Journal of Information Technology Education: Innovations in Practice **Volume 11, 2012**

Utilizing BlueJ to Teach Polymorphism in an Advanced Object-Oriented Programming Course

Basem Y. Alkazemi
Umm Al Qura University, Makkah, Saudi Arabia
bykazemi@uqu.edu.sa

Grami M. Grami
King Abdulaziz University, Jeddah, Saudi Arabia
ggrami@kau.edu.sa

Executive Summary

Teaching Polymorphism can be best implemented by using a combination of bottom-up and top-down approaches. However, from our observation and students' self-reporting, the former seems to be the predominant in the Saudi context. We try to investigate whether applying a more balanced approach in teaching the comprehensive concept of Polymorphism would be more beneficial in developing learners' effective analytical skills. In this project, we applied the BlueJ IDE to address the ambiguity in expressing Polymorphism and to compensate for shortcomings resulting from the exclusive use of common programming editors such as Eclipse. We observed that students who were taught using BlueJ IDE did considerably better in tasks that required the producing of flexible and extensible programs than those instructed in Eclipse IDE. We therefore recommend utilizing BlueJ for teaching this design concept.

Keywords: BlueJ IDE, Polymorphism, object-oriented programming, Alternative Approaches.

Introduction

Learning programming techniques requires among other things combining bottom-up and top-down processing in order to fully comprehend the topics under discussion (Bruning, Schraw,& Ronning 1995; Zelle, Mooney, & Konvisser, 1994). This eclectic approach should help develop students' problem-solving ability especially in object-oriented programming (OOP) where system structure reasoning is an essential learning outcome. Building this analytical skill would not be possible without explicitly describing different OOP techniques, some of which may not be adequately expressed without proper visual representation.

In OOP, a system is theoretically composed of different objects interacting with each other through message passing or method invocations. System structure is traditionally represented abstractly at the design stage using different modelling languages such as UML (Fowler, 2003). However, when it comes to programming, the relationships between classes become very implicit as they are tangled into the source code. Basically, certain OOP fundamental principles cannot be taught exclusively using bottom-up thinking, i.e., using traditional source code editors such as Eclipse, JCreator, JDveloper and other Integrated Development Environments (IDE). A visual representation that combines high-level design (top-down approach) with existing low level implementation details (bottom-up approach) is required.

Editor: Elsje Scott

According to Sommerville (2010), two of the fundamental design principles imposed by software engineering are the notions of *loose-coupling* and *dynamic-binding* of objects in a system structure. In this regard, OOP embraces a number of techniques to effectively implement these principles, an important example of which is the model of *Polymorphism* (i.e., *Inclusion Polymorphism*). While learning fundamental OOP techniques, such as inheritance, by programmers is not necessarily insignificant (Mayer, 1989), we nonetheless observed that teaching Polymorphism as a technique to overcome the problem of coupling and facilitate dynamic-binding is one of the most challenging topics. As a result, students may find them hard to comprehend. This difficulty is further complicated as it involves teaching the main concepts at high level of abstraction that is not mapped correctly into implementation details of a system. We believe that expressing the relationships between objects is very important if students are to understand the exact design decisions underpinning the concept of Polymorphism. This is the main problematic area we identified in current methods of teaching Polymorphism to programmers. In fact, it has been observed that even UML cannot fully satisfy teaching needs especially when it comes to precise description of Polymorphism. UML lacks the capability of demonstrating the concept at the implementation level. Current programming editors are not capable of representing dynamic relationships in a form easily understood by programmers. In addition, current editors cannot represent the accurate concept of loose-coupling and dynamic-binding as they require writing additional code into a *Test* class that links corresponding classes together. At first impression, this additional code fulfills one requirement of Polymorphism. On closer inspection, however, the code actually invalidates the fundamental principle of loose-coupling causing the annulment of particular features of Polymorphism.

