A Beginning Specification of a Model for Evaluating Learning Outcomes Grounded in Java Programming Courses

Robert Joseph Skovira
Robert Morris University, Moon Township, PA, USA

rjskovira@att.net

Abstract

The essay discusses the development of a way of doing learning outcomes assessment for Java programming courses. It presents a tentative model for doing learning evaluation utilizing a report mechanism. The essay presents the opinion that this report is framed by the explicit and implicit learning outcomes assessment models found in a course's description and objectives and in the Object-Oriented paradigm. The paper discusses the report instrument as a way fulfilling the set of evaluative categories of the models presented by a course syllabus.

Keywords: learning outcomes assessment, learning evaluation, assessment model

Introduction

Learning outcomes assessing has become an increasingly important issue at universities and colleges (Chrysler & Van Auken, 1999; Mitri, 2003; Salegna & Bantham, 2002; Taylor, 2002). Outcomes assessment efforts appear to be framed more often by control and marketing concerns than by educational philosophies. These concerns focus on institutional accountability and program (major or degree) effectiveness and quality for the learning experiences of students (Cunningham & Omolayole, 1998; Nichols & Nichols, 2000; Redmond, 1998). Assessing educational outcomes is about how well programs inculcate the subject matter which programs (in their design) claim ought to be known and doable (even if this seems like an old-fashioned idea because disciplines have lost control over their common body of knowledge). Assessing outcomes is about the verification of these institutional claims and improving the situation (Palomba & Banta, 1999; Salegna & Bantham, 2002; Sims, 1992; Wade, 1999).

While the drive to assess learning outcomes is focused on the institutional and programmatic levels, assessing students' knowing and doing of things must begin at the course level (Dominguez & Ridley, 1999; Redmond, 1998). This can be done with modeling how knowing in the head shows up as knowing and doing in the world (Norman, 1988).

Background

I have had the responsibility for teaching the introductory and advanced courses in Java at both the undergraduate and graduate levels. I also have had the responsibility for developing several addition Java courses, undergraduate and graduate, beyond the introductory level. At the undergraduate and graduate levels, there are

now courses in advanced Java application programming, Java applet and web-based programming, and Java data structures. This is necessary, I believe, because of the immense number of Java classes developed in the various Java APIs (Application Program Interface). Of course, any instruction in or study of the Java programming language necessitates Object-Oriented concepts. In the advanced levels of Java, this means using Object-Oriented analysis and design techniques, together with UML (Unified Modeling Language) to visualize the classes, objects, relationships, and other programming structures diagrammatically.

The Research Question

The evaluative question for every instructor of a programming language such as Java is how well he or she is presenting the course content so that students will be able to analyze programming problems and develop and implement program code using Java and Object-Oriented concepts. It is no longer a matter of merely learning reserved words and syntax. Part of this assessment situation is devising ways of evaluating the process holistically, of determining somehow what mental models (Norman, 1983, 1988) are in use as students work on solving a programming problem. The mental models (for example, primitive data types, control structures, and loop structures, to name only a few) may be specific to the programming language studied, and consists of a language's grammar and syntax. The question is, for a (Java) programming course or any course, how to verify that knowing in the head has happened. Course assessment is about finding out how well students know the grammar (the whys) and syntax (the hows) of a programming language, not simply that they present an executed program (Peterson & Quarstein, 2001).

Purpose of Essay

The reason for this discussion is the description and analysis of a model designed and developed for ascertaining the growth of knowing and doing of Java programming and understanding fundamental Object-Oriented concepts.

A Notion about Assessing Learning

While assessing outcomes is a driving force behind institutional and programmatic redeployment of resources and efforts under the guise of "quality assurance" (Millis, Lowe & Aretz, 2003; Yorke, 2002), learning assessment begins at the course level. It is the evaluation of the learning context or situation in terms of what is expected as intended learning outcomes and actual performance. The idea is to restructure the learning situation to enable and enhance students' growth of knowledge of the subject matter (Salegna & Bantham, 2002).

