A Project Designed to Assess Overall Programming Skills

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

kjw@it.rit.edu

Abstract

This paper examines the process and outcomes of a final project given to students at the end of their third course in Java programming. The purpose of the project was to assess the students' knowledge of program design and the various Object-Oriented Programming (OOP) concepts covered throughout the three-course sequence. This project required the students to work in teams of two then interact with all the other teams in the class. A communication protocol was specified to ensure that each team's program could work with every other team's program. At the end of the course each team had to present their results and experiences to the class. Every team's response was positive, but the surprising result was the overwhelming response from the students extolling the benefits and strengths of preliminary design, following established protocols, the use of core OOP concepts, and how it helped them create good, stable programs. This project surpassed our expectations and this paper discusses the details of the assignment, the student's progress throughout the project, and the final results.

Keywords: OOP, programming, java, teamwork, assessment

Introduction

We have a three-course Java programming sequence that is required of all IT students. Students typically take these courses in their freshman year and have little or no prior programming experience. This paper discusses the final assignment that was given to the first group of students to take the third course of the sequence.

We needed a final experience that integrated everything they had learned. We also wanted something that would make them do a preliminary design and adhere to basic OOP techniques in order to have a high probability of success. It was also important to us to make this final assessment a learning experience.

One of the problems with beginning programming students is that they tend to disregard the importance of preliminary design. Even if they are forced to present a design, they will usually create a design after they finish coding their program. One reason for this attitude stems from the fact that they are not creating multi-thousand line programs that are shared amongst other programmers and integrated into other programs. Perhaps the most exciting result of this project was that we did seem to change the students' attitudes towards design.

A team project was chosen in order to let the students collaborate with each other and exchange ideas. This teaching technique is based on the constructivism theory of learning where students construct new knowledge based on previous learning. Current studies in teaching introductory programming courses tend to recommend a constructivism approach to learning (Ben-Ari 1998; Fleury, 2000; Hadjerrouit, 1999).

Paper Accepted as a Regular Paper

We wanted a realistic application that the students would enjoy so we chose a GUI client/server chat application. The addition of a chat room made them deal with multiple clients in a "real-world" scenario. Since most students are intimately familiar with chat rooms, they could bring their previous experience and knowledge into this project. Protocols were specified for users logging in and out, sending a message to a specific user, and broadcasting messages to all logged-on users. This assignment was spread out over a two-week period and we guided them throughout the process by assigning a specific goal for each class. At the end of the project, each team had to share their experience and demonstrate their solution to the entire class.

Based on the subjective analysis presented in this article, we created a project that demonstrated the benefits of design while engaging the students with an assignment that they enjoyed. We also forced the students to design their classes before they started coding so they could discover the benefits of preliminary design for themselves.

This article discusses the OOP techniques and concepts we expected the students to use, the challenges they faced, their experiences and comments, and finishes with a subjective analysis of the project.

Details

The classes met twice a week and the students had 2 ½ weeks to complete the project. The classrooms were equipped with computers and every student had access to a computer. This worked well for this project since it enabled the students to work on their assignment during class time and ask for help from the professors.

This section describes the expected outcomes of the final project and is broken up into several subsections. Each subsection describes a major component of the project. The required programming skills, OOP concepts, Java classes, and the various problems that students tend to have with this material are discussed.

Client/Server Application

A client/server application requires a large degree of programming skill just to get it working correctly. A client/server application has two difficult concepts: interoperability of programs and I/O classes.

Interoperability

A client/server application requires two programs to communicate with each other. This tends to be challenging from both a code perspective and an abstract level. The main problem lies in the concept that if one program is writing, the other program has to be reading. For example, assume that the client program is trying to send information to the server, if the server is not trying to read this information, the client will freeze. Most students, when faced with this problem, assume their client application has crashed.