Figure 1 illustrates a typical example used to teach the concept of Polymorphism. It shows a Test class called *AnimalTypes* that is considered as a client to access one of three servers of different types. It contains few lines coded to link this Test class to the other sub-types (i.e., Cow, Dog, Snake). Although this example exemplifies the general principle of Polymorphism, it actually invalidates expressing the comprehensive utilization of it. One problem in this example is that the class must be compiled in case adjustments are made to any of the sub-classes (e.g., Cow, Dog, Snake) in complete contradiction to the actual purpose of loose-coupling. Moreover, when students are presented with similar examples, they will probably find the concept of dynamic-binding hard to understand, as what they observe is only the static relationships hardcoded into the class.

```
public class AnimalTypes
{
    public static void main(String args[])
    {
        Animal animalType;
        Cow aCow = new Cow("Bossy");
        Dog aDog = new Dog("Rover");
        Snake aSnake = new Snake("Ernie");

        // now reference each as an Animal
        animalType = aCow; animalType.speak();
        animalType = aDog; animalType.speak();
        animalType = aSnake; animalType.speak();
    }
}
```

Figure 1: AnimalTypes Test Class

In order to address these concerns, we need an effective teaching tool capable of exploiting all the potentials of Polymorphism. Having identified the shortcoming of traditional IDE tools and decided on the criteria for choosing an alternative tool, we proposed BlueJ IDE as our tool of choice. More details of BlueJ are provided in a following section.

This paper reports on our experimentation with BlueJ and evaluates its appropriateness as an effective teaching tool capable of demonstrating the fundamental design decisions of Polymorphism. Our judgment is based on its capacity for eliminating unnecessary coupling between classes and facilitating dynamic-binding between objects.

The paper is organized as follows: the following section describes the principle underpinning Polymorphism. This is followed by a representation of BlueJ's in teaching Polymorphism. The following two sections describe the methodological framework and also report on our experimentation including an evaluation of the approach we adopted for teaching Polymorphism. Finally, we summarize the findings and highlight the implications of our study in the conclusion. We also acknowledge the limitations of the current study and provide recommendations for future research at their respective sections.

Polymorphism

According to Rotem-Gal-Oz (2006) and Lahman (2011), a function that accepts a parameter of type T (e.g., *Person*) can still work if passed a parameter of type S (i.e., *Teacher* or *Student* as shown in Figure 2) as long as type S is a sub-type of the type T. This, in fact, is the exact semantic of Polymorphism. Polymorphism is a feature mostly used in object-oriented design and programming that is mainly associated with inheritance. Gamma, Helm, Johnson, and Vlissides (1995) also mention it can be generalized to cover a wide range of applications that involve multiple parties such as a *client-server* pattern that is a one of the standard patterns in distributed systems design principles.

For example, a client can be dynamically bound to a server at run-time only if a parameter passed to the client matches a corresponding sub-type. The notion of Polymorphism provides several benefits to systems structure, one of which is the elimination of unnecessary coupling between classes. So, a change in one part of an application will not affect other parts. For example, Polymorphism can be utilized to build a client-server based system where client code is not tightly linked to server code at run-time, thus implying loose-coupling. Figure 2 illustrates an example of Polymorphism generated using the BlueJ IDE that is similar to the UML class diagram.

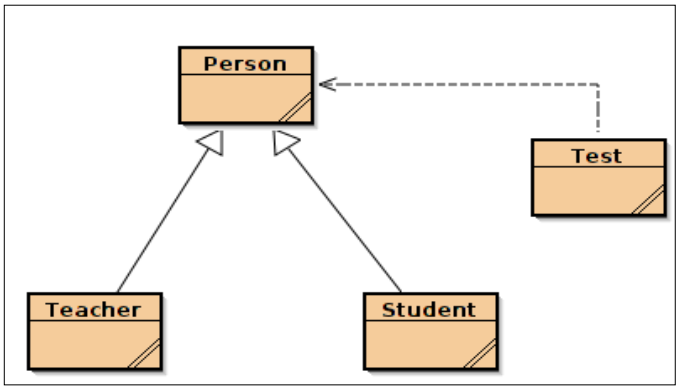


Figure 2: Example of Polymorphism

As the above example in Figure 2 shows, the *Test* class (i.e., type) is not aware of the *Teacher* or *Student* sub-types. It is only hard-coded to use the *Person* type regardless of the sub-types available. At run-time, programmers deal with o*bjects* that are instances of the classes they have written at compile-time. Based on the type of parameter passed by the *Test* object at run-time, the *Test* object can be dynamically-bound to either *Teacher* object, *Student* object, or simply use the *Person* object directly. The *Test* class assumes that the *Person* class implements a method that it will