In a programming course, the basic problem is if the students have understood paradigmatic concepts as well as the grammar and syntax of a particular computer language, which embodies the ideas of the programming paradigm. This problem is especially interesting when the language is object-oriented. It is no longer a matter of merely learning reserved words and syntax of a specific computer language. It also involves dealing with and understanding the object-oriented framework of the syntax, reserved words, and their functions in writing a working computer programs (Peterson & Quarstein, 2001).

Assessing students' work is a way of trying to determine what they know, and how they demonstrate their knowledge. Assessment is conducting an audit of what the student has learned (Mitri, 2003; Palomba & Banta, 1999). This may be called a knowledge audit of the knowledge assets learned (that is, syntax etc). This is also a way of evaluating teaching effectiveness. Evaluating learning means looking at how the artifacts (course notes, Java programs and examinations) of students' experiences in a course show engagement with and learning of content and which end up in use (demonstrable applications of content to programming problems) (Serwatka, 2003). As-

sessing learning outcomes also means trying to determine if students transfer and use information and behaviors in other arenas of performance, both other courses and employment opportunities (Dominguez & Ridley, 1999).

Assessment is an evaluation of the thinking processes students go through to do an assignment, and how we know this (Wade, 1999). This assumes that the assigned work is a way or instrument for focusing and engaging students in the learning and using of the course's subject matter to solve problems. An assignment is a way of getting implicitly known (assumedly learned) material to be explicitly represented in acceptable and appropriate (to the logic of the subject matter and discipline) forms and results (Ebert-May, Batzli & Lim, 2003; Palomba & Banta, 1999).

Articulating Syllabus-embedded Assessment Models

Judging the growth of knowing by members of a course, in this case a Java programming course, is not a facile affair. The reason, why this assessing is difficult, is that learning or coming to know something, is a personal affair and in the head. An important aspect of the course when being designed is how to get the knowing in the head to be presented as knowing in the world (Norman, 1988). A course plan (sometimes called a syllabus) has at least three segments that directly relate to assessing, coming to know about, outcomes of learned subject matter among persons studying a particular course. A syllabus represents a taxonomy of content and artifacts, objects-to-be-known, and known-ways-of-doing-things as well as a timeline of events which are expositions of the content (Mitri, 2003).

Explicit Evaluative Model

The explicit evaluative model of a course is the general descriptive framework and a set of subject matter specific objectives. The description and learning objectives give an interpretive frame for evaluating performance in solving programming problems. That is, the course description and course objectives construct a strong relationship of the course to the subject matter and discipline. Course descriptions and course objectives are not idiosyncratic views of the subject. These work to provide assessing heuristics.

Implicit Evaluative Model

The implicit evaluative model of a course is a taxonomy of paradigm-specific categories of concepts-to-be-known. These paradigm-specific categories (see Appendix 2) are modes of knowledge claims, and identifiers of potential knowledge assets. This model consists of the architecture (taxonomy) of the subject matter to be known (learned), arranged in some hierarchical and logical order that progressively immerses people studying the subject matter in the knowledge space (conceptual space) of the subject. These become a basis for a system of evaluative heuristics at a conceptual level (Millis et al., 2003; Yorke, 2002). Generally, the heuristics are in need of articulation as they are usually tacit in nature.

Model for Assessing Learning Outcomes

Assessing Learning

Assessing learning in light of the explicit and implicit evaluative models of a course syllabus has to do with the presentation, as a report, of the results of the progressive immersion (the course timeline) into the conceptual space of the subject matter. The report is a type of knowledge audit. In the report, the student presents a claim of knowing that the program works. Proof of this are the facts generated and recorded in the process of developing and implementing the program required by the assignment. The report is a way of thinking visually about the problem or problems

at hand. This presentation is a reporting of the notable events resulting from syntactical and logical errors in developing a program and a person's response to these events. In the case of a programming course, i.e., a Java programming course, the report is a verification of knowing in the head of the concepts, techniques, and syntax of Java and Object-Oriented programming. An important assumption here is that the report, presenting knowing in the head as knowing in the world (Norman, 1988), is a somewhat accurate representation of the knowing in the head of a person studying the subject matter.

