Progressive Programming Assignments

Keith J. Whittington
Rochester Institute of Technology, Rochester, NY, USA

kjw@it.rit.edu

Abstract

Progressive programming assignments were used in an introductory Java programming course where every assignment built on the previous one. The major goal was to help students learn difficult, abstract concepts. This technique allows students to concentrate on the current topic while building on their previous work. This also provides an incentive for students to keep up with their work. Students often feel that they can skip the current topic and pick up after it is over, but it is difficult to do this in a programming course because every new concept builds on the previous ones. This approach also has built-in scalability, which is difficult to achieve in introductory programming courses due to time constraints and the students limited knowledge.

These assignments were given in a CS2-type course where the topics predominantly deal with abstract concepts. This paper discusses the assignments, goals, faculty observations, student comments, and results.

Keywords: Introductory programming, Java, paradigms of learning, programming assignments.

Introduction

One constructivist view of learning is that learners construct new knowledge based on their prior knowledge (Carbone, Hurst, Mitchell, & Gunstone, 2001; Hadjerrouit, 1999). This view was basic idea behind the development of programming assignments used in an introductory Java programming course. Every assignment built on the previous one, which allowed the students to concentrate on the new concepts. In this way, the previous programming assignments became the knowledge base that the students build upon. This approach was also helpful in this particular course since it mainly dealt with abstract concepts that freshmen students typically have difficulty understanding (Whittington & Bills, 2004: Whittington, Bills, & Hill, 2003). The new topics included inheritance constructs, abstract classes, interfaces, user-defined exceptions, I/O, and GUI.

An unexpected benefit of this process was the increased discussions that ensued during the lectures. Since the students were aware that the new materials were going to be incorporated into their previous assignment, they were always asking questions about how the new material related to their next assignment. They were also motivated to listen to discussions on modularization, simplification, and elimination of duplicate code, because their code got increasingly harder to control. In previous courses, students tended not to appreciate the importance of these topics. This approach also helped the students deal with scalability and continuously changing requirements, which typically happens in industry. By the end of the course, the students had created fairly complex programs consisting of thousands of lines of code.

In the first assignment, the students created a program that incorporated everything they had covered in their previous programming courses. They were initially given detailed requirements but very little design direction. This allowed them to create programs in the best way that they could, and the resulting programs represented their current conceptual model of Java programming. Subsequent assignments dealt with properly designed programs.

This paper describes the details of each assignment, the pedagogical goals, faulty observations, student comments, and the results of this experiment.

Background

The example used in this paper was used in an introductory Java programming sequence designed for students who struggled in their first programming course. Typically, the second course in the IT programming sequence presents a significant challenge to a lot of students. So a two-course sequence was offered as an alternative to the traditional second programming course. This allowed these students to take twice as long to cover the same material. This alternatively paced sequence turned out to be extremely effective and resulted in a dramatic increase in student retention, grades, and self-confidence (Whittington, Bills, & Hill, 2003).

The assignments were given in the second course of this alternative sequence. The first course of this sequence concentrates on solidifying the student's of OOP conceptual knowledge and the second course deals with abstract concepts such as inheritance, user-defined exceptions, I/O, and GUI. The students had varying amounts of time to complete each assignment based on the complexity of the assignment. These first year, under-prepared students eventually created complex, well designed programs containing thousands of lines of code.

On the first day of class the students were told that all of their assignments would build on the previous assignment. This was done to provide them with the incentive to keep up with the course material and assignments. Several students complained that it wasn't fair because if they fell behind it would be difficult to catch up. This was a valid point, but they were also told that programming is a cumulative process where every new concept builds on the previous ones. So if you don't understand one topic, you will have a difficult time understanding the next topic.

The Initial Program

Although this problem isn't particularly brilliant or innovative, it was effective.

Figure 1 shows the initial design of the program. This diagram is in a pseudo-UML format where the method and attribute details were removed and only the class name and relationships are included. This was done to simplify the diagram and to remove details that do not contribute to this discussion.

This program collects information about employees of a corporation. The Person class contains basic information about a person, such as their name, address, and employee number. The Employee class is an abstract class that extends the Person class. It contains one abstract method that calculates the weekly pay for the Employee, plus other common information such as department name. The SalaryEmployee and HourlyEmployee are concrete extensions of Employee. These classes define the inherited abstract method and include additional attributes and methods. For instance, SalaryEmployee contains the yearly salary, and HourlyEmployee contains the hours worked and the hourly wage.

The Corporation class provides the user interface that gathers information about each employee and instantiates objects of SalaryEmployee and HourlyEmployee.

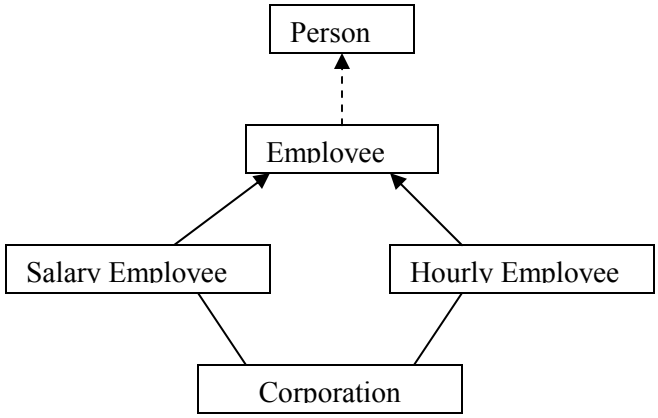

Figure 1: Program Initial Design

Assignment 1

New Topic

Abstract class

Program Requirements

Only the attributes with their acceptable values and method behaviors were specified for the classes. They had to create accessors, mutators, methods, default constructor, and a multi-parameter constructor based on these specifications.

They were instructed to get all user input in Corporation class. The two employee types had to be initialized in different ways: salary employees with a default constructor, and hourly employees with a multi-parameter constructor. They were also required to create a collection to store the employee objects and had to keep asking for new employees until the user indicated they wanted to stop. At the end of the program they had to identify and list all the attributes for every employee in the collection.

Pedagogical Goals

The main goal of the first assignment was to have them create several classes to build on throughout the term. The purpose for using multiple constructors was to have the students create objects in two different ways. Creating default objects then adding each attribute value is appropriate for command-line input, and gathering all attribute values first then creating the object is more appropriate for GUI applications. Although they would not create GUI applications until the end of the course, they were required to use multi-parameter constructors in order to have a working constructor before they converted their programs.

Very little direction was given for the Corporation class. They had complete freedom to design this class. This also allowed the students who wanted to put everything in one method to have their code grow out of control. The students had been told the merits of modularization in previous courses, but most of the students ignored this principle because their programs were never large enough to become unmanageable. Predictably, without specific constraints and direction, most of the students put everything in the main method and created deeply nested loops and if-statements. Another purpose for allowing them to loose control was to make them proficient at

writing decisions and loop statements—deeply nested code is difficult to control. In subsequent assignments they were shown how modularization would help them manage their existing code.

Collections were used to make the students manage multiple objects in the same program.

Observations and Experiences

There were more questions than usual concerning the specifics of the programming assignment. This was most likely due to the student's realization that their success in the course depended on completing the first assignment.

Assignment 2

Figure 2 shows the design of the program for assignment 2. This diagram is in also in a pseudo-UML format where the method and attribute details were removed.

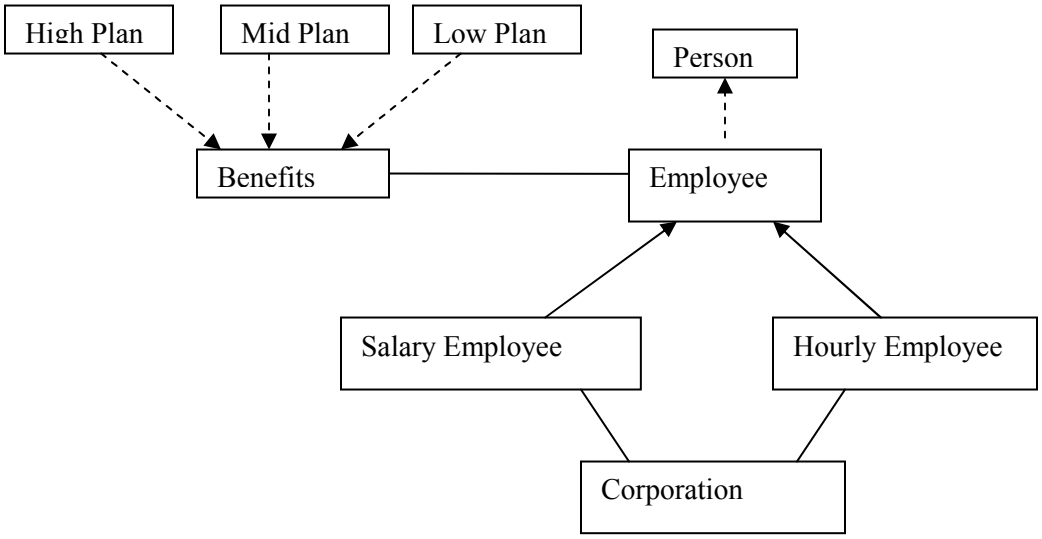

Figure 2: Assignment 2 Design

New Topic

Interface

Program Additional Requirements

An interface class was added to the program specifications. Since employees typically get to choose from a variety of medical plans, they were given the specifications for three concrete implementations of the interface (HighPlan, MidPlan, LowPlan). The Benefits interface was added as an attribute of the Employee class.

In the Corporation class they had to ask the user for the type of benefit they wanted, create the appropriate object, and assign it to the attribute in the Employee object. They also had to add sev-

eral options that extracted information from the objects in the collection. One example was to get the name of the employee and display their mailing address.

Pedagogical Purposes

The choice of using a benefits package as the interface was to provide the students with a "real-world" example where a class defines the requirements but the details are unknown. In this case, every employee gets a benefit package, but the specifics vary depending on their choices. Making Benefits the attribute type made the students deal with inheritance concepts because they had to assign a concrete class to the interface data type, it also gave them practice using aggregate objects

The purpose for retrieving information from objects inside a collection was to help them realize that collections are used for more than just a repository of objects.

Another purpose for adding additional requirements into the Corporation class was to have them struggle with ever increasing demands. Since most of the students had written all their code inside the main method, they would experience, first hand, the problem with disorganized, non-modularized code.

Observations and Experiences

Unexpectedly, the progressive programming assignments were creating a more cohesive course where the lectures and homework assignments became intimately intertwined. Another positive development was the increased classroom discussions that ensued due to the questions that were raised concerning the programming assignment.

The students struggled with the concept of assigning a concrete class to an interface variable. For example, since the HighPlan class inherits from the Benefits interface, it is-a Benefits class. However, several students did start to use words like "is-a" in their conversations concerning inheritance.

Discussions also ensued concerning the various operations performed on specific objects in the collection. They had difficulty conceptualizing the idea of accessing an object once it was stored in a collection.

Miraculously, the students were maintaining their code in the Corporation class even though the majority of them had hundreds of lines of code in one method.

Assignment 3

New Topics

User-defined exceptions and general purpose methods using inheritance relationships.

Program Additional Requirements

They had to create their own exception class and throw the exception whenever an invalid value was passed to a mutator. They had to catch these exceptions in the Corporation class and ask the user to re-enter a correct value.

They were finally instructed to organize their Corporation class. They had to provide a menu that called a different method for each option. They also had to create several general-purpose methods that utilized inheritance relationships. For instance, one method had to accept any employee type as a parameter and display the attributes of the Person class.

Pedagogical Purposes

User-defined exceptions are a good way to enhance student understanding of inheritance. All that is needed is to extend the Exception class and the class behaves like every other exception class. This also showed them a different way to deal with errors. Instead of returning a Boolean value where there is no guarantee that the caller of the method will look at the returned value, when an exception is thrown, the caller of the method has to deal with it.

Since the students were still wrestling with inheritance concepts, several methods were added to enhance their knowledge and increase their comfort with inheritance principles. For instance, if a method defines its parameter as an Employee then you can pass either an hourly or a salary employee object as the parameter because hourly and salary employees are employees. Inheritance is typically a very difficult concept to understand and is a fundamental concept of object-oriented programming.

The purpose of the modular requirements in the Corporation class was to show the students how to get their code under control and appreciate the benefits of modularized code.

Observations and Experiences

Students were surprised to discover latent errors in their code now that exceptions were thrown. Most of these errors occurred in their constructors because their constructors called mutators to initialize their attributes. Since constructors do not return values, the correctness of the constructor code is never verified. Several students observed that they now had self-policing code. This was a major breakthrough. In previous courses students had never vocally appreciated the effort to incorporate exceptions.

The addition of several general-purpose methods utilizing inheritance relationships seemed to help more students understand inheritance at a deeper level. Inheritance is a difficult concept to understand, but once you do, coding becomes more efficient and versatile. They were also told about additional reasons for creating general-purpose methods. For example, if a new Employee type is created in the future, they can still use the same methods.

Before the students saw the new modularization requirement for the Corporation class, they were shown a main method that contained only a handful of lines. This got an instant reaction and the students begged to see how it was done. At this point, the students were highly motivated to change their code because they had suffered while trying to maintain their disorganized code. In prior years, the students never really embraced the merits of modularization.

Assignment 4

New Topics

IO Classes and Tokenizers

Program Additional Requirements

They had to store the attribute values of all the objects inside the collection into a file. When the program started, the program had to read the file, create each object with its attribute values, and store the objects into the collection. Every time their programs exited, they had to re-write the file with the existing data in the collection. This way, information could be used from one program invocation to another. They were also required to use a Java Tokenizer class to extract each item out of the file.

Pedagogical Purposes

IO is also another difficult concept to grasp and this assignment exposed them to a variety of classes and techniques. Efficient IO processing in Java requires using nested IO classes. To discourage the students from using non-nested classes, the requirements were difficult enough that it necessitated the use of nested classes. The Tokenizer classes introduced them to parsing and made extracting the individual pieces of data easier.

This assignment also introduced them to the purpose for databases by creating a program that retains information. Plus, by making them use flat files, they also got to see why databases were created.

Observations and Experiences

The students probably had the most difficulty with this assignment. Creating the proper IO classes and correctly tokenizing a string is fairly difficult to accomplish. This is another instance where the progressive assignments helped. The students were able to concentrate solely on the IO difficulties without having to spend time writing a new program.

By this time, the students had grown tired of entering data every time they tested their program, so despite their difficulties, the students expressed appreciation for the ability to reuse the data.

Assignment 5

New Topic

GUI applications

Program Additional Requirements

They had to convert their programs from a command-line program into a GUI application. They had to create an entry form for a salary employee and another form for an hourly employee. They created a third window that displayed the attribute values for each employee object. They tied these forms together by using another GUI window that let the user choose between the forms to create or display the objects.

Pedagogical Purposes

The primary purpose of this assignment was to expose them to event-driven, GUI programming. However, another strong purpose was to demonstrate how easily a properly designed program can be converted from one style to another. Their existing code for the Person, Employee, SalaryEmployee, HourlyEmployee classes could be used without any modifications. Since the user now entered in all of the information before sending the data, they now had a real reason to use the multi-parameter constructor. This also demonstrated the reason to create versatile code because you never know how a program will be used in the future.

Observations and Experiences

Having worked so hard at creating stable classes, the students commented at how easy it was to convert a command-line program into a GUI application. They also commented that coding GUI applications were fairly easy because they had spent so much time dealing with inheritance. Some students also stated that they were glad that they had already created the code to use a multi-parameter constructor because it made the conversion a lot easier.

Student Comments

The students were surveyed at the end of the course to elicit their opinions. Eighty percent of the students indicated that they preferred the progressive programming assignments over the individual assignments.

The following statements are a sampling of the comments made by the majority of the students:

- Building on the assignment helped me understand the concepts better
- I liked the building upon past assignments because you can apply new knowledge to old knowledge
- I didn't have to spend a lot of time reinventing the wheel. I could spend that important time working on the problem
- One of the most effective learning aides was using/modifying one program for all the assignments

However, a few students didn't like this method and their comments are summarized below:

- Having a different homework each week reinforces what you've learned already through repetition. when you build on the last homework, you're only dealing with the new stuff and I tend to forget the old
- I dislike homework that builds on the previous assignment. If you fall behind, it's very hard to catch up

This assignment was given in the second course of this special 2-course sequence where the material is primarily abstract in nature. The students that didn't like this method seem to be those who were still struggling with fundamental concepts. This theory was substantiated when a progressive assignment was used in an earlier course that deals with building fundamental OOP concepts. In this course, the favorable student opinions dropped to 67%.

Conclusions

This process turned out to be an effective way to help the students learn abstract material that is difficult to learn. The students that took this course struggled in their first programming course and by the end of this course they had created a program with thousands of lines and incorporated encapsulation, polymorphism, inheritance, I/O, for both GUI and command-line programs. These students predominantly received low B's and C's in their first programming course and predominantly received A's and B's in this course (Whittington, Bills, & Hill, 2003).

Based on student comments and faculty observations, the students felt that this method for programming assignments was more effective in helping them learn than using different weekly assignments. They learned all the material and performed a high level on the assignments and exams. However, the evidence seems to indicate that this method should not be used too early in the curriculum. The students should be fairly comfortable programming and have a solid knowledge of fundamental programming concepts before using this method.

The students seemed to have a better appreciation for the benefits of a well designed program. This was most likely a direct result of the frustration they felt while attempting to maintain their poorly designed code.

One unexpected benefit of this technique was the effect it had on the lectures. The students' questions increased as the term went on. The majority of the questions related to the programming

assignments. Some of questions were specific problems they were having in the current assignment, but a lot of questions were asked while the new material was delivered. They seemed to be paying better attention while trying to understand how the current topic was going to enhance and be incorporated into the next assignment. In previous courses, the programming assignments tended to be ignored by the students until after the material was delivered.

Using this technique, students can be pushed harder because the core of the program already exists and they have more time to deal with the difficult topics. The assignment requirements can be tailored to the ability of the current collection of students. If they struggle with a particular topic, more requirements can be added on that subject in the next assignment. They also get another chance to complete the material they didn't get correct in the previous assignment.

This approach helped students learn as they continually added code to their program. They used their current constructed knowledge, experienced difficulties with their current approach, discovered a better technique to solve the problem, and reformed their knowledge base.

References

Carbone, A., Hurst, J. Mitchell, I. & Gunstone, D. (2001). Characteristics of programming exercises that lead to poor learning tendencies: Part II. *ITiCSE*. Canterbury, UK: ACM Press

Hadjerrouit, S. A. (1999). Constructivist approach to object-oriented design and programming. *ITiCSE*. Cracow, Poland. ACM Press.

Whittington K., & Bills, D. (2004). Success with alternative pacing. *Proceedings of the 5th Conference on Information Technology Education*. ACM Press.

Whittington K., Bills, D., Hill, L. (2003). Implementation of alternative pacing in an introductory programming sequence. *Proceedings of the 4th Conference on Information Technology Curriculum*. ACM Press.

Biography

Keith J. Whittington is an Assistant Professor in the Information Technology Department at the Rochester Institute of Technology. His teaching area currently focuses on programming but has also taught in the multimedia, HCI, and networking areas. He spent over 20 years in industry as a computer programmer/systems analyst. His current research interest is teaching programming using active learning techniques, and is currently the PI for an NSF grant entitled "Active Learning for Programming in Information Technology".