access to obtain the necessary functionality. Through inheritance, all sub-classes of the *Person* class are guaranteed to have the required method implemented in them either as inherited ones or overridden. So, the *Person* class (usually defined as an abstract class) can be considered as a contract that if a class is inherited from, it can be guaranteed to fit in this application without any compilation problems. Figure 3 depicts the source code of the *Test* class. A discussion about how the Polymorphism concept can be fully explained to students is given in the next section where this example is executed in the BlueJ IDE showing how dynamic-binding can be achieved without invalidating the loose-coupling concept.

```
public class Test
{

    public void run(Person person)
    {
        person.mySkills();


    }


}
```

Figure 3: Test Class Source Code

The method "`mySkills()`" is inherited from the *Person* Class by both *Student* and *Teacher* sub-classes. It can be overridden by sub-classes in order to customize its functionality as needed.

BlueJ and Teaching Polymorphism

BlueJ is a programming environment designed to improve teaching introductory programming using Java language first developed and implemented by Michael Kölling and John Rosenberg (1996) (Van Haaster & Hagan, 2004). Barnes and Kölling (2011) further explain BlueJ is an IDE specifically designed to simplify different object-oriented programming capabilities particularly in Java.

Kouznestova (2007) mentions that BlueJ is a free and easy to use tool that can be utilized to provide visual representation of small-to-medium system structure. The majority of students, according to Van Haaster and Hagan (2004), reported positive attitudes towards BlueJ either from first impressions during early stages of the programming course or in the final stages. In fact, the percentage of students passing the programming course with the help of BlueJ has increased noticeably.

In Table 1 are summaries of previous studies involving Polymorphism and BlueJ. The spectrum of these studies covers a large range of research including accounting for popular tools in teaching programming languages to computer science students (Bergin, 2003; Garner, 2003; Raadt, Watson & Toleman, 2002), the merits of polymorphism in OOP (Donchev & Torodova, 2008; Purewal & Bennett, 2006; Van Haaster & Hagan, 2004), improving the practices of teaching polymorphism (Bergin, 2003; Ross, 2005), and the benefits of adopting BlueJ in programming courses (Kouznestova, 2007; Laakso, Malmi, Korhonen, Rajaala, Kaila, & Salakoski, 2008; Lee, Pradhan, & Dalgrano, 2008; Rajaala, Laakso, Kaila, & Salakoski, 2008).

Table 1. The position of the current study in relation to earlier research

Related Studies (in chronological order)	Primary Source of Data	Important Findings and Arguments
Raadt et al.,(2002)	A Census of Introductory Programming Courses in Australian Universities	- What and how programming languages are taught. - The percentage of students who were taught using BlueJ in practical session (4%)
Bergin (2003)	Technical Report	- Using elementary patterns to teach polymorphism at early stages.
Garner (2003)	Literature Review	- Discussing tools and resources in the context of software lifecycle.
Donchev and Torodova (2008)*	Literature Review	- Mastering polymorphism helps code organization and readability - Polymorphism helps expandability - Classification of polymorphism is a prerequisite for successful learning
Van Haaster and Hagan (2004)	Literature Review and Evaluation Study: Surveys (n = 115)	- The effectiveness of BlueJ in OOP for teaching principles and practice to first year programming students. - Literature helps the selection process NOT evaluating. - The three categories of evaluation: usability, paradigm support, and teaching and learning support.
Ross (2005)	Literature Review	- OOP three requisites: encapsulation, inheritance and polymorphism - Different categories of polymorphism - Clearer definition of polymorphism is required - Coverage of polymorphism in various C++ and VB textbooks - Textbook selection criteria for teaching polymorphism
Purewal and Bennett (2006)	2 second-semester student projects	- Projects that use polymorphism in gaming are unlimited and genuine opportunities for students to freely express their creativity. - Polymorphism improve students' motivation
Kouznestova (2007)	A sequence of four assignments	- BlueJ environment helps novice students developing working models. - BlueJ helps students develop better understanding of OOP. - BlueJ minimizes graphic programming difficulties.
Lee et al., (2008)	Case Study	- The effects of screencast-based and BlueJ-aided instruction - Difficulties in the transition from graphic tools to writing codes. - Alternative approaches in teaching OOP. - BlueJ as a cognitive tool representing learners' schemata. - BlueJ is more useful at high level of abstraction.
Laakso et al., (2008)	Literature review and experiments	- Using ViLLE in teaching program debugging (See Laakso et al., 2008 below) - BlueJ in program visualization
Rajaala et al., (2008)	Literature Review and Experimental Research	- Development of ViLLE (Visualization program) - BlueJ is a static program visualization tool - Program visualization enhance students' learning
The Current Study	Literature Review and Experimental Study (n = 30)	- investigating shortcomings in IDEs used in teaching - Identifying shortcomings in teaching Polymorphism using traditional programming editors - Evaluating BlueJ as an alternative, top-down approach in teaching OOP under GUI that demonstrates the full design considerations of Polymorphism