The model is a report that is produced and handed in as a course requirement to demonstrate a student's progress in learning how to analyze programming problems, and develop Java programs as solutions. The report is also a way by which a student demonstrates his or her abilities to make clear by writing about what it is that he or she has done to solve the problem. The Java programs are the results of applying Object-Oriented programming concepts and techniques, and, of course, the Java APIs to devise solutions to given programming exercises or problems. It is assumed that the reporting of actions taken to overcome errors and mistakes in program code is descriptive of learning. This description is either sparse or dense. The report shows engagement or no engagement by the level of informational detail reported on, or, in the language of the subject matter, how much information has been encapsulated

The report consists of several sections. These sections are the assigned problem, the output of the program that solves the problem, a discussion of errors and events encountered and worked through on the way to solving the problem and getting the program to execute successfully, the use of Unified Modeling Language to describe the programming model, and finally, the Java code of the class or classes developed and implemented to solve the problem.

Problem-assigned–to-be-solved

This first section (see Appendix 1) is the identification of the programming exercise and problem and its specifications. This part of the report is to reinforce the idea that programming is a problem solving approach and that the problem itself should be identified clearly for any reader.

Output-of-program-model

This is the second section (see Appendix 1) shows the program's results or output and indicates that the program successfully solved the problem. Output from the program consists not only of screen shots depicting the actual results of the program but also the screen shots of any input necessary to the program.

Discussion-of-activity

This third section (see Appendix 1) is the most important one because it is the presentation and discussion of the problems encountered and things done to overcome the problems (errors) during the development, implementation, and testing of the program code. In the case of errors in compilation, the learner is expected to capture the error messages, so that they can be understood, and remembered for future use. The student is expected to keep a progressive record of the things done to solve any problems causes by errors in the code. This discussion is to be as detailed as possible, and as complete as possible. It is a way of managing the growth of knowledge as the student progresses through the exercise. This section is about having people make conscious the stuff what usually remains in their heads. This means that the ideas, in the form of Java syntax, and procedures that are considered and tried to make the programs execute, have to be written out. This makes things visual. Consequently, this section is about knowledge capture and knowledge building. This section represents incremental steps in knowing about solving programming problems. This allows for self-reflection and, perhaps, becoming more efficient in programming.

There is also a repository of events and activities based on these events. This section personalizes the learning situation.

Problem-model-presentation

The fourth section (see Appendix 1) of the report is a presentation of the UML (Unified Modeling Language) class and object diagrams representing the classes and objects making up the programming solution. This section is intended to reinforce analysis and design before development and implementation. By enforcing the use of UML diagrams, the section also serves as a way of thinking visually about a problem and its solution. It reinforces UML usage and Object-Oriented concepts for software development. The section also intends to help learners structure and systematize their approach to thinking analytically about writing program code.

Java-code-presentation

The fifth and final section (see Appendix 1) is the presentation of the actual Java code used to solve the programming exercise. This is intended as a means of reflection on Java syntax use.

Observations on Using the Model and its Effectiveness

History

The report technique has been a feature for 5 years. The report began as an attempt to "remember" actions taken to correct syntactical and logical errors in program. Students are apt to "solve for the moment" in order to complete on time an assignment. They would frequently be resolving the same problems over and over again. The reports become repositories of learned and known solutions to development and implementation problems. No attempt, at this time, has been made to ascertain whether or not students see these reports as a repository of knowledge.

Labor Intensive

Doing the reports are work intensive because of the *Discussion-of-activity* section. If this part of the report is done right, constant attention is paid to always collecting the error messages, understanding what the messages mean, and developing or changing the program's code to overcome the problem. This is tedious and tiresome at first.

Problems of Encapsulation

One of the effects of such tediousness is encapsulation or the hiding of information. There is a tendency to sometimes describe errors and their solutions in generalities, and not the details. Generalities usually do not lead to efficient information use for solving future developmental problems.

On-going Recording

A reason why the discussion section of the report is labor intensive and perhaps tedious, is the necessity of on-going recording and capturing of error messages, their analyses, and the writing out, or drawing out, of thinking about solving the problems that the error messages indicate within the code.

Individuals and Groups

The initial and current use of the report focuses on an individual's navigation through any problems encountered in developing and implementing Java programs. However, it has been used to

report on group activity. A group of advanced Java students who were working as a team on a semester-long Java application project were asked to do weekly discussions that were consolidated into one report at the end of the semester. In this case, some of the discussion of problems were issues of relationships among the group's members.

Does It Work?

Anecdotally, the report as an assessment technique seems to work. Perfectly? No! Future development of this model will have to look at ways of evaluating its effectiveness as a learning tool. One possible way of doing this is to ask for a summarizing and reflective critique of past reports produced throughout the course. Another possible way will be to relate what the student claimed to have learned to performances on examinations (or as I tend to call them, knowledge audits). Another possible way would be to see if, over the course of the semester, past reports have been accessed and "mined" for previous solutions in reference to current problems, consequently decreasing the number of errors generated in developing Java programs. For this to happen, a more structured presentation of errors encountered and actions taken on the errors must be used.

Conclusion

The paper presented a discussion of thinking about how the assessment of learning outcomes might be accomplished at the course level. A specific evaluative technique, a report, was described. The technique was discussed in relation to the implicit outcomes assessment model considered to be embedded in the syllabus. The paper presented the implicit model as well as the explicit taxonomy of categories model. The paper is an exploratory discussion about framing a particular assessment technique.

References

Chrysler, E. & Van Auken, S. (1999, Summer). A methodology for alumni assessment of an MIS program. *The Journal of Computer Information Systems, 39* (4), 33-39.

Cunningham, B. & Omolayole, O. (1998, March/April). An assessment-oriented syllabus model for business courses. *Journal of Education for Business, 73* (4), 234-240.

Dominguez, P. S. & Ridley, D. (1999, September). Reassessing the assessment of distance education courses. *T.H.E. Journal, 27* (2), 70-76.

Ebert-May, D., Batzli, J., & Lim, H. (2003, December). Disciplinary research strategies for assessment of learning. *BioScience, 53* (12), 1221-1228.

Millis, B. J., Lowe, J. K., & Aretz, A. J. (2003, Summer). Making program assessment work. *Liberal Education, 89* (3), 38-41.

Mitri, M. (2003, Summer). A knowledge management framework for curricular assessment. *The Journal of Computer Information Systems, 43* (4), 15-24.

Nichols, J. O. & Nichols, K. W. (2000). *The departmental guide and record book for student outcomes assessment and institutional effectiveness* (3rd ed.). New York: Agathon Press.

Norman, D. A. (1983). Some observations on mental models. In D. Gentner & A. L. Stevens (Eds.), *Mental models* (pp. 7-14). Hillsdale, NJ: Lawrence Erlbaum Associates.

Norman, D. A. (1988). *The design of everyday things*. New York: Currency Doubleday.

Palomba, C. A. & Banta, T. W. (1999). *Assessment essentials: Planning, implementing, and improving assessment in higher education*. San Francisco: Jossey-Bass.

Peterson, P. A. & Quarstein, V. A. (2001). Quality assurance in education. *Bradford, 9* (1), 46-53.

Redmond, M. V. (1998, Fall). Outcomes assessment and the capstone course in communication. *The Southern Communication Journal, 64* (1), 68-76.

Salegna, G. J. & Bantham, J. H. (2002, March). Curriculum assessment-a systems approach. *Quality progress, 35* (2), 54-59.

Serwatka, J. A. (2003, Fall). Assessment in on-line CIS courses. *The Journal of Computer Information Systems, 44* (1), 16-20.

Sims, S. J. (1992). *Student outcomes assessment: A historical review and guide to program development*. New York, Westport, CN: Greenwood Press.

Taylor, J. H. (2002, Spring). Using assessment to build a culture of improvement. *Research Library, 5* (2), 38-39.

Wade, W. (1999, October). What do students know and how do we know that they know it? *T.H.E. Journal, 27* (3), 94-100.

Yorke, M. (2002). Subject benchmarking and the assessment of student learning. *Quality Assurance in Education, 10* (3), 155-171.

