to select examples that were motivating, realistic, and allowed the students some creative freedom.

Unit 1 (One Week) The first unit was an introduction to the basic syntax of Java. Throughout this unit command-line programming was used to help teach the basics of the Java programming environment without the use of an IDE. Unit 1 focused on the differences between Java and C++, and programming assignments were designed to introduce students to the compilation and running environment. By the end of this unit students were able to write, compile, and run Java applications consisting of a couple of simple classes.

Unit 2 (Three Weeks) The second course unit explored object-oriented programming (OOP) with inheritance, polymorphism, libraries, and packages, presented in one or two class periods each. Most students were previously familiar with OOP, but a few non-traditional or transfer students were not. OOP concepts are reinforced through many in class examples, individual programming assignments, and homework problems.

The concepts of Java interfaces, exception handling, files, streams, collections, and the Java API and class libraries were also included in Unit 2. The non-Android components of the Trainster programming assignment (described in Section 3.1) were presented using only command-line interactions. Other topics explored in Unit 2 include the Eclipse IDE, Javadoc documentation, UML class diagrams, and UML object diagrams. A midterm examination was administered after this unit to access student knowledge of the Java basics.

Unit 3 (Two Weeks) The third unit introduced the Android operating system and programming environment. Students began by participating in a lab to create an Android tip calculator application using a single class and simple user interface. Most of the code for this project was provided by the instructor. Concepts of event-driven programming, operating systems, and GUI objects and layouts are discussed using this and other Android examples.

The Trainster code developed in Unit 2 was re-implemented for the Android environment with some added functionality and GUI features as described in Section 3.1. After students successfully implemented their individual Trainster projects, they formed groups to brainstorm ideas for the their final project.

Unit 4 (One Week) This next unit includes lectures on more advanced Android and Java concepts such as threads, intents (interprocess communication), libraries, sensors, GPS, and more. The students proposed, planned, and began the development of their final projects during this time (as described in Section 3.2), so lectures were tailored to the topics required for the wide variety of proposed projects.

Unit 5 (Seven Weeks) As the students developed their final projects, lectures concentrated on the concepts of debugging, design patterns, testing, project management, and effective presentations. Time was also devoted to examining software documentation, external libraries, intellectual property, open source licenses, and other topics. Students were required to give project update presentations, and the class was encouraged to share code and suggest improvements to each others’ projects. The Android platform introduced many real-world programming examples that required students to put software engineering theory into practice and work as a collaborative development group.
3. ANDROID PROJECTS

Two major projects were used in CSCI 3033: one individual project (Trainster) and one group project. The individual Trainster project required the students to develop an application that stores and calculates Body Mass Index (BMI) data for use by a personal trainer. The group projects were unique with each group required to propose, design, and create their own Android application.

3.1 Trainster Project

The Trainster project consisted of a series of smaller, incremental projects designed to teach basic Java syntax; these projects were completed in the first half of the semester. Upon completion, Trainster is a software application that a personal trainer (at a gym) could use for storing information about clients. The specific learning outcomes of Trainster related to the topics of inheritance, polymorphism, exception handling, and basic Android application programming. The project was divided into three phases to correspond to the concepts discussed in lectures.

**Phase 1: Simple BMI** Students wrote an application to calculate a person’s BMI given a height (in inches) and weight (in pounds) as shown in (1). This phase of the project only required command-line programming, to ensure student focus on Java syntax and good programming techniques. The result of the BMI calculation is a floating-point value that crudely indicates the status of an individual’s health and is categorized using Table 3.1 [4]. Students were required to create an object to contain the client’s name, height, weight, BMI, and category.

\[
BMI_E = \frac{703 \times \text{weight}}{\text{height}^2}
\]  

(1)

<table>
<thead>
<tr>
<th>BMI</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;18.5</td>
<td>Underweight</td>
</tr>
<tr>
<td>18.5-25</td>
<td>Normal</td>
</tr>
<tr>
<td>25-30</td>
<td>Overweight</td>
</tr>
<tr>
<td>&gt;30</td>
<td>Obese</td>
</tr>
</tbody>
</table>

Table 1: BMI categorizations

**Phase 2: OO Extension** For this phase of the project, students used inheritance and polymorphism, permitting application users to enter, calculate, and save clients’ BMI data with both English and metric units. The BMI calculation for metric units is similar (2) to the English calculation, so the students were required to create an abstract ‘Body’ class as well as ‘English’ and ‘Metric’ subclasses that inherit from the ‘Body’ superclass and overload a calculate() method.

Phase 2 also required sorting a list of clients in order to expose students to Java data structures. During this phase of the project students were also introduced to the Eclipse IDE [2]. The OO phase of the project also only required command-line programming to allow students to focus on the OO concepts.

\[
BMI_M = \frac{\text{weight}}{\text{height}^2}
\]  

(2)

**Phase 3: Android Application** In the final phase of the project the students were required to create an Android application that allowed users to enter multiple clients, including name, height, weight, and units (English or metric). Students used the Java classes developed in Phase 2 and were required to use exception handling to ensure robustness. Students were encouraged (but not required) to serialize their objects to save state in the event that the application was interrupted.