The main relationships that BlueJ defines are the "is-a" and "has-a" relationships, representing inheritance and association between classes respectively. Figure 2 illustrates the two types of relationships in BlueJ. The "is-a" relationship links the *Teacher* and *Student* sub-classes to the *Person* super class while the "has-a" relationship links the *Test* class to the *Person* class. It is apparent

that there are no interdependencies between the *Test* class and the *Teacher* or *Student* sub-classes. However, it can be linked at run time as illustrated in Figure 4.

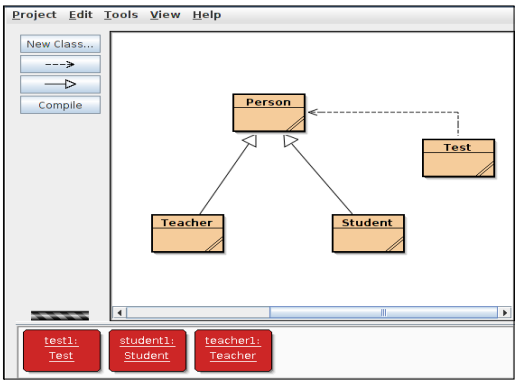


Figure 4: Polymorphism representation using BlueJ

The red boxes at the bottom of Figure 4 are the objects generated after instantiating the three classes *Test*, *Student*, and *Teacher*. The generated objects can be manipulated at run-time by selecting the set of public methods displayed upon right-clicking an object. Any method can then be invoked if it passes the required parameter that can be either a primitive type or a defined object by selecting one from the generated list. Passing a *Student* or *Teacher* object to the *Test* object results in a dynamic-binding of these objects and, thus, executing the corresponding method of the selected class. Therefore, when a *Student* object is passed as a parameter to the method "`public void run(Person person)`" the method "`public void mySkills()`" associated with Student object is going to be executed. Figure 5 illustrates binding the *Test* object to the *Student* object at run-time.

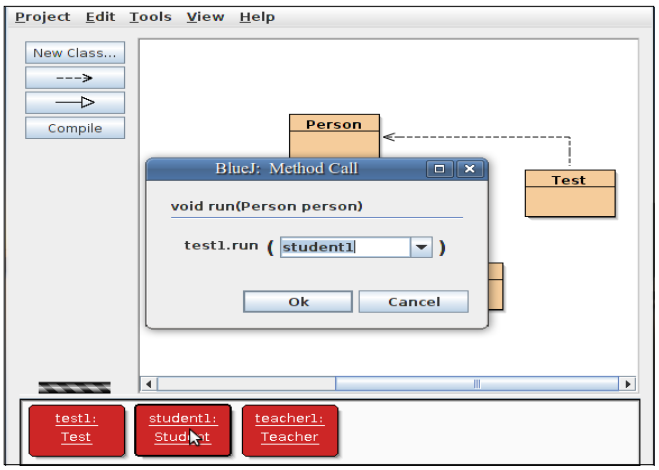


Figure 5: Binding Student Object to Test Object

BlueJ can facilitate expressing Polymorphism in a practical way as any modifications to the *Student* or *Teacher* classes (i.e., servers) would not affect the *Test* class (e.g., client) in terms of recompilation, hence the principle of loose-coupling is maintained and visibly shown to students. Moreover, the notion of dynamic-binding is visually demonstrated to students and put into practice throughout this example when the *Test* class is dynamically-bound to either *Student* or *Teacher* upon the type of parameter passed to it. Another added feature provided by the BlueJ which helps to improve students' learning curve is the ability to generate the source code auto-