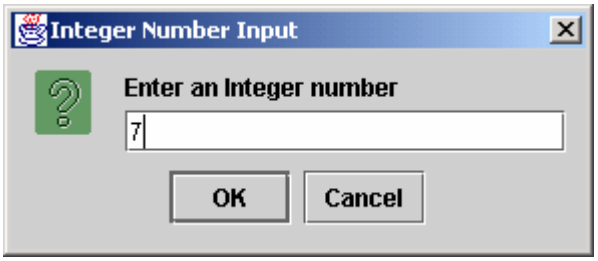
Appendix 1: Java Assignment Report Example

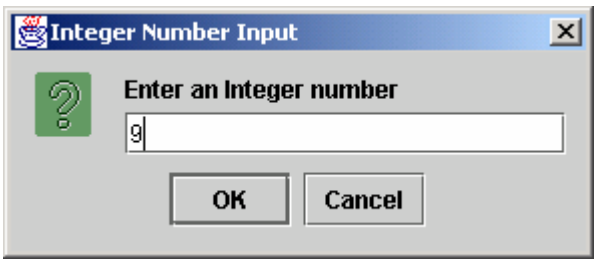
Problem-assigned–to-be-solved

The problem is to demonstrate method overloading as well as the use of the swing component JoptionPane and two of its methods for getting input from the keyboard and displaying output information in dialog boxes. Uses a method that determines the larger number of two or three when they are input. The Java class name is DialogMethodOverload.

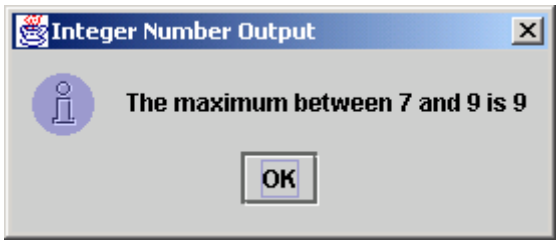
Output-of-program-model

Input screens for DialogMethodOverload.java using primitive datatype integer numbers

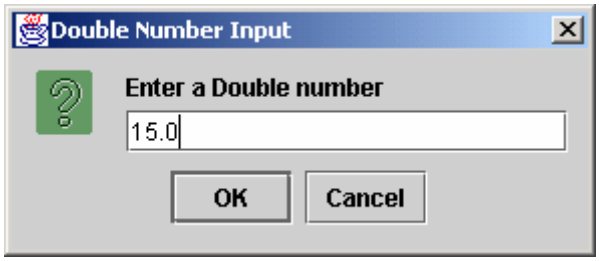


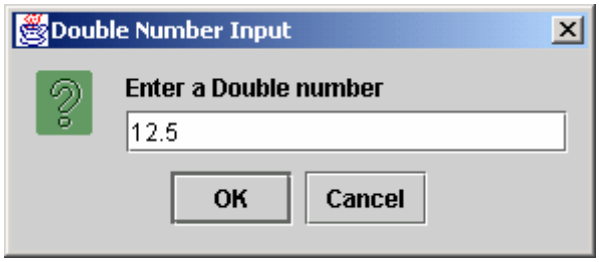


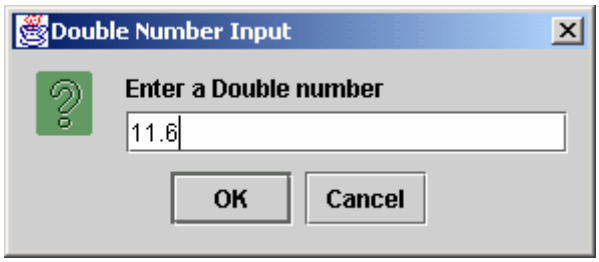
Output screen for DialogMethodOverload.java



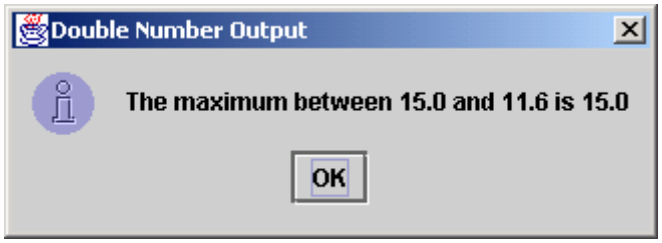
Input screens for DialogMethodOverload.java using primitive datatype double numbers