Phase 3 also introduced students to Android GUI programming concepts, using EditText and TextView objects as well as buttons, XML layouts, and views. Students were required to create a main screen to allow the user to enter data as well as a second screen to allow users to view the clients. This satisfied learning outcomes related to the concepts of emulators, event-driven programming, Activities (threads), and file systems. A screen shot of an example version of the application is shown in Figure 1. Students were required to design their own GUI using the XML layout editor included in the Eclipse ADT plugin.

![Figure 1: Personal trainer Android application](image)

3.2 Group Project

Few programming projects in the real world allow programmers to work exclusively on their own, so exposure to group projects is a critical part of any programmer’s education [8]. I was particularly interested in developing team programming skills while students learned Java to integrate this experience with OO programming practices.

Near the midpoint of the semester, students presented proposed ideas for Android-based applications to their peers. During the proposal presentations, the class was encouraged to brainstorm, suggest improvements, and highlight challenges in each others’ ideas. The students were given time in class to meet with each other and form groups (on their own). One week after forming teams students were required to present the details of their proposed projects and generate initial requirements documents including preliminary UML diagrams. The initial presentations were intended to allow students to ‘dream big’ while the second presentations were
intended to force the students to impose realistic goals and distill plans into a practical form from their initial ideas.

The requirements for the project were intentionally open to encourage creativity. Guidelines required that all projects demonstrated Java fluency and incorporated at least one of the following topics: networking, multithreading, databases, sensors (GPS, accelerometer, etc.), or advanced graphics.

Two more presentations were required of the project teams: a mid project presentation and a final presentation. In both, students were required to present working prototypes, discuss challenges, and highlight obstacles. In the final presentation the students were also required to present their testing scheme, project documentation, and to summarize the experience. During all presentations, students were asked to critique and suggest improvements for others’ projects.

4. EXPERIENCE

A post-course survey (see Figure 2) was conducted to assess various aspects of the course from the student perspective. The survey was completed by 26 students (35% were third-year and 65% were fourth-year or beyond). The results of this survey are summarized in Section 4.4.

4.1 Trainster Project

The Trainster project is an effective project to introduce Java because it illustrates the utility of OO programming and code reuse using Java. Given the students’ varied programming backgrounds, forcing them to use OO concepts and start with command-line programming appeared to equalize student abilities. Enhancements for programming assignments, such as using more efficient data structures, sorting, and file manipulation, were encouraged (but not required) to challenge the more advanced programmers.

Grades on the early phases of Trainster were lower than the grades for the later phases of the project, most likely due to the differences in OO programming understanding and ability. The students who struggled with the basics of OO programming mostly caught up by the later phases of the project because they re-implemented what they had written earlier so it would utilize the OO concepts correctly and work for the later phases of the project.

Student survey responses demonstrated a favorable attitude toward code reuse and code refactoring. Other comments expressed pleasure with a project that allowed for some creative freedom. When asked to rate the effectiveness of the Trainster application in teaching Java concepts no student rated it this as ‘Poor’, 19% of students rated it ‘OK’, 54% rated it ‘Good’, and 27% rated it ‘Great’.

Student survey responses associated with Phase 3 of the Trainster project indicated a favorable attitude toward working on a project that had obvious real-world applications. When students were asked to rate the effectiveness of the Trainster application for teaching the Android basics, no students rated this ‘Poor’, 12% of students rated it ‘OK’, 54% rated it ‘Good’, and 34% rated it ‘Great’.

Students also remarked the project was easy to understand, grade expectations were clear, and it was nice to be able to take creative liberties with the assignment. The best projects were demonstrated in class. The students appeared enthusiastic during these demonstrations, creating a positive competitive environment.

4.2 Group Project

Team programming was an opportunity for students to use Java and apply OO design in a group setting. Students related well to mobile phone applications, perhaps because most use such applications regularly in daily life. To foster creative ideas for the team projects, one class period was used to brainstorm and discuss ideas as a class. Students were told to prepare for the class by talking to teammates to propose specific applications. The current popularity of Android applications together with student creativity generated more good ideas than there were teams to implement them. The variety of applications reflected the diversity of student interest.

A common characteristic of initially proposed projects was that they tended to be overly ambitious for the limited duration of the class, but this too can be an important lesson. Students were required to show how the proposed application was significantly different than existing applications in order to encourage students to research before writing code. Example projects are presented in Section 4.3.

Grades for the final projects were generally quite good (mostly ‘A’s or ‘B’s). A large part of the grade (60%) was for planning, proposing, and presenting the work, while the actual code represented the other 40%. The 60% weighting was intended to encourage students to hone communication skills and enhance the collaborative environment. All submitted projects were fully functional, with several exhibiting more functionality than initially proposed. Some groups did not implement every proposed requirement, but all projects demonstrated adequate proficiency with the Java language and Android platform.