I/O Classes

Java uses I/O classes for client/server communication. I/O classes are especially challenging because they use an intimidating hierarchy of inherited classes. These classes allow a user to choose a particular data format that best suits their need. However, because there are so many choices, many beginning programmers find it difficult to know which class to choose. Although we spent a lot of time on I/O classes, most students never really understood how to use them and were dismayed at the though of having to revisit these classes.

Another confusing issue is that I/O classes are split into two main hierarchies: character streams and byte streams. Beginning programmers tend to struggle with the concept of multiple data formats. Many

students never seem to understand the difference between a binary number and a character that looks like a number. They always want to know why a number isn't just a number.

Teams

In addition to the pedagogic benefits for having the students work in teams, we also felt this was important since these students were freshmen and may not have worked on teams before. We limited the teams to two members in order to let one member write the client application and the other one write the server application. This made them communicate with each other while independently developing their own code.

Protocol

We also wanted to simulate a real application where interacting classes have to follow a set protocol. Just writing a program that works independently of any other class is not always good enough. This is especially true in a client/server situation where there may be several versions of the client application. We specified protocols for login messages (with password), messages to a specific user, and multicast messages. Using protocols also helps avoid the problem of creating code that only works through mutual kludges. So once each team had their applications working we had them connect to the other teams' programs.

Message Handling

In order to process the messages, students had to strip out the various pieces of the message. The easiest way to do this was to use the various String manipulation classes supplied in Java rather than trying to decipher the message character-by-character. We purposely chose to separate the individual fields with a colon because it is not one of the default separator characters used by the supplied java classes. To make this work, they would have to consult the java documentation and use a constructor that lets them change the default separator characters.

Design

A well thought out, preliminary design is important for any program, however, we were aware that students tend to feel that design is a waste of time. We knew if we didn't force them to design their program first, they would magically come up with a design after they wrote their code. So on the first day of the project, the teams were given the project description and told to design the program. They also had to show their design to their instructor before leaving class.

We taught the design technique of using CRC (class-responsibility-collaborators) cards and identifying the nouns and verbs in a program description to help identify classes, attributes, and methods (Gilbert & McCarty, 1998; Weisfeld, 2000).

Testing

Thoroughly testing a program is another important but difficult concept to convince students of its importance. We designed this project to let them experience the art of debugging and testing in several different ways. First they had to get their own programs communicating with each other, then they had to get the other students' programs interacting correctly with their programs We were hoping that this experience would convince them that thoroughly testing their code is very important especially if they want "error free" programs.

Presentation

Each team had to give a presentation on their project for several reasons. First of all, we wanted them to share their experience with their classmates and explain their unique design. Additionally, since one of the best ways to learn something is to teach it, we felt that explaining their designs was important to their learning experience. We also wanted the students to see that a program description can yield multiple solutions.

Public speaking is an important skill to bring to the workplace. It is especially important to make technically oriented students speak since they tend to disregard verbal skills. For these reasons, we made each person orally participate in the final presentation.

Lectures

To reinforce our objectives we started each class with a lecture that focused on various topics relating to the project. These topics included: design, project management, testing techniques, and presentation skills.

The Assignment

This section contains the actual assignment given to the students.

Objectives

To build a multi-user network chat server.

To build a matching chat client.

Teams

Each team will have two members. One member is responsible for the client side and the other member is responsible for the server side.

Server Details

When the server starts up, it should obtain the port number to listen to from the command-line, and it should allow multiple simultaneous logons by multiple clients from multiple remote machines.

The server should have a logon manager that is responsible for: logging new users on, logging users off, and managing the redistribution of messages to all users. This component should enforce the fact that all user names must be unique.

The server should have a user manager for each logged on user which is responsible for: channeling incoming and outgoing communications to/from that user and for notifying the rest of the system when that user needs to be removed.

The server should have an input manager for each logged on user, which is responsible for: handling and forwarding (as appropriate) incoming messages from that user.