matically, having ensured the current system structure and design is validated. A Unit Test class can be used to capture the actions executed by the programmer on objects at run time, and consequently generates the corresponding source code that reflects these actions. Figure 6 shows a generated unit-test class called *AutoGenCode* which traces all the actions applied by a programmer.

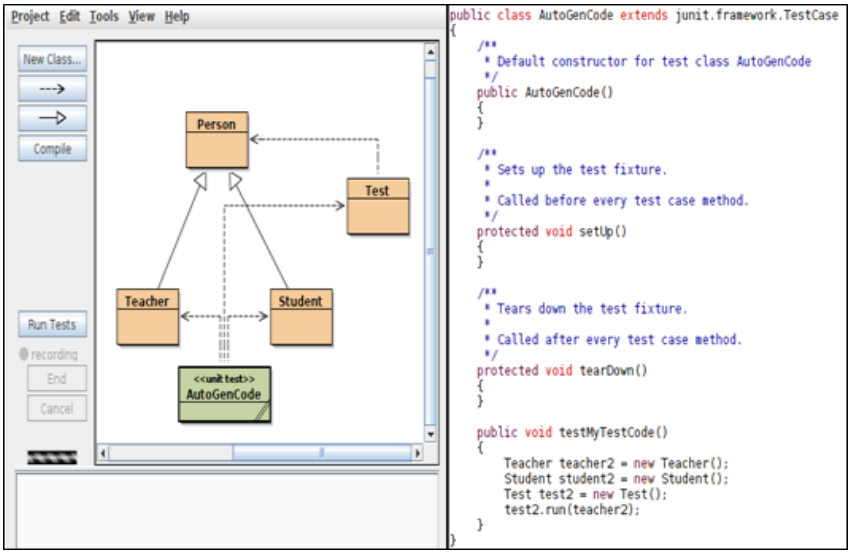


Figure 6: "AutoGenCode" Unit_Test Class

The method "`public void testMyTestCode()`" contains the source code that defines the instantiated objects, how they are bound together, and the sequence of their execution. This method is generated once the programmer stops recording programmer's actions. In other words, whatever action the programmer performs will be reflected in the source-code of the Unit_Test class. This additional feature in BlueJ illustrates how it can be used to teach Polymorphism in a top-down manner as learners become familiarized with the principle of dynamic-binding before moving on to the complexity of source-code writing.

Methodology

Kölling and Rosenberg (1996) insist that the notion of Polymorphism can be understood much more easily if the correct tools are used in teaching, one of which is BlueJ that was developed specifically for teaching purposes (Brunning, Schraw & Ronning, 2005). With this in mind, we were trying to evaluate if utilizing BlueJ as a teaching tool can expose the concepts of loose-coupling and dynamic-binding. We, therefore, opted for a comparative experimental study in the form of experiment and control groups. The notion was that this should help us test our hypothesis about the effectiveness of BlueJ IDE in comparison to traditional IDE tools like Eclipse for instance.

Research Population and Sampling

A 'cluster sampling' procedure was followed in this project which Walliman (2001) describes as cases forming clusters by sharing one or more characteristics, the samples are otherwise homogenous. The only differing characteristic is the different treatment students received in the form of BlueJ versus Eclipse. They share the same background otherwise. Other types of sampling like systematic, simple and proportional stratified were disregarded because they were not applicable for the intended research population.

In more precise terms, two groups of undergraduate computer science students who met a set of criteria were selected for the experiment. They were group 1 (experiment) and group 2 (control) consisting of 15 students each. The two groups had different class schedules, group 1 had OOP class on Monday and group 2 had the class on Wednesday. We tried to account for different factors apart from teaching methods, so the criteria of students' inclusion in both groups were as follow:

- Only computer science students were eligible for inclusion in the study as there were a number of computer engineering majors attending the same classes.
- Only students with a GPA (Grade Point Average) of 3.0 or higher (out of 4.0) were selected. The attempt is to evaluate the tool in terms of teaching the design decisions underlying polymorphism (i.e., loose-coupling and dynamic-binding) not how it correlates to the students' potential.
- Students who obtained a grade of at least 'B+' in the *Structured Programming* course were eligible.
- Students who obtained 25+ out of 30 in the *Java classes and inheritance* short exam previously given, were selected.

The first criterion was designed to ensure students share similar educational background while the other three criteria collectively were drafted to ensure that all participants in this experiment had similar levels of competency. The argument was that this procedure should help minimize the impact of factors other than different teaching approaches on students' subsequent performances.

Procedures and Data Collection

The concept of Polymorphism was taught to group 1 using BlueJ while Eclipse in association with UML like diagrams was used for the other group. Both groups were taught that the principles of loose-coupling and dynamic-binding are part of the design principles underpinning Polymorphism. The topic of Polymorphism was covered in two lectures for both groups in addition to revision exercises. A short quiz was given at the end of the session to both groups simultaneously. The quiz contained two questions (see Appendix). The first requires a general description of the concept of Polymorphism and how it can benefit programming. The second question requires the use of Polymorphism to write a short program in Java that retrieves the details (e.g., title, author name, ISBN) of five different types of books in a library. These are Computer Science, Art, Math, Physics, and Linguistics. The last question evaluates students' understanding of polymorphism. Different types of books requires a demonstration of students' real understanding of the concepts of loose-coupling and dynamic binding as integral parts of polymorphism.

Data was collected from both groups based on their performances in the quiz. The information was processed and analyzed quantitatively using descriptive values such as counts, percentages, and means. The aim was that these descriptive values should facilitate the intended comparison task we set upon when initiating this project, more specifically in order to find out more about the short-term effect of the proposed teaching method.

Results

All the participants from both groups answered question 1 correctly by describing the theory behind Polymorphism as originally described to them in the lectures. Answers to the second question vary. The exam focuses on evaluating students' understanding of three main concepts - all considered constituents of the intended course outcome - as follows:

- System structure: students must recognize the organization of Java classes and establish the relationship between them. The expected answer should include a high-level class

diagram that illustrates the different classes in the system.

- Usage of inheritance: students must understand how inheritance is utilized to build a structure based on generalization-specialization of types. The ideal answer should involve the definition of super class called "Book" and sub-classes that define the different types of books those satisfy the “is-a” relationship.
- Utilization of polymorphism: students must generate a Test class that is completely decoupled from sub-classes. It can only invoke methods in the *Book* super-class (or interface) without any hard-coded link to sub-classes. The binding of *Test* class and sub-classes should be achieved via parameter passing only at runtime. All students were asked to demonstrate their solutions in a computer lab to exhibit their understanding.

Table 2 summarizes the main outcomes of the quiz and displays students’ achievements in each category investigated.

Table 2. Test Outcome of Control and Experiment Groups

Outcome (understanding of)	Group 1 (%)	Group 2 (%)
System structure	100	100
Usage of inheritance	100	87
Utilization of Polymorphism	100	13

Discussion and Implications

The most remarkable finding of the study is that students in group 1 have performed extremely well by scoring the maximum number of points in all three test areas. As far as system structure is concerned, there was no immediately observed difference in students’ responses since all students achieved the highest possible mark regardless of their group. We, therefore, infer that this particular area has no association with different types of treatment, i.e., traditional methods against the proposed BlueJ IDE alternative method.

However, the two other categories of the test show more contrasting results. First of all, participants in group 1, as noted earlier, performed extremely well in every area including the usage of inheritance category and duly received the full mark. Students in group 2, on the other hand, did fairly well as the majority of them (87%) answered the question correctly, not as well as their counterparts but good nevertheless. Despite their good performance, there is still a considerable gap to be reckoned with when the results of the two groups are placed next to each other. We of course cannot rule out the impact of factors other than different treatments in this result despite our attempts to control their impact. However, in this small-scale study, we can attribute the difference in the results to the application of BlueJ IDE in our alternative approach.