Students indicated in the post-course survey that they enjoyed having the opportunity to learn Java using the Android group project. The survey asked students to rate the effectiveness of the group project for teaching Java concepts. No students rated this ‘Poor’, 4% of students rated it ‘OK’, 23% rated it ‘Good’, and 73% rated it ‘Great’.

Students also mentioned that they enjoyed the ability to work in teams where the strengths of each student could be leveraged and they could choose a project of interest. They claimed to spend much more time on their projects than they have in previous CS courses, and felt that the time spent was enjoyable. Lastly, students indicated that they enjoyed the collaborative environment of the class and were surprised by how much they learned from each other through their presentations, discussions, and other in class interactions.
4.3 Example Projects

The team projects ranged from clever utility applications to games to entertainment applications. The vast majority of the projects had positive outcomes because the students were self-motivated and encouraged by each other to create useful, functional applications. Some of the best projects required hundreds of programming hours and several thousand lines of code. A few projects created by the students are explained briefly below.

Figure 3: Audio recording application

**Audio recorder** - This application allows users to record audio, encode it into MP3, and save notes in the audio using timestamps. The application also utilizes voice-to-text libraries, permitting users to name their recordings based on what is said in the recording. Figure 3 shows a screen shot of the recording screen of the application.

**Wiring Diagram Generator** - This utility application for car audio enthusiasts allows users to select a configuration of speakers, amplifier, and impedance to create a custom color-coded wiring diagram. The application also provides the user with other calculators to determine the wiring gauge necessary and capacitor required for a specific installation. Figure 4 shows a screen shot of an example wiring diagram.

**Tap the duck game** - This game randomly generates sprites (small duck images) that fly around the screen and entice the user to tap them, at which time they disappear. The sprites bounce around the screen quickly, but with varying speeds, and the program keeps track of how many sprites the user taps in a limited amount of time. The sprites also switch directions when reaching a boundary, and it supports multiple game layers.

This application is especially sophisticated because the students had to incorporate 2D graphics programming as well as design their own game engine using multithreading for performance reasons. A screen shot of the application can be seen in Figure 5.

**Wheelchair trainer** - This application assists wheelchair users in training for athletic events by keeping maintaining the number of pushes made and the distance traveled over time. The application uses the Android device accelerometer to track the force of pushes and the GPS to measure the distance traveled. Data is recorded so the user can compare training runs. A sensitivity adjustment was included to account for the manner in which different road surfaces and different wheel types affect accelerometer readings. A screen shot of this application can be seen in Figure 6.

4.4 Evaluation

The primary goal of the course was to teach students how to become competent Java developers. The post-course survey asked students to rate their Java expertise before and after the course. 69% of students rated their before course Java expertise as ‘None’, 23% rated their expertise ‘Little’, 8% rated it ‘Some’, and no students considered themselves an ‘Expert’. No students rated their post-course expertise as ‘None’ or ‘Little’, 46% rated it ‘Some’, and 54% considered themselves an ‘Expert’.

To assess the impact of Android students were asked to rate the effectiveness of using Android to teach Java concepts. No student rated this ‘Poor’, 12% of students rated it ‘OK’, 42% rated it ‘Good’, and 46% rated it ‘Great’. One student wrote a comment that he learned Java concepts using Android that he felt that he would not have learned in a traditional Java course.

One of the challenges of computer science education in the internet age is the availability of online solutions to common programming assignments. While some Android software is available online, most of the applications are closed-source, so students learn to use developer websites and forums to collaborate with other programmers instead of just copying them. Another advantage of using Android for projects is that students inherently incorporate creativity into their own projects making plagiarism obvious.

In the post-semester survey students were asked to rate their feeling of preparedness for the job market prior to and
5. CONCLUSIONS

This paper presents a course designed to teach C/C++ programmers how to develop OO Java software using the Android mobile operating system platform. The course is designed to accommodate students from a wide variety of programming backgrounds and to harness their creativity and enthusiasm for programming mobile phones in order to enhance learning outcomes. According to [3] approximately 30 percent of CS departments use C/C++ as the language for CS1/CS2; this suggests that approaches such as the one described could be applicable to such institutions. Further, Android can be used as a platform to teach other upper-division computing courses such as operating systems, networking, parallel programming, graphics, and others.

There are several challenges of incorporating Android into an established course. First, class time must be spent discussing the nuances of the operating system and visualization that are specific to Android, which takes away from the time for other topics. Another challenge of this approach is the ever-changing landscape of mobile operating systems. The Android plug-in and libraries are constantly changing, so the course and computer labs must be kept up to date to incorporate the changes. Another challenge was created by the so-called digital divide because some students were unfamiliar with mobile devices, and therefore, they did not feel motivated to use them in the course. Despite these challenges, this course has demonstrated that incorporation of Android programming into courses requires little investment beyond time, so I posit that Android is an effective platform for teaching Java and selected programming concepts.

6. REFERENCES