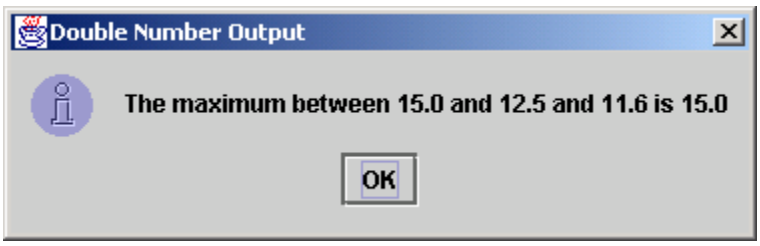






Output screens for DialogMethodOverload.java





Discussion-of-activity

These were the error messages produced as the program DialogMethodOverload .java was worked on.

First Compile

Errors Encountered

21 error messages showed up. They were of two types.

Line 18, 23, 36, 41, 46:
<null> cannot be dereferenced
String stgNum1 = JOptionPane.showInputDialog(null.

Lines 18, 19, 23, 24, 33, 34, 36, 37, 41, 42, 46, 47, 56, 57, 65, 66 :
cannot resolve symbol
symbol : variable JOptionPane
location: class exch4.DialogMethodOverload
iPrompt, iTitle, JOptionPane.QUESTION_MESSAGE);

Explanation of Errors

The errors were mostly misspellings and forgetting about the necessity of using certain syntax to display the JoptionPane.

Resolution

Corrected misspellings and added import statement: import javax.swing.JOptionPane;

Second Compile

Errors Encountered

Still 5 error messages showed up.

Lines 20, 25, 38, 43, 48:
<null> cannot be dereferenced
String stgNum1 = JOptionPane.showInputDialog(null.

Explanation

These error messages showed a period after null. This should be comma.

Resolution

Changed the period after the null reserved word to a comma.

Third Compile

Errors Encountered

2 errors showed up with variable use.

Lines 21, 26:
cannot resolve symbol
symbol : variable iPrompt
location: class exch4.DialogMethodOverload
iPrompt, iTitle, JOptionPane.QUESTION_MESSAGE);

Explanation

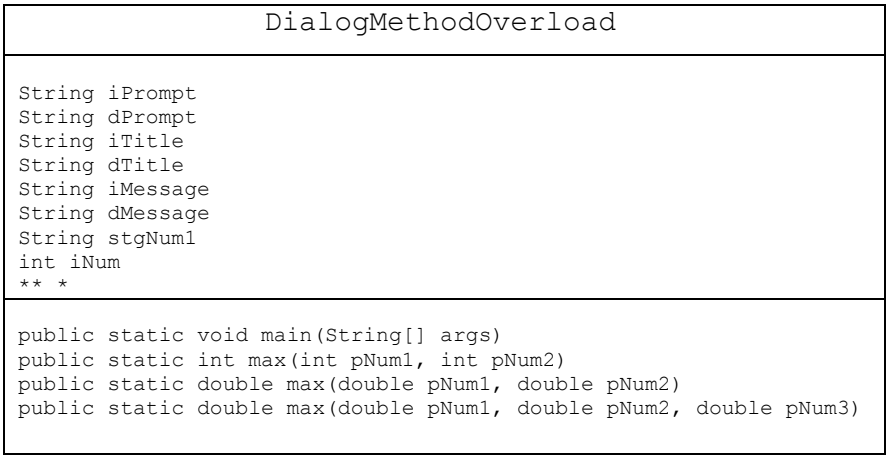
Misspelling of the identifier when it was declared.

Resolution

Changed and program ran successfully.