The last category “utilization of Polymorphism” yields the most disproportionate result where group 1 considerably outperforms the other group rendering our alternative approach very effective indeed. Although both groups have produced fully functioning programs capable of retrieving data as requested, the first group’s code tends to be more standardized in terms of flexibility and extensibility than that of the second group whose members mostly hardcoded invocations into the client’s code. The first group stopped at the level of defining the composition relationship between the *Client* and the *Book* super-class showing their awareness of potential data retrieval at run time based on the parameters passed to the superclass, as demonstrated during the practical session. The second group, however, provided a fully working C*lient* that can retrieve only the specified types of books. As a result, whenever a new book is added then the client code has to be modified to accommodate this new requirement. In terms of client-server pattern, the operation

requires a total shutdown to the server and recompile to the client code in order to proceed with new changes. As a result, we would argue that the code designed by students in group 1 greatly benefited from the application of BlueJ by showing the defining characteristics. This cannot be achieved by means of Eclipse for an IDE alone as shown by the code designed by the other group.

We believe the type of treatment students received has a significant impact on developing certain skills, especially ones which require analytical thinking and comprehensive perception. BlueJ in our experiment has proven to be especially useful when students had to envisage different components of a system structure at the same time.

Conclusion

Taking all the results into consideration, we would recommend teachers to devote more time teaching students by means beyond technical terms and formulas and in a way which would enhance their wider perception of the given tasks. As is the case with any new technique in the classroom, we anticipate some challenges particularly in the early implementation stage but these should phase out when the potential gains become obvious. Although other practical issues, like the time limit and teacher training programs, have to be taken into consideration, these technicalities, in our opinion, should be addressed by the educational authorities in the institutions.

Teaching Polymorphism is by no means a trivial task in OOP especially when teaching is carried out from a design decisions perspective of software engineering where Polymorphism is used as the driving vehicle to implement the concepts of loose-coupling and dynamic-binding. However, we noticed that in some contexts Polymorphism is traditionally taught using a regular source code editor or IDE virtually ignoring the considerations of comprehensive design. We, however, utilized BlueJ IDE as a teaching tool in order to exploit the implicit capabilities of Polymorphism that other tools usually fail to recognize.

Our empirical study of teaching certain Polymorphism features produced encouraging results. BlueJ does not only compensate for traditional IDE methods' shortcomings but also acts as an effective teaching tool capable of demonstrating design concepts in a more practical and explicit manner.

Despite the encouraging results, we are aware that our findings are more indicative than conclusive due to factors like the sample size and the spectrum from where the participants were drawn. A more comprehensive study that involves a wider range of learners from various IT backgrounds would further verify the results. Our study in spite of that deals with a dynamic topic and follows rigorous scientific research conventions making it valuable to teachers and learners alike. In fact, the carefully designed experiment procedures and data analysis opportunities make us consider applying BlueJ in our future classes and recommend others to follow suit. The study shows that BlueJ IDE explains the design decisions underlying polymorphism better and more accurately than more traditional coding editors like Eclipse or JCreator.

References

Barnes, D., & Kölling, M. (2011). *Objects first with Java* (5th ed.). Prentice Hall.

Bergin, J. (2003). Teaching Polymorphism with elementary design patterns. *OOPSLA Companion 2003*, 167 – 169.

Bruning, R. H., Schraw, G. J., & Ronning, R. R., (1995). *Cognitive psychology and instruction*. Englewood, NJ: Merrill.

Donchev, I., & Todorova, E. (2008). Polymorphism in the course of object oriented programming - Didactic aspects. *The Sixth International Conference "INTERNET –EDUCATION - SCIENCE"*, Vinnytsia, Ukraine, pp. 100 – 113.

Fowler, M. (2003). *UML distilled: A brief guide to the standard object modeling language.* Boston: Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software.* Boston: Addison-Wesley.

Garner, S. (2003). Learning resources and tools to aid novices learn programming. *Technology Education Joint Conference* (*InSITE)*, Finland, 213 – 222.

Kouznetsova, S. (2007). Using BlueJ and Blackjack to teach object-oriented design concepts in CS1. *Journal of Computing Sciences in Colleges, 22*(4), 49 – 56.