The server should have an output manager for each logged on user, which is responsible for handling and forwarding (as appropriate) outgoing messages to that user.

The server should be able to operate with any other group's client. In order to do that, it must follow the communication protocol described below.

Client Details

When the client starts up, it should obtain the machine to connect to, the port to use, and the desired user-name from a client-side GUI component.

When the client tries to connect, it should display meaningful error messages should there be problems connecting.

The client should have a GUI component that allows the user to type in a message, and sends the message when the user hits enter or presses a button (or both).

The client should have a GUI component for displaying ("logging") both incoming and outgoing messages.

The client should not display the logging-on messages until the server has echoed the messages back to the client.

After logging on, the client should display a GUI component that will allow the user to log off without closing down the client.

After logging off, the client should be capable of logging on to a new session (possibly at a new server and/or port)

The client should be able to operate with any other group's server. In order to do that, it must follow the communication protocol described below.

Communication Protocol

The following table shows the various commands transmitted to the server and the possible response(s). Both the client and server need to follow this protocol.

Message To Server	Response(s)
LOGIN: username	ACCEPTED: user logged in ERROR: User name in use
LOGOUT: username	ACCEPTED: user logged out ERROR: unknown user ERROR: cannot log out other user
MESSAGE: messagetext	CHAT: username: messagetext (see note 1)

CHAT messages will appear at the client even if a MESSAGE was not sent. This is a natural result of other users placing chat messages onto the server. The server is responsible for creating a CHAT message from the MESSAGE data and sending it out to all logged in users.

The uppercase words are exact keywords that must be exchanged

The lowercase words change based on the information exchanged

Design Requirements

You are expected to have a completed design prior to doing any coding. For the purposes of this project and this course, that means you will create a set of CRC cards to define both parts of the project. (If you

are familiar with other design methods like UML (Unified Modeling Language), you should check with your instructor before using them to design the project.) These cards are part of your deliverables. Any additional design materials that you feel are appropriate can be submitted.

Additional Requirements

You should be able to use your client with the server created by the team next to you (or by another randomly-selected team's server). Another team should be able to use their client with your server.

Deliverables

Your team must present your design to the instructor and must be able to defend the choices that your team made. Plan to turn over your design materials, such as CRC cards, at this time.

Your team must provide both the Java source code and compiled class files for a working server and a working client.

Your team must ensure that both your client and your server meet the project requirements stated earlier in this document and can inter-operate with some other team's client and server.

Your code must be fully commented, and JavaDoc output files must be provided to the instructor along with your code. (See second bullet above.)

Schedule

The following schedule should be followed in order to complete the project on time:

Day	Activities
1	Decide on teams, design the client and server
2	Coding for the client and server, some unit testing
3	Complete coding, finish unit testing, integration testing
4	Complete testing including cross testing between groups, complete documentation, plan presentation
5	Presentation to the class

Presentation

In place of the final exam for this course, you will have to make a short presentation about the final project you worked on during the last two weeks of the course. This will be very similar to what many companies do at the end of a project.

Most projects end with a post-mortem of the project. The various steps of the project are reviewed and a set of the lessons that were learned over the course of the project is produced. The hope is that the project members will learn those lessons and not repeat those mistakes in the future.

Your presentation must address all of the following topics to get full credit:

Discuss your design by showing a diagram of the various classes. Be prepared to discuss how the classes work together. Talk about any specific issues you encountered in getting the client and server to work together.

Discuss any issues encountered during coding and testing. How effective was your testing? Did errors surface after you had tested? How did you create test cases?

Discuss what worked well and what did not work well when creating these programs. How was the work divided up? What problems with the design, coding, or testing did you see occurring on the project? What would you do again and what would you avoid doing?

Summarize what you learned on this project.

The Project Experience

This section describes what happened throughout the project assignment. The experience is broken down into a day-by-day description.

Day 1

The lecture given this day was a review of the design techniques that they were taught earlier in the programming sequence. We used the CRC card technique described by Gilbert & McCarthy (1998) and Weisfield (2000).

After the lecture, the students were given the problem description (shown above) and their team assignments. Each team was comprised of 2 people. To promote team cooperation, we let the students choose whom they wanted to work with and we assigned the remaining teams.

Next, they broke into teams and started to design their programs. They were told to identify all their classes, attributes and methods. Once they had completed their CRC cards they had to show their design to the instructor and defend their design decisions.

Once their design was approved, they were directed to decide which member would write the server and which member would write the client application. They were also directed to come to the next class with both applications working.

Observations

I had 18 teams in my class and it was interesting to see the design variability. Obviously, everyone came up with at least two classes (a client class and a server class) but most groups defined multiple classes. Because this was a fairly complex program, I was looking for the teams that only defined two classes. In particular, I wanted the opportunity to discuss the repercussions of choosing a design with classes that had too much responsibility. I tried to convince these teams how difficult it would be when they tried to debug their programs.

The prevalent reason that the students used to justify using only two classes was that they perceived that the more classes you made, the more difficult it was to create a program. I told them that the exact opposite is true--one of the main advantages of OOP languages is the separation of responsibility. It allows you to concentrate on a smaller segment of the program while ignoring the rest of the problem. Most of the groups listened and redesigned their project to include more classes. One group insisted that creating multiple classes only makes programming more difficult. Since they were particularly adamant about their decision, I let them keep their design and watched them slowly loose control.

Day 2:

This day's lecture topic was planning a project. We talked about breaking the project into pieces, assigning a person to each task, and developing a schedule with established deadlines. We also showed them examples of various types of schedules including Pert and Gantt charts.

Today's goal was for each group to get their client and server applications communicating with each other. Once a group demonstrated that their programs worked together, they were instructed to start up several of their client programs and see if their server application still worked. Once they had multiple client programs working, they were told to purposely send messages that violated the protocol. They were told that their program should never crash, and had to catch every error and keep running.

Observations

Several common problems emerged from this day's activity.

Most groups had the problem where their server and client programs froze. The problem was the lack of communication synchronization—when one program was writing, the other program wasn't reading. A lot of groups had a difficult time with this and I guided them through the process of printing messages inside their code so they could see exactly where they were in both programs.

Some groups had difficulty just getting the two programs connected. I helped these groups go through the proper sequence of events to get a client/server application working. These same groups usually had problems choosing the proper IO classes to use and had difficulty creating sockets.

Every group struggled with the interpretation of messages. Various problems emerged when they tried to enforce the protocol including: spaces, case sensitivity, and punctuation. Interestingly, every group attempted to parse the messages one character at a time even though they were shown several string manipulation classes. Some of the groups had attempted to use the string classes but couldn't get them to work. The main problem is that the default separators used in these classes do not include the colon character. It never occurred to them to look at the documentation and see if they could modify the behavior of these classes. I told these groups to look at the constructors for these classes and redefine the separator characters.

Some of the groups had problems getting multiple clients working with their server. These were the groups that tried to avoid using threads. But no matter how desperately they tried, they eventually realized that they had no choice but to use threads.

Overall, the students were working hard at solving all of their problems. It was apparent that the students were learning a lot and several groups demonstrated the enthusiasm that programmers get when presented with an interesting challenge.

Day 3:

This day's lecture dealt with testing techniques. We discussed the importance of catching errors early, how early error detection dramatically decreases program development time, the difference between unit testing and integration testing, and the benefits of a test plan.

Today's task was to exchange messages with the other teams. They were instructed to get other team's to communicate with their server, and to connect to other team's servers. The purpose for this activity was to get the teams talking to each other, experiencing errors, and working together to find the source of the errors.

We knew they would experience errors they had never thought about. We also hoped they would realize that just getting a program to work isn't always good enough. Strict adherence to an established protocol and protection against badly formatted data is critical to a well-behaved program.

Observations

At this point most groups had their own client/server application working, but once they connected to other people's programs their programs crashed. The major problem was the various interpretations that

each group had with the protocol. Several groups now complained that the specified protocol was neither strict enough nor explicit enough.

All of the teams were actively engaged trying to debug each other's code while trying to decide where the problems existed. Even the teams that had correctly interpreted the protocol now had to deal with data formatted in ways they had never expected. I was enjoying watching the teams work together while honing their troubleshooting skills.

A couple of the teams fell apart on this day. One was the team that used a two-class design. They found out that they could no longer support their design. The ever-increasing errors and complexity of the program finally got to them. They spoke to me and decided that in spite of their preconceptions that using multiple classes was bad, they now realized that using multiple, narrowly-focused classes was a good thing. So after class they completely reorganized their programs.

Another group had never understood the benefits of data encapsulation so they declared all of their attributes public. Their theory was to make the coding easier by letting any methods access any data from anywhere. They also reasoned that another benefit was that they could avoid using accessors and mutators. By this day, however, their code was in disarray and they had no idea where errors were occurring. I finally convinced them that they had to go back and make all their attributes private and provide accessors and mutators for them.

To their credit, both problem teams came to the next class with redesigned code and successfully integrated their programs with the other teams. This was certainly a good (albeit painful) lesson for both of the teams.

Day 4:

Today's lecture discussed the important components for an effective presentation. The various topics we discussed were planning, researching, creating an outline, reviewing the materials, limiting words on the slides, arriving early, practicing the presentation, providing handouts, and making eye contact.

The purpose for this day's activity was to increase the scalability of their programs. The teams were told to start their server and write their team number, IP address, and port number of their server on the board. Next, they were told to randomly connect to any (or all) servers in the room and see if they can break each other's programs.

Observations

The teams were now getting dozens of clients all running at the same time with all sorts of commands and messages. Once again, each team had to work with other teams to access and fix the problems that emerged. By the end of the class, one of the teams was so confident that they issued a challenge to the other teams to try and break their code. This challenge inspired another team to send embedded control characters to see if they could handle it.

The students were now testing their code to the depths required to make stable and reliable programs. It was satisfying to watch the process of students creating delicate code that only worked if everything was perfect to creating very robust programs.

Day 5:

Today the students made their presentations. They had to describe and demonstrate their unique solution to the problem and state the reasons for choosing their particular design. They also had to discuss the various changes and modifications they did to their program and discuss the lessons they learned from the project.

Observations

In addition to observing the students' presentation skills, it was interesting to observe the pride they exhibited towards their programs. Even though no dress code was specified, several students came dressed in ties. I found it amazing to see unprompted freshmen students wear ties. These students were genuinely proud of their product and couldn't wait to show it off.

Lively discussions erupted during the presentations. The students asked a lot of questions and wanted to know how the other teams dealt with various issues and why they chose their particular design.

Student Comments

This was an extremely successful project based on both the instructors' and students' expressed feelings and observations. But most surprising of all, were the comments that the teams made during their presentations.

The following sections provide a list of actual student comments and are grouped into various categories. These comments were extracted from the PowerPoint slides that the students presented at the end of the course. The comments were unsolicited and seemed to be based on their own experiences.

Lessons learned

Develop skills in network communication, working and integrating with other programmers

Planning helps a lot, I/O streams, and GUIs are versatile

Need to take the time to plan

Have to be in right frame of mind, trial and error doesn't always work, improve team-working skills, and need to try different things (don't stick to one idea)

How to add scrollbars with changing text components

As good as you think something is, when it is actually used something will go wrong,

Use more methods, how to code as a team, and communication is key

Learned difference between the equals test versus the compareTo() method for objects

What Worked Well

CRC cards and integration testing

What went right: initial planning and forethought preempted most problems

Static synchronized methods, ArrayList of objects, and threads

Using a key listener for the JTextArea to send messages after the user pressed the enter key, scrollable text areas, mnemonics, and key accelerators

Enumerations, Vectors, and CRC cards

Learned how to work as a team, CRC cards, debug and test to make all programs work well.