Problem-model-presentation

The class diagram



Java-code-presentation

```
//DialogMethodOverload e.g. 4.3, p. 163

package exch4;

import javax.swing.JOptionPane;

public class DialogMethodOverload

{

        public static void main(String[] args)
        {
                String iPrompt = "Enter an Integer number";
                String dPrompt = "Enter a Double number";
                String iTitle = "Integer Number Input";
                String dTitle = "Double Number Input";
                String iMessage = "Integer Number Output";
                String dMessage = "Double Number Output";

               String stgNum1 = JOptionPane.showInputDialog(null,
                        iPrompt, iTitle, JOptionPane.QUESTION_MESSAGE);

                int iNum = Integer.parseInt(stgNum1);
```

```
            String stgNum2 = JOptionPane.showInputDialog(null,
                    iPrompt, iTitle, JOptionPane.QUESTION_MESSAGE);

             int jNum = Integer.parseInt(stgNum2);

             int kNum = max(iNum, jNum);

             String outVar1 = "The maximum between " + iNum +
                            " and " + jNum + " is " + kNum;

                    JOptionPane.showMessageDialog(null, outVar1, iMessage,
                            JOptionPane.INFORMATION_MESSAGE);

            String stgNum3 = JOptionPane.showInputDialog(null,
                    dPrompt, dTitle, JOptionPane.QUESTION_MESSAGE);

                     double mNum = Double.parseDouble(stgNum3);

            String stgNum4 = JOptionPane.showInputDialog(null,
                    dPrompt, dTitle, JOptionPane.QUESTION_MESSAGE);

                     double nNum = Double.parseDouble(stgNum4);

            String stgNum5 = JOptionPane.showInputDialog(null,
                    dPrompt, dTitle, JOptionPane.QUESTION_MESSAGE);

                     double oNum = Double.parseDouble(stgNum5);

                    double pNum = max(mNum, oNum);

            String outVar2 = "The maximum between " + mNum +
                    " and " + oNum + " is " + pNum;

                    JOptionPane.showMessageDialog(null, outVar2, dMessage,
                            JOptionPane.INFORMATION_MESSAGE);

             double qNum = max(mNum, nNum, oNum);

            String outVar3 = "The maximum between " + mNum +
                    " and " + nNum + " and " + oNum + " is " + qNum;

             JOptionPane.showMessageDialog(null, outVar3, dMessage,
                            JOptionPane.INFORMATION_MESSAGE);

                    System.exit(0);

}//end main

public static int max(int pNum1, int pNum2)
{
            int result;
            if (pNum1 > pNum2)
                                    result = pNum1;
                    else
                                    result = pNum2;
                    //end if else
                    return result;
}//end max int

public static double max(double pNum1, double pNum2)
{
```

```
                double result;
                if (pNum1 > pNum2)
                                        result = pNum1;
                        else
                                        result = pNum2;
                        //end if else
                        return result;
        }//end max double

        public static double max(double pNum1, double pNum2, double pNum3)
        {
                double result;
                return result = max(max(pNum1, pNum2), pNum3);
        }//end max double 3

}//end class DialogMethodOverload
```

Appendix 2: Subject Matter Taxonomy

Categorical Taxonomy of the Object-oriented Paradigm (partial)

- Object-oriented analysis & design
  - Class and object
  - States (of objects)
  - Sequences of actions
  - Inheritance
  - Encapsulation
  - Polymorphism
- Object-oriented programming
  - Class and object
  - Members of class
    - Data fields (Attributes)
      - Primitive datatypes
      - Casting
    - Methods (Actions)
  - Classes
    - Abstract classes
    - Interfaces
    - Inner classes
    - Concrete classes
  - Constructors
  - Inheritance
    - Classes
      - Extends
    - Interfaces
      - Implements
  - Encapsulation
    - Modifiers
      - (default)
      - public
      - private
      - protected

- Packages
- Polymorphism
  - Overloading methods
  - Overriding methods
  - Abstract methods
- Exceptions
- I/O byte & character-oriented streams
- Serialization
- AWT & Swing classes
  - JavaBeans component model
- Threads & Multithreading
- Event-driven programming
- Utility & Collections classes

Biography

Robert Joseph Skovira is a Professor in the Computer and Information Systems Department at Robert Morris University, Moon Twp PA, USA (near Pittsburgh PA). He has taught various programming languages over the past 21 years; he currently teaches undergraduate and graduate courses in Java. He also teaches courses on Decision Support Systems, Ethics of Technology, and Information and Web Design. He teaches a doctoral course in Ethnography of Information Systems (Doctor of Science in Communications and Information Systems). He is currently developing courses on Knowledge Management. He will be a visiting research professor in the Departments of Library and Information Science, and Psychology at Comenius University, Bratislava, Slovakia (Spring 2006). He resides with his wife Mary Elizabeth in Coraopolis PA.