Kölling, M., & Rosenberg, J. (1996). An object-oriented program development environment for the first programming course. *SIGSE Bulletin, 28*(1), 83 – 87.

Laakso, M., Malmi, L., Korhonen, A., Rajaala, T., Kaila, E., & Salakoski, T. (2008). Using rules of variables to enhance novice's debugging work. *Issues in Informing Science and Information Technology, 5*, 281 – 296.

Lahman, H. (2011). *Model-based development applications.* Boston: Addison-Wesley.

Lee, M. J. W., Pradhan, S., & Dalgarno, B. (2008). The effectiveness of screencasts and cognitive tools as scaffolding of novice object-oriented programmers. *Journal of Information Technology, 7*, 61 – 81.

Mayer, R. E. (1989). The psychology of how novices learn computer programming. In E. Soloway & J. C. Spohrer (Eds.), *Studying the novice programmer* (pp. 129– 159). New Jersey: E. Lawrence Erlbaum Associates.

Purewal, T. S., & Bennett, C. (2006). A framework for teaching Polymorphism using game programming. *Journal of Computing Sciences in Colleges. 22*(2), 154 – 161.

Raadt, M., Watson, R., & Toleman, M. (2002). Language trends in introductory programming courses. *Proceedings of the InSITE Conference*, pp 329 – 338. Retrieved from http://proceedings.informingscience.org/IS2002Proceedings/papers/deRaa136Langu.pdf

Rajaala, T., Laakso, M., Kaila, E., & Salakoski, T. (2008). Effectiveness of program visualization: A case study with ViLLE Tool. *Journal of Information Technology Education: Innovations in Practice, 7,* 15–33. Retrieved from http://www.jite.org/documents/Vol7/JITEv7IIP015-032Rajala394.pdf

Ross, J. M. (2005). Polymorphism in decline? *Journal of Computing Sciences in Colleges, 21*(2)*,* 328 – 334.

Rotem-Gal-Oz, A. (2006). *Liskov substitution principle (Design by contract).* JDJ Editors Choice.

Sommerville, I. (2010) *Software engineering.* Boston: Addison Wesley.

Van Haaster, K., & Hagan, D. (2004). Teaching and learning with BlueJ: An evaluation of a pedagogical tool. *Information Science and Information Technology Education Joint Conference*, Rockhampton, Australia, 99 – 110.

Walliman, N. (2001). *Your research project: A step-by-step guide for the first time researcher.* London: SAGE Publications.

Zelle, J. M., Mooney, R. J., & Konvisser, J. B. (1994). Combining top-down and bottom-up techniques in inductive logic programming. *Proceedings of the Eleventh International Workshop on Machine Learning, New Jersey*, 343 – 351.

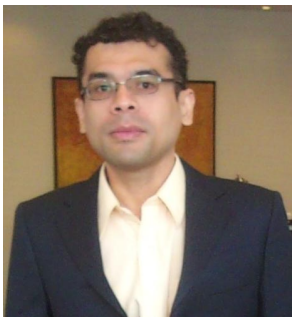
Appendix
Advanced Programming (Quiz # 3)

Q.1\ Describe, using an example, the concept of Polymorphism and discuss its impact on programming practices?

Q.2\Apply the concept of Polymorphism to write a Java program that retrieves five different types of books in the fields of Computer Science, Art, Math, Physics, and Linguistics In terms of their titles, author names, and ISBNs.

Biographies

Dr Basem Alkezemi currently holds the position of vice-dean of Umm Al-Qura University's IT Deanship for E-Government. He received his Bachelor degree in Electric and Computer Engineering in 1999. He then went to study his MSc and PhD in Software Engineering at Newcastle University in the UK which he received in 2004 and 2009 respectively. Dr. Alkazemi has published a number of articles in regional and international journals and participated in many specialised conferences around the world. His main research interests are in software engineering, wireless sensor networks, computer supported education, and e-government.



Grami M Grami graduated in 2001 with a degree in English Language and Literature. He went to Essex University in 2003 to study his Master's degree before embarking on a PhD project at Newcastle University completed by 2010. He currently teaches English and Applied Linguistics at King Abdulaziz University. Grami has published a number of articles in international journals about IT and education and is a current member of the editorial board of Information Technology and Teacher Education Journal.