Computer Programming Assignments Based on a Progressive and Constructive Approach

Christian Servin
Computer Science Instructor
El Paso Community College
Outline

1. Student Learning Outcomes
2. Teaching/learning challenge
3. Progressive and Constructive Approach
4. Intervention: Assignments Design
5. Results
6. Future Work
Student Learning Outcomes

• Recognize and implement syntax and semantics of an Object Oriented language to solve problems
• Utilize functions and parameter passing involving both primitive types and reference types
• Apply problem-solving strategies to design a solution to a problem in addition to testing and debugging
• Design simple ADTs to solve a problem similar to one seen before
• Formulate complex logical expressions involving multiple and/or/not combinations
• Implement nested if statements
• Formulate complex arithmetic expressions involving operators of differing precedence and associativity and understand the order of evaluation of sub-expressions
• Implement simple ADTs incorporating multiple primitive instance variables and at least one reference instance variable, with appropriate accessor and modification methods
• Utilize dynamic memory allocation, e.g., new, to create new objects
• Utilize simple I/O to read and write character and numeric data to and from files, keyboard, and display
• Utilize predefined or built-in exceptions to handle violation of preconditions
• Utilize standard documentation to determine the use of an unfamiliar class or method
Teaching/Learning Challenge
1. Excitement reduction
2. Frustration
3. Approx. 2% drop
4. They don't know what is going on

---

**Student Excitement!!**

**CS1**

- Starts

---

**Test 1**

- Students understand Problems/test questions
- Frustration increases
- Memorization is NOT useful

---

**Test 2**

- "I don’t know if CS is for me"
- "I have the wrong impression about computers"
- "I just don’t get it!"
- Stop attending to class

---

**Test 3**

- Enrollment decreased (20~30%)
- CS majors, ECHS, not enough IT
- They partially understand what is going on
- 50% are ready to take CS2
- Some don’t like CS at all anymore

---

**CS1**

- Finishes

---

**Student Excitement!!**

**CS1**

- Starts

---

**Test 1**

- Students understand Problems/test questions
- Frustration increases
- Memorization is NOT useful

---

**Test 2**

- "I don’t know if CS is for me"
- "I have the wrong impression about computers"
- "I just don’t get it!"
- Stop attending to class

---

**Test 3**

- Enrollment decreased (20~30%)
- CS majors, ECHS, not enough IT
- They partially understand what is going on
- 50% are ready to take CS2
- Some don’t like CS at all anymore

---

**CS1**

- Finishes
How to motivate the Community College student to keep him/her in the field (context matters)

Student Diversity

- EPCC Associate in Arts in IT and CS
- Transfer Students (UTEP, NMSU, Texas A&M)
- Early College Students (ECHS)
We want a course that:

- Keeps consistency with the student learning outcomes
- Is motivational for IT, CS, Math, and other fields
- Provide strong fundamentals in computer prog.
- Is fun and excited
A Progressive and Constructive Course
Progressive Project Assignments

Programming Problem set (non-credit)

Quizzes (50% from Prob. set + 50% class)
Intervention: Assignments Design
Assignments Design

- Break one Project into small components
  - Student learn problem-solving
  - Student understands coding techniques
  - Student develops his/her own prog. style
  - Student understands the “big-picture” on computer programming

- Complementary exercises are assigned during semester (to improve prog. skills)
Assignments
Design

• Main Projects:
  ★ Calculator
  ★ Movie Rental System
  ★ State ID System
  ★ Mp3 Player
  ★ Piggybank
Results
Results

• Apply Lab 1 and 2 at COSC1437/CS2401

• Lab1: UTEP’s CS 2401 Lab 1 to students

• Lab2: Incorporate extra complexity:
  • read from file,
  • create object,
  • store object into an array,
  • then print the array using accessors

• Quiz 0 (Similar to final test for CS1)
Results

• Group of: 26 students
  • 17 students were part of the Progressive Constructive Approach (PCA)
  • 5 students different class at EPCC
  • 4 transfer students
## Results

<table>
<thead>
<tr>
<th>Instruments</th>
<th>PCA (17 students)</th>
<th>Other EPCC (5 students)</th>
<th>Transfer (4 students)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTEP CS2401Lab 1</td>
<td>13</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>EPCC COSC1437 Lab 2</td>
<td>14</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Quiz 0 (based on final Test CS1)</td>
<td>12</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Future Work
Future Work

• Incorporate more instruments to evaluate the Progressive and Constructive Approach

• Incorporate the Progressive and Constructive Approach to COSC 1437 to prepare students to COSC 2436

• Incorporate quantitative approach to the approach
Thank you for your Attention

• Christian Servin
• El Paso Community College
• prof.chris.servin@gmail.com
• http://www.cs.utep.edu/christians/
Objective: To understand basic functionalities in Java such as arithmetic operators, assignment statements, manipulation of variables, and evaluation of arithmetic expressions. Additionally, understand how to handle input from the user.

```java
                        ******************************
                        ********** This is a Calculator **********
                        ******************************
Adding: number1 = xxx and number2 = yyy
The result is: zzz

Subtracting: number1 = xxx minus number2 = yyy
The result is: zzz

Multiplying: number1 = xxx times number2 = yyy
The result is: zzz

Dividing: number1 = xxx over number2 = yyy
The result is: zzz

                        ******************************
```

Write a simple calculator that takes two numbers and performs the following operations: addition, subtraction, multiplication, and division. The numbers (i.e., number1 and number2) should be asked from the user (e.g., You can use code from book: Code Listing 2-32, as an example of Dialog Boxes as a way to ask the user to enter data)
After Lab 1 students:

• Know how to declare variables (different types, e.g., int, double, String)
• Recognize primitive types from the object String
• Know how to print to the screen
• Know how to use arithmetic operations
Lab 2 (I/O)

• **Objective**: To understand basic functionalities in Java such as arithmetic operators, Scanner and JOptionPane as a way to interact with the user, manipulation of variables, and evaluation of arithmetic expressions.

• Modify your program 1 (Calculator.java). This program should ask the user for two values, i.e., x and y. Use JOptionPane to read the input from the user. Once you obtain the two inputs from the user, perform the basic arithmetic operations, i.e., addition, subtraction, multiplication, and division. You should print the result of your 4 arithmetic operations. For extra credit, display in a dialog box the result. (Hint: you must use Integer.parseInt or Double.parseDouble in order to work).
After Lab 2 students:

• Know how to ask the user for input (using JOptionPane, