Journal of Computer Information Systems

ISSN: 0887-4417 (Print) 2380-2057 (Online) Journal homepage: www.tandfonline.com/journals/ucis20

Assessing Students' Object-Oriented Programming Skills with Java: The "Department-Employee" Project

Xihui Zhang, John D. Crabtree, Mark G. Terwilliger & Tyler T. Redman

To cite this article: Xihui Zhang, John D. Crabtree, Mark G. Terwilliger & Tyler T. Redman (2020) Assessing Students' Object-Oriented Programming Skills with Java: The "Department-Employee" Project, Journal of Computer Information Systems, 60:3, 274-286, DOI: 10.1080/08874417.2018.1467243

To link to this article: https://doi.org/10.1080/08874417.2018.1467243

JOURNAL OF COMPUTER INFORMATION SYSTEMS
2020, VOL. 60, NO. 3, 274–286
https://doi.org/10.1080/08874417.2018.1467243

Assessing Students' Object-Oriented Programming Skills with Java: The "Department-Employee" Project

Xihui Zhang, John D. Crabtree, Mark G. Terwilliger, and Tyler T. Redman

Computer Science & Information Systems, University of North Alabama, Florence, AL, USA

ABSTRACT

Java is arguably today's most popular and widely used object-oriented programming language. Learning Java is a daunting task for students, and teaching it is a challenging undertaking for instructors. To assess students' object-oriented programming skills with Java, we developed the "Department-Employee" project. In this article, we review the history of object-oriented programming, provide an overview of object-oriented programming with Java, and present a summary of existing Java projects and their limitations. We also provide the project specifications as well as the course background, grading rubric, and score reports. Survey data are presented on students' backgrounds, as well as students' perceptions regarding the project. Results from the instructor score reports, correlation of the project score and the final course score, and student perceptions show that the "Department-Employee" project is effective in assessing students' object-oriented programming skills with Java.

KEYWORDS

Structured programming; object-oriented programming; Java

Introduction

Java is an object-oriented programming (OOP) language. Compared to structured programming techniques, object-oriented design and programming provides a more natural and intuitive way to describe real-world objects by creating classes and their runtime objects (also called instances). People possessing strong OOP skills with Java are in high demand in industry. Nine out of today's 10 most popular programming languages support OOP.[1] Although there are over 100 programming languages in existence today, research has shown that language adoption follows a power law, and the top six languages account for 75% of the software projects at SourceForge.[2] As such, it is imperative that institutions of higher learning teach their students OOP skills so that the students are well prepared for their prospective careers in the field of information technology.

Learning an OOP language such as Java, especially for novices, can be a daunting task.[3] This is because they need to learn not only structured programming concepts and skills, which are used for the code written within Java methods, but also object-oriented concepts and skills, which are used for higher level abstractions. Furthermore, they also need to learn "how to use the Java Class Library to locate the details of classes, methods, and toolkits that they can use in their own classes"[3] (p. 365). Java is said to have a "fairly complicated syntax and fairly complicated static semantics"[4] (p. 19). Ramesh and Wu[5] have identified three main hurdles that students face: (a) clearly distinguishing between classes and objects, (b) understanding and using complex objects (objects that contain one or more objects within them), and (c) grasping the concepts of inheritance and interfaces.

Teaching an OOP language such as Java can be a challenging undertaking. A key question facing programming instructors is whether to take an objects-first approach or a structures-first approach to the OOP education. Primitive types in Java (i.e., boolean, char, byte, short, int, long, float, and double) have long been considered a weakness of Java because they violate the principle of orthogonality in programming language design.[6] The classic "hello, world" program was also considered harmful because OOP should be learned naturally, not procedurally.[7] The research results by Johnson and Moses[8] indicate that students who take an objects-first approach to OOP outperform those who take a structures-first approach. Hu[9] maintains that the debate between "objects early" and "objects later" is not that important, and just saying "a class defines a data type" would be adequate. Xing and Belkhouche[10] argue that critical object-oriented design should be given precedence over OOP techniques in teaching OOP. Furthermore, the use of a modern Integrated Development Environment (IDE) such as NetBeans, with active learning and a breadth-first approach, is found to increase student satisfaction, increase success rates, and lower dropout frequencies.[11]

In short, learning object-oriented Java programming is a formidable task for students, and teaching it is a challenging undertaking for instructors. To improve the effectiveness of both instructor teaching and student learning, it is imperative to develop tools to assess students' OOP skills with Java. In this research, we create a "Department-Employee" project to assist in this process. This project may be used in any Java course in which the proper concepts, as detailed below, have been covered. We are not advocating a particular course design or

CONTACT Xihui Zhang xzhang6@una.edu Computer Science & Information Systems, University of North Alabama, Florence, AL, 35632-0001, USA
Color versions of one or more of the figures in the article can be found online at www.tandfonline.com/ucis.

approach (e.g., objects-first) and, in fact, the authors of this article have different preferences. However, assessment of the students' grasp of object-oriented concepts is important in any Java course, regardless of the particular course design, and this assignment has been a useful assessment tool in our courses.

This article describes the "Department-Employee" project. It begins with a review of the history of OOP, an overview of OOP with Java, and a summary of existing Java projects and their limitations. The specifications of the project, the course background, the grading rubric and score reports, the correlation between the project score and the final course score, the students' perceptions about the project, and the students' backgrounds are also presented.

Object-oriented programming: a holistic view

A brief history of object-oriented programming

Smalltalk, the first true OOP language, has been around since the 1960s. However, it was not until the 1980s that hardware advances, especially in the area of random access memory, made OOP practical enough to gain mainstream acceptance in industry.

OOP has always been difficult to define, but the concept of the "object" is certainly central to any definition.[12,13] The focus on objects naturally led to object-oriented approaches to analysis and design. Many different approaches to problem decomposition in terms of objects and classes led to the "method wars" of the 1980s and 1990s. Many thought-leaders produced methodologies, along with their associated object notations and/or tools. Three authors who gained preeminence in the early 1990s published the following definitions:

- *Grady Booch (Rational Rose)*: OOP is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships[14] (p. 36).
- *James Rumbaugh (Object Modeling Technique)*: Superficially the term "object-oriented" means that we organize software as a collection of discrete objects that incorporate both data structure and behavior. This is in contrast to conventional programming in which data structure and behavior are only loosely connected. There is some dispute about exactly what characteristics are required by an object-oriented approach, but they generally include four aspects: identity, classification, polymorphism, and inheritance[15] (p. 1).
- *Ivar Jacobson (Objectory)*: In his chapter titled "What is object-orientation?", Ivar Jacobson states that his aim is "to introduce the actual idea [of object-oriented technology], not to give strict and precise definitions"[16] (p. 42). It is worth noting that in the 1990s Jacobson joined Booch and Rumbaugh in the collaboration that gave us the Unified Modeling Language (UML) and the Unified Process.

While properly defining objects and their precise characteristics may be difficult, the standard premise of OOP has always been that it should be simpler to work with objects since "people already think in terms of objects"[17] (p. 6). However, as Bruce Eckel points out, there are hurdles that the typical programmer must deal with: "If there's a downside, it is the expense of the learning curve. Thinking in objects is a dramatic departure from thinking procedurally, and the process of *designing* objects is much more challenging than procedural design, especially if you're trying to create reusable objects"[18] (p. 25).

In the 1980s, the new C++ language, created by Bjarne Stroustrup, gained traction and remains one of the most popular OOP languages today.[1] This new language, which leveraged the existing robust support for the very popular C language, was originally named C with Classes.

While C++ was certainly popular in the early 1990s (and still is), it has its problems.[1,2] The language was not formally standardized until 1998 as ISO/IEC 14882:1998. While the release of *The Annotated C++ Reference Manual* in 1990 served as a de-facto language standard, all compilers do not follow the same language rules and consequently porting code from one compiler to another is often difficult. In addition, C++ code compiled for a particular hardware architecture and/or runtime environment cannot be used in a different environment without recompilation from the source code (which may involve using a different compiler).

These and other issues inspired James Gosling at Sun Microsystems to create a language, originally known as Oak, that was object-oriented, standard, and portable. The first releases of the rebranded Java programming language and its Java Development Kit (JDK) were made available to the public in 1995. According to Benander et al.,[19] the best feature of Java is its "platform independency."

Object-oriented programming with Java

Since it did not need to rely upon an existing language for support (e.g., C), Java was designed from the ground up to be object-oriented. All Java programs are object-oriented. On the other hand, with a language like C++, it is possible for programmers to fall back into a more procedural, C-like programming style. While not all Java programs are guaranteed to have *good* object-oriented design, all Java code must be written in the context of a class and, more specifically, within a method belonging to a class. From a software engineering perspective, object-oriented design is often used in the architectural design, while structured design and programming is used for detailed method implementation.[20] Virtually all of the executable code in a Java class is written within methods, and that code must be structured code.

However, not all variables in a Java program are objects. Performance concerns led the creator of Java to provide "primitive" data types that were not classes. These eight primitives are: boolean, char, byte, short, int, long, float, and double. Apart from these eight data types, all other variables in a Java program must hold objects—or, more precisely, object references.

Objects are created or *instantiated* from classes. This concept is central to all OOP languages. The object represents a particular instance of the class. The state of this object is defined by the collective values of its internal data.

A key point for programmers new to OOP is the fact that the state of the object is *encapsulated* within that object. The programmer should not necessarily know the details of the internal state of the object or the internal behavior, which is defined by the code inside the methods in the class from which the object came. The programmer need only know the *interface* for this object.

The Application Programming Interface (API) for an object is usually composed of the methods from the object's class that have been marked as public. On rare occasions, a class designer may choose to mark a field as public, but this should be avoided—especially if the field is not a constant (i.e., final).

As previously mentioned, good class design is much more difficult than procedural (structured) program design. Therefore, students who have little or no experience with OOP should be provided with well-designed classes which employ solid, object-oriented design principles to use in creating an application-level program. In other words, it is our belief that students (especially those who have a background in structured programming techniques) should first be tasked with writing code that uses *existing* classes which make use of good encapsulation principles. Students can then experience how they should focus on the API and write their code to the interface and not to the implementation.

This is extremely important and is the central difference between object-oriented thinking and thinking in terms of procedures. The existing code that is made available to the students could be provided by the instructor, or it could come from the Java API. Even if you limit your students, as we do, to the java.lang and java.util packages, there are plenty of classes that can be used by the students in order to understand how to use the Java Class Library (via the Java API) just as effectively as code from the instructor.

Another important point to stress is that while the student may read the code in the classes provided by the instructor (hopefully to see an example of well-written code), this is not necessary in order to use the code. The code in the Java library is revealed by the Java API—not source code. Well-designed classes are like black boxes. The students' only concern should be the API that provides the interface to the black box.

Once the students have gained confidence in the use of objects instantiated from existing classes provided by the Java Class Library and/or the instructor, they can then be tasked with the implementation of classes (given the design for those classes). This project represents one possible exercise that can help students make the move from procedural to object-oriented thinking.

Existing Java projects and their limitations

All Java textbooks (e.g.,[21,22]) offer OOP projects. However, these projects are typically designed for students to practice with certain programming concepts that are tied closely to certain chapters. For instance, in Liang's[22] latest textbook edition titled "Introduction to Java Programming and Data Structures, Comprehensive Version," there are five chapters pertaining to OOP. These chapters include: (1) Chapter 9: Objects and Classes; (2) Chapter 10: Object-Oriented Thinking; (3) Chapter 11: Inheritance and Polymorphism; (4) Chapter 12: Exception Handling and Text I/O; and (5) Chapter 13: Abstract Classes and Interfaces. At the end of each of these five chapters, many programming projects are provided. However, none of the projects are comprehensive enough to require students to understand all of the major concepts and apply all of the major techniques related to OOP with Java.

Similarly, there are many Java teaching case studies published in numerous academic journals. These journals include the *Journal of Information Systems Education, Journal of Information Technology Education, Communications of ACM, Decision Sciences Journal of Innovative Education, Journal of Computer Information Systems*, and others. Most of the projects presented in these journal articles are narrow in focus and myopic in scope. For instance, Cavaiani (2006) addressed OOP principles and the Java class library; Pendergast (2006) focused on Java problems and pitfalls; Xing and Belkhouche (2003) believed that pseudo OOP is harmful; Ramesh and Wu (2004) described an approach to teaching OOP concepts in business schools; and Benander et al. (2003) presented the perceptions of Java from the perspective of experienced programmers.

Given the limitations of the existing projects in both OOP Java textbooks and Java teaching case studies published at academic journals, it is important to develop a project that will be comprehensive in scope, flexible in delivery, and holistic in process. That was the motivation for the design and implementation of the "Department-Employee" project.

The "Department-Employee" project

Project background

The "Department-Employee" project was created to assess students' OOP skills with Java. This project asks students to implement a system that uses important OOP concepts and techniques. This includes classes and objects, aggregating classes and aggregated classes, encapsulation and information hiding, superclasses and subclasses, inheritance, method overriding and overloading, abstract classes and methods, polymorphism and dynamic binding. Specifically, this project asks students to implement four classes: Department, Employee, Consultant, and SalariedEmployee. There is an aggregation relationship between the Department class and the Employee class where Employee is the aggregating class and Department is the aggregated class. This indicates that an employee belongs to a department and that the department object is modeled as a field within the employee object. Both Consultant and SalariedEmployee are subclasses of the Employee base class. The UML class diagram for the project is given in Figure 1.

Project instructions

For this Department-Employee project, students are provided four files: DepartmentEmployeeProjectInstructions.docx (see Appendix A), DepartmentEmployeeUML.pdf (see Figure 1),

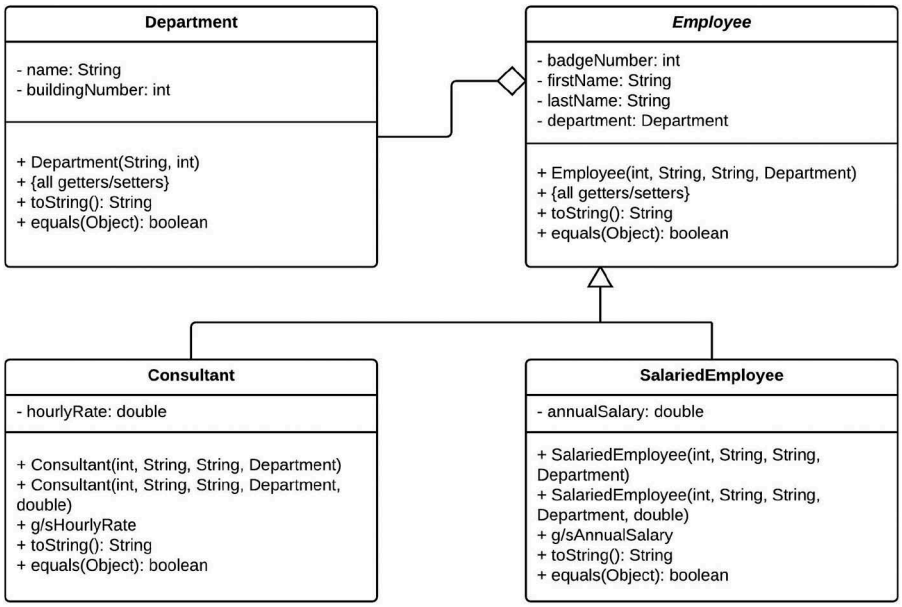

Figure 1. The UML class diagram of the "Department-Employee" project.

```
Executive 100
Research 112
Engineering 115
Production 120
Accounting 122
```

Figure 2. The content of depts.txt.

depts.txt (see Figure 2), and DepartmentEmployeeTest.java (see Figure 3).

Students are required to use the UML class diagram in DepartmentEmployeeUML.pdf (see Figure 1) to implement the four classes: Department, Employee, Consultant, and SalariedEmployee. They are also required to use the provided DepartmentEmployeeTest.java (see Figure 3) to test their implementation. The correct output is also provided so that students can compare it with their program output for accuracy. When finished, students are required to zip all four .java files (Department.java, Employee.java, Consultant.java, and SalariedEmployee.java) and submit the archive file.

Project evaluation

The evaluation of the "Department-Employee" project will be based on three types of data: the instructor score reports, the correlation between the project score and the final course score, and a survey of student perceptions. After having received the student submissions to the project, the instructor will assess them and assign a score to each of the submissions accordingly using a predefined grading rubric. Both the project score and the final score for the course will be collected at the end of each semester. The correlation coefficient will be calculated and a scatter plot will be created to show the relationship between them. The student perceptions will be collected from an online survey using SurveyMonkey.com. The survey includes nine questions; the first two are essay questions, which are related to the student perceptions of the project.

Course background, grading rubric, score reports, and correlation test

Course background

The project was assigned to undergraduate Computer Information Systems (CIS) and Computer Science (CS) students enrolled in an advanced OOP course (comparable to a CS2 course). These students are generally juniors or seniors who are required to have completed an introductory programming course (comparable to a CS1 course) that focuses mainly on structured programming concepts, or an introduction to computer science course that focuses mainly on the theoretical foundations of computer science and the components of algorithms. The students are required to pass either of these two prerequisite courses with a grade of C or higher.

The content covered in this advanced OOP course can be roughly divided into five major groups: (1) fundamentals of programming, i.e., a quick review of structured programming such as control statements, methods, and arrays; (2) objected-oriented programming such as classes and objects, as well as inheritance and polymorphism; (3) GUI programming such as event-driven animations, UI controls, and multimedia; (4) data structures and algorithms such as lists, stacks, queues, binary search trees, and graphs; and (5) advanced Java programming such as multithreading, networking, and accessing databases with Java Database Connectivity (JDBC). The content of this course is comparable to the content suggested by.[11] The "Department-Employee" project was assigned to the students after structured programming constructs were reviewed, object-oriented concepts were presented and practiced, but before the GUI programming techniques were introduced. Students had been given 10 previous less-complicated programming assignments before this one was introduced. These assignments focused on practicing with objects

```
package cis315.dep;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Scanner;

public class DepartmentEmployeeTest {
	public static void main(String[] args) {
		ArrayList<Department> depts = readDepartmentsFromFile("depts.txt");
		displayDepartments(depts);

		ArrayList<Employee> company = new ArrayList<>();
		populateEmployeeArray(company, depts);

		// Display everyone's pay w/o referring to the subclasses (i.e., using
		// Employee only)
		for (Employee e : company) {
			System.out.println(e);
		}

		// Display all departments that are equal to this one:
		Department d1 = new Department("Research", 112);
		for (Department d : depts) {
			if (d1.equals(d)) {
				System.out.println(d);
			}
		}

		// Display all departments that are equal to this one:
		Department d2 = new Department("Accounting", 50);
		for (Department d : depts) {
			if (d2.equals(d)) {
				System.out.println(d);
			}
		}
```

Figure 3. The source code of DepartmentEmployeeTest.java.

and classes, object-oriented thinking, inheritance and polymorphism, exception handling and text I/O, as well as abstract classes and interfaces. The students were required to complete the "Department-Employee" project within 72 hours after it was made available online. Students could work in or outside of a computer lab.

Grading rubric

Appendix B contains a set of code examples for the project that will earn a perfect score. To simplify the grading process and ensure consistency, a grading rubric was developed for the assignment (see Table 1 for the grading rubric). Each submission for the project was required to contain four .java files, i.e., Department.java, Employee.java, Consultant.java, and SalariedEmployee.java. A perfect submission was assigned a score of 100 points (pts), with each perfect .java file being assigned a score of 25 pts. Specifically, each of the four .java files was assessed using a scale of six possible ratings, including (1) perfect—25 pts, (2) good—20 pts, (3) satisfactory—15 pts, (4) below average—10 pts, (5) clear failure—5 pts, and (6) no submission—0 pts.

Score reports

This advanced objected-oriented programming course is typically offered twice a year: online during the summer term and face-to-face during the fall term. The project was assigned to students and the students' project scores were collected during the following four semesters: summer 2016, fall 2016, summer 2017, and fall 2017. The number of enrolled students for the course was 19 in summer 2016, 17 in fall 2016, 10 in summer 2017, and 22 in fall 2017. For the total of 68 students over the past four semesters, the average project score was 87.21 pts (87.21%). The differences among the average scores for each .java file were not significant: Department.java (21.62, 86.47%), Employee.java (21.99, 87.94%), Consultant (21.76, 87.06%), and SalariedEmployee (21.84, 87.35%).

For the majority of students who did not earn a perfect score on this project, the primary reason was because they failed to correctly implement the toString() methods and/or the equals() methods for some or all of the four required classes. Students were expected to implement the toString() method based on the example output provided to them. The implementation of the equals() method was based on a simple OOP convention. That convention, discussed in the classroom, is that two objects of the same class are deemed to be equal when all of their corresponding fields have the same values (i.e., each field is "equal"). The requirements in this area provide an opportunity for interesting classroom discussions regarding what make objects "equal" and how these

```
    }

    private static void displayDepartments(ArrayList<Department> list) {
        // Display the list to standard error (NOT standard output)
        for (Department d : list) {
            System.err.println(d);
        }
    }

    private static ArrayList<Department> readDepartmentsFromFile(String filename) {
        ArrayList<Department> list = new ArrayList<>();
        File file = new File(filename);

        Scanner sc = null;
        try {
            sc = new Scanner(file);
            while (sc.hasNext()) {
                String dept = sc.next();
                int building = sc.nextInt();
                list.add(new Department(dept, building));
            }

        } catch (FileNotFoundException e) {
            System.err.println("Department file not found: " + e.getMessage());
        } finally {
            if (sc != null) {
                sc.close();
            }
        }

        return list;
    }

    public static void populateEmployeeArray(ArrayList<Employee> array, ArrayList<Department> departments) {
        SalariedEmployee exec = new SalariedEmployee(100, "Steve", "Jobs", departments.get(0), 100000000);
        SalariedEmployee architect = new SalariedEmployee(200, "Bill", "Joy", departments.get(1), 30000000);
        SalariedEmployee engineer = new SalariedEmployee(300, "James", "Gosling", departments.get(2), 30000000);
        Consultant consultant = new Consultant(1100, "Craig", "McClanahan", departments.get(3), 25000);
        SalariedEmployee accountant = new SalariedEmployee(400, "Kevin", "Malone", departments.get(4), 5156700);

        array.add(exec);
        array.add(architect);
        array.add(engineer);
        array.add(consultant);
        array.add(accountant);
    }
}
```

Figure 3. Continued.

Table 1. The grading rubric.

	Perfect	Good	Satisfactory	Below Average	Clear Failure	No Submission
Department.java	25 pts	20 pts	15 pts	10 pts	5 pts	0 pts
Employee.java	25 pts	20 pts	15 pts	10 pts	5 pts	0 pts
Consultant.java	25 pts	20 pts	15 pts	10 pts	5 pts	0 pts
SalariedEmployee.java	25 pts	20 pts	15 pts	10 pts	5 pts	0 pts

decisions should be made. These discussions should be revisited when database programming using JDBC is presented since primary keys can greatly simplify the implementation of the equals() method.

Correlation test

To test whether there is a linear relationship between the project score and the corresponding final course score, we collected both scores, a total of 68 pairs. We ran a simple correlation test on the project score and the final course score and obtained a correlation coefficient (r) of about 0.51. A scatter chart was also created to show the relationship between the project score and the final course score graphically (see Figure 4).

In statistics, the correlation coefficient r measures the strength and direction of a linear relationship between two variables on a scatter chart. The value (r = 0.51) of the correlation coefficient indicates that there is a moderate positive relationship between a student's project score and his/her

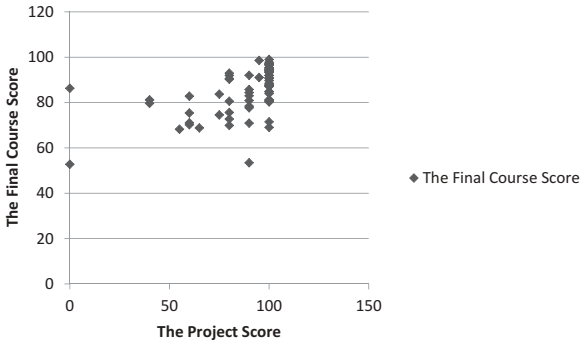

Figure 4. The scatter chart of the project score and the final course score.

final course score. This means that the project score can be used to predict the overall success of a student for the course. To a certain degree, this moderate positive relationship shows that the "Department-Employee" project is effective for assessing students' OOP skills with Java. However, it is worth noting that this relationship is not very strong. One reason may be that the final score for the course is based on multiple sections in this class, including fundamentals of programming, OOP, GUI programming, data structures and algorithms, and advanced Java programming.

Student perceptions and backgrounds

Student perceptions

To evaluate the students' backgrounds and their perceptions of the project, an online survey was created using SurveyMonkey.com. Using this tool helped ensure anonymity so that students would be more comfortable in sharing their thoughts and perceptions. The survey included nine questions: two essay questions, six multiple choice questions, and one short-answer question (see Appendix C for the survey questionnaire). The survey link was distributed to students by course announcement through Canvas, the university's learning management system (LMS). Participation in the survey was voluntary and no extra credit was awarded as an incentive for students to take the survey. Of the 68 students enrolled over the past four semesters, 56 (82.35%) students completed the survey (the full survey results are available upon request).

The first essay question asked whether the student liked the project and why they did or did not like it. In total, 42 students (75.00%) responded positively towards the project. Students who responded positively said that the project helped them to better understand the object-oriented side of programming and the concept of inheritance. Some of the positive comments included:

- "Yes, the project is a good way to demonstrate both inheritance and aggregation in Java in one task. Overall, it wasn't very difficult but focused on two important concepts and required you to figure them out to learn."
- "Yes, it brought home a lot of Java concepts that I was still unsure of and reminded me how flexible object-oriented programming is."
- "I felt as if this project resembled more closely the programming techniques we would use at a real job such as file I/O, inheritance hierarchies, and exception handling."

Seven students (12.50%) responded negatively to the project, and seven students (12.50%) responded neutrally to the project. The most commonly cited issue in the negative and neutral comments was a lack of clarity in the project instructions.

The second essay question asked students to list the most difficult challenges that they faced while working on the project. The challenges most commonly cited by students were the implementation of the toString() method and the equals() method. This is consistent with the instructor score reports. Other common challenges included vague instructions (as noted in the first essay question) and understanding when functions needed to be overridden. Some examples of these comments are:

- "The hardest challenge for me was the toString and equals method. I was confused on what I was supposed to be returning on each part of the code. After looking over the instructions a little more carefully I started to understand what parts were supposed to return."
- "My most difficult challenge was deciding what needed to go into the overridden equals in each file. We were only instructed on what we needed for the department file, but told to override all of the equals methods and I personally like to know more detail in my instructions. Other than that I did not have much of a problem with the assignment."
- "I still have trouble overriding the toString() and equals() methods. I can never remember which checks to use in the equals method, and it takes me a second to understand what exactly the toString() method is needing to return."

Student backgrounds

The remainder of the questions (three through nine) asked students questions relating to their backgrounds. In regard to question three, 37 students (66.07%) identified their major as Computer Information Systems, 17 students (30.36%) as Computer Science, and the remaining two students (3.57%) as Geographic Information Science. The results to question four indicated that 49 students (87.50%) were male, and 7 (12.50%) were female. Question five indicated that 47 students (83.93%) were seniors, 6 (10.71%) were juniors, 2 (3.57%) were sophomores, and 1 (1.79%) was of another classification. According to question six, 51 students (91.07%) were full-time students, and 5 (8.93%) were part-time.

Question seven was a short-answer question asking for the student's age. The results showed that the average age of the students surveyed was 23 with a standard deviation of 3.41. The highest recorded age was 37 and the lowest was 18. Question eight asked for the student's current overall GPA: 15 students (26.79%) replied with 3.50–4.00, 21 students (37.50%) responded with 3.00–3.49, 15 students (26.79%) replied with 2.50–2.99, and the remaining 5 students

(8.93%) responded with 2.00–2.49. The final question asked whether students had taken the course CIS 225: Introduction to Object Oriented Programming. Thirty-eight students (67.86%) said that they had taken the course and 18 students (32.14%) said that they had not.

Discussion

Summary of contents

This article described the "Department-Employee" project, which was used to assess students' OOP skills with Java. The article reviewed the history of OOP, provided an overview of OOP with Java, and summarized existing Java projects and their limitations. It provided the project specifications as well as the course background, grading rubric, score reports, and correlation test. It also presented survey data for evaluating students' backgrounds, as well as the students' perceptions regarding the project. Results from the instructor score reports, correlation test, and student perceptions showed that the project was effective for assessing students' OOP skills with Java.

Pedagogical implications

OOP Java textbooks and published Java teaching case studies offer similar but typically smaller projects compared to the "Department-Employee" project discussed in this paper. However, our "Department-Employee" project provides some important implications and offers several advantages. One, our project is more comprehensive. To complete this project, students are required to understand and apply all of the major concepts and techniques pertaining to OOP in Java. These concepts and techniques include classes and objects, aggregating classes and aggregated classes, encapsulation and information hiding, superclasses and subclasses, inheritance, method overriding and overloading, abstract classes and methods, polymorphism and dynamic binding.

Two, our project is more flexible and it is much easier to customize. (1) This project can be assigned as an individual project or as a group project. When assigned as a group project, instruments can be applied to capture the group dynamics and collaborations. (2) This project can be assigned with or without the UML class diagram. When assigned without the UML class diagram, students have the opportunity to practice transforming project requirements into a UML class diagram and on how to search for and use a tool to draw the UML class diagram. (3) This project can be assigned with or without the test driver, i.e., DepartmentEmployeeTest.java. When assigned without the test driver, students have the opportunity to design and implement the test driver so that the program will produce the expected output. (4) This project can be assigned with or without the expected output and the test driver. When assigned without the expected output and the test driver, students have the opportunity to design and implement a test driver that will verify and validate all of their implemented classes and their related methods. (5) The project completion time can be varied, e.g., from 12 hours to a whole week. This would certainly change the approaches that students will take when completing the project. Of course, instructors can always combine any of the above suggested options to customize their implementation of the project.

Three, our project is more holistic. Particularly, we have provided the course background, the grading rubric, the score reports, survey data on students' backgrounds, and the students' perceptions regarding the project. We believe that all of these resources can assist other instructors in developing a holistic view of the project.

Limitations and future research

This study has several limitations. First, the respondents were drawn from one university, and the sample size, although adequate for this study, was somewhat limited. As such, future research should use respondents from several universities and try to increase the sample size to a significant magnitude. Second, during each of the four semesters, the project was assigned to students as an individual project. In the future, we would like to assign this project to students as a group project. This would provide an invaluable opportunity for instructors to capture data of group dynamics and for students to engage in group collaborations and collaborative thinking when trying to solve real-world problems in a timely and effective manner. Third, the project was only evaluated with the instructor score reports, correlation of the project score and the final course score, and student perceptions from an online survey. Future research should try to collect other types of data from a different perspective. For instance, we would like to conduct interviews of students and triangulate the results with the survey results. By doing this, we should be able to gain more insight about the validity of the "Department-Employee" project. In summary, there are additional opportunities to improve the project itself, the assessment approach, and the process of delivering the project and carrying out the assessment.

Concluding remarks

Teaching OOP and helping students think in terms of objects and classes can be a challenge. The project presented in this paper, along with the guidelines and suggestions, should assist instructors in moving students toward that goal. We have demonstrated that our students have benefited from this assignment and we strive to find or create similar tools in the future that will help us gauge student concept comprehension and skill mastery in OOP.

References

[1] TIOBE index. 2018. https://www.tiobe.com/tiobe-index/.
[2] Meyerovich LA, Rabkin AS. Empirical analysis of programming language adoption. ACM SIGPLAN Not. 2013;48(10):1–18. doi:10.1145/2544173.
[3] Cavaiani TP. Object-oriented programming principles and the Java class library. J Inf Syst Educ. 2006;17(4):365–68.

[4] Shein E. Python for beginners. Commun ACM. 2015;58(3):19–21. doi:10.1145/2739250.
[5] Ramesh V, Wu J-TB. An approach to teaching object-oriented programming concepts in business schools. Decis Sci J Innovative Educ. 2004;2(1):83–87. doi:10.1111/dsji.2004.2.issue-1.
[6] Ourosoff N. Primitive types in Java considered harmful. Commun ACM. 2002;45(8):105–06. doi:10.1145/545151.545182.
[7] Westfall R. Hello, world considered harmful. Commun ACM. 2001;44(10):129–30. doi:10.1145/383845.383874.
[8] Johnson RA, Moses DR Objects-first vs. structures-first approaches to OO programming education: an empirical study. Acad Inf Manage Sci J. 2008;11(2):95–102.
[9] Hu C. Just say 'a class defines a data type. Commun ACM. 2008;51(3):19–21. doi:10.1145/1325555.
[10] Xing C-C, Belkhouche B. On pseudo object-oriented programming considered harmful. Commun ACM. 2003;46(10):115–17. doi:10.1145/944217.
[11] Pendergast MO. Teaching introductory programming to IS students: Java problems and pitfalls. J Inf Technol Educ. 2006;5:491–515. doi:10.28945/261.
[12] Pokkunuri BP. Object oriented programming. ACM SIGPLAN Not. 1989;24(11):96–101. doi:10.1145/71605.
[13] Rentsch T. Object oriented programming. ACM SIGPLAN Not. 1982;17(9):51–57. doi:10.1145/947955.
[14] Booch G. Object-oriented analysis and design with applications. Cambridge, MA: Addison-Wesley; 1991.
[15] Rumbaugh J, Blaha M, Premerlani W, Eddy F, Lorensen W. Object-oriented modeling and design. Englewood Cliffs, NJ: Prentice-Hall; 1991.
[16] Jacobson I. Object-oriented software engineering: a use case driven approach. Cambridge, MA: Addison-Wesley; 1993.
[17] Weisfeld M. The object-oriented thought process. 3rd ed. Cambridge, MA: Addison-Wesley; 2008.
[18] Eckel B. Thinking in Java. 1st ed. Upper Saddle River, NJ: Prentice Hall PTR; 1998.
[19] Benander AC, Benander BA, Lin M. Perceptions of Java-Experienced programmers' perspective. J Comput Inf Syst. 2003;43(4):1–7.
[20] Zhang X. Assessing students' structured programming skills with Java: the 'blue, berry, and blueberry' assignment. J Inf Technol Educ. 2010;9:227–35.
[21] Deitel PJ, Deitel H. Java how to program, late objects. 11th ed. New York City, NY: Pearson; 2017.
[22] Liang YD. Introduction to Java programming and data structures, comprehensive version. 11th ed. New York City, NY: Pearson; 2018.

Appendices

Appendix A: Project instructions

Department-Employee Project Instructions

Files you get: DepartmentEmployeeProjectInstructions.docx, DepartmentEmployeeUML.pdf, depts.txt, and DepartmentEmployeeTest.java.
Files you need to submit: Department.java, Employee.java, Consultant.java, and SalariedEmployee.java.
Use the UML DepartmentEmployeeUML to implement the four classes: Department, Employee, Consultant, and SalariedEmployee. Use the provided EmployeeTest to test your implementation.
If implemented correctly, the output of your program should look like the following:

```
Executive 100
Research 112
Engineering 115
Production 120
Accounting 122
100 Steve Jobs Executive 100 100000000
200 Bill Joy Research 112 30000000
300 James Gosling Engineering 115 30000000
1100 Craig McClanahan Production 120 25000
400 Kevin Malone Accounting 122 5156700
Research 112
```

Submission: Zip all 4 .java files and submit the .zip file to this drop box.

Appendix B: Source code

Department.java

```
package cis315.dep;
public class Department {
   private String name;
   private int buildingNumber;
   public Department(String name, int num) {
      this.name = name;
      this.buildingNumber = num;
   }
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
   public int getBuildingNumber() {
      return buildingNumber;
   }
   public void setBuildingNumber(int buildingNumber) {
      this.buildingNumber = buildingNumber;
   }
   @Override
   public String toString() {
      return name + " " + buildingNumber;
   }
   @Override
   public boolean equals(Object obj) {
      if (obj == null) {
         return false;
      }
      if (!(obj instanceof Department)) {
         return false;
      }
      Department other = (Department) obj;
      return (name.equals(other.name) && (buildingNumber == other.buildingNumber));
   }
}
```

Employee.java

```
package cis315.dep;
public abstract class Employee {
   private int badgeNumber;
   private String firstName;
   private String lastName;
   private Department department;
   public Employee(int badge, String fname, String lname, Department dept) {
      this.badgeNumber = badge;
      this.firstName = fname;
      this.lastName = lname;
      this.department = dept;
```

(*Continued*)

(Continued).

```
    }
    public int getBadgeNumber() {
        return badgeNumber;
    }
    public void setBadgeNumber(int badgeNumber) {
        this.badgeNumber = badgeNumber;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    public Department getDepartment() {
        return department;
    }
    public void setDepartment(Department department) {
        this.department = department;
    }
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (!(obj instanceof Employee)) {
            return false;
        }
        Employee other = (Employee) obj;
        return (department == other.department) && (this.badgeNumber == other.badgeNumber)
                && (this.lastName.equals(other.lastName)) && (this.firstName.equals(other.firstName));
    }
    @Override
    public String toString() {
        return badgeNumber + " " + firstName + " " + lastName + " " + department;
    }
}
```

Consultant.java

```
package cis315.dep;
public class Consultant extends Employee {
    private long hourlyRate;
    public Consultant(int badge, String fname, String lname, Department dept) {
        super(badge, fname, lname, dept);
        hourlyRate = 0;
    }
    public Consultant(int badge, String fname, String lname, Department dept, long rate) {
        super(badge, fname, lname, dept);
        hourlyRate = rate;
    }
    public long getHourlyRate() {
        return hourlyRate;
    }
    public void setHourlyRate(long hourlyRate) {
        this.hourlyRate = hourlyRate;
    }
    // same hourly rate
    public boolean equals(Object obj) {
        if (super.equals(obj)) {
        if (obj instanceof Consultant) {
        Consultant other = (Consultant) obj;
        return (hourlyRate == other.hourlyRate);
            }
        }
        return false;
    }
    @Override
    public String toString() {
        return super.toString() + " " + hourlyRate;
    }
}
```

SalariedEmployee.java

```
package cis315.dep;
public class SalariedEmployee extends Employee {
    private long annualSalary;
```

(*Continued*)

(Continued).

```
    public SalariedEmployee(int badge, String fname, String lname, Department dept) {
        super(badge, fname, lname, dept);
        annualSalary = 0;
    }
    public SalariedEmployee(int badge, String fname, String lname, Department dept, long salary) {
        super(badge, fname, lname, dept);
        annualSalary = salary;
    }
    public long getAnnualSalary() {
        return annualSalary;
    }
    public void setAnnualSalary(long annualSalary) {
        this.annualSalary = annualSalary;
    }
    @Override
    public String toString() {
        return super.toString() + " " + annualSalary;
    }
    // same dept and same salary
    public boolean equals(Object obj) {
        if (super.equals(obj)) {
            if (obj instanceof SalariedEmployee) {
                SalariedEmployee other = (SalariedEmployee) obj;
                return (annualSalary == other.annualSalary);
            }
        }
        return false;
    }
}
```

Appendix C: The survey questionnaire

The "Department-Employee" Project Survey Questionnaire

(1) Introduction

This survey includes two essay questions and seven multiple choice questions. The two essay questions are about your perceptions of the "Department-Employee" project. The seven multiple choice questions are about your background. Please answer those two essay questions with as much details as possible, and answer those seven multiple choice questions as accurately as possible. Note that your responses will be totally anonymous. We appreciate your time and effort. Thank you.

(2) Essay Question 1 of 2

Do you like the "Department-Employee" project? Why or why not?

(3) Essay Question 2 of 2

What are the most difficult challenges that you encounter while working on this "Department-Employee" project? Please list all the challenges and provide as much details as possible.

(4) Multiple Choice Question 1 of 7

What is your major?

- Computer Information Systems (CIS)
- Computer Science (CS)
- Other (please specify)

(5) Multiple Choice Question 2 of 7

What is your gender?

- Female
- Male

(6) Multiple Choice Question 3 of 7

What is your current classification?

- Freshman
- Sophomore
- Junior
- Senior
- Other than any of the above

(7) Multiple Choice Question 4 of 7

(*Continued*)

(Continued).

The "Department-Employee" Project Survey Questionnaire

Are you currently a full-time or part-time student?
- Full-time
- Part-time

(8) Multiple Choice Question 5 of 7 (This is actually a short-answer question)
What is your age?

(9) Multiple Choice Question 6 of 7
What is your current overall GPA?
- 1.49 or less
- 1.50–1.99
- 2.00–2.49
- 2.50–2.99
- 3.00–3.49
- 3.50–4.00

(10) Multiple Choice Question 7 of 7
Have you taken CIS 225: Introduction to Object-Oriented Programming?
- Yes, I have.
- No, I haven't.
- Don't know.