After designing the application, the coding was relatively easy because the program was carefully designed with all of the classes and most of the methods already identified

I went through all of my scenarios and added the classes needed by each one

One of the most helpful methods was adding print statements to find exactly where the client was hanging

What Went Wrong

Specifications should have been more specific, merging code was difficult

Banging out code on day 1, large chunks of code, and choosing a bad design for project

Lack of communication, no scheduling, procrastination

Early coordination would have saved a lot of effort, trying to keep the design limited to two classes wasted a lot of time and effort

Creating the server as a single class – confusing, didn't take advantage of object separation

Creating the client written with only inner-classes

Couldn't get a list of all users due to lack of a set protocol to receive the users logged in

Summary

Challenging and took lots of time and thought

Standards MUST be followed, planning is necessary for programming, testing and debugging is time consuming

This project was an accurate representation of what we have been working towards all year. We used all of the methods we have learned up to this point to create a program that at the beginning of the year I thought couldn't have been done.

This project showed us how real life business works in a small project

By working in a group, I learned something about other people's coding styles and how much more planning is required when you need two programs written by two different people to work together.

This was a very practical project, in that we may do something like this in our jobs.

This is just a sampling of the comments, but it is representative of all the comments we received. It was satisfying to hear all the positive comments, the in-depth lessons they learned, and the general consensus of the importance of preliminary design, teamwork, testing, and adherence to established protocols.

Conclusion

Although this was a difficult project, every group successfully created properly working programs. Some of the reasons that probably contributed to the success of this project were the many hours the students spent in-class with their instructors and collectively working on their code with their fellow students. We had carefully orchestrated the whole process and guided them every step of the way. We combined several learning techniques to help them get through the project: working alone, working with one other student, working with other teams, and they had an expert to help with difficult issues.

The most exciting result of this project was the observed change in attitude of the students. Most students had entered this course with a preconceived attitude against design and a general disregard (or misunderstanding) of OOP concepts and techniques. By the end of this project, the students were obviously excited about the project and seemed to be embracing the importance of preliminary design and OOP concepts.

I believe this project was responsible for the students' attitude metamorphosis, and I advocate its use as a culminating experience in JAVA. Several professors are now using this project as their final experience in the graduate JAVA course. I don't believe the success of the project lies solely in the specific choice of a chat client/server program, but more in the fact that we choose a challenging project, en-

gaged the students, and guided them through the entire process of creating a representative "real-world" program.

In order to keep from assigning the same project quarter after quarter, we are hoping to design additional final projects that will similarly engage and challenge the students. However, we are planning to periodically reuse this project since it was so effective.

References

Ben-Ari, M. (1998). Constructivism in computer science education. *SIGCSE Bulletin*, *30(1)*, 257-261

Fleury, A. E. (2000). Programming in Java: Student-constructed rules. *SIGCSE Bulletin, 32(1)*, 197-201

Gilbert, S. & McCarty, B. (1998). *Object-Oriented design in Java.* Corte Madera, CA: Waite Group Press.

Hadjerrouit, S. (1999). A constructivist approach to object-oriented design and programming. *SIGCSE Bulletin 31(1)*, 171-174

Weisfeld, M. (2000). *The object-oriented thought process.* Indianapolis: Sams Publishing.

Biography

Keith J. Whittington is an Assistant Professor in the Information Technology Department, in the Gallisono College of Computing and Information Sciences, at the Rochester Institute of Technology. His teaching areas currently focus on programming but has also taught in the Multimedia, HCI, and networking areas. He also spent over 20 years in industry as a supervisor, engineer, and computer programmer/systems analyst. His current interests include teaching using active learning techniques, and developing active networks using network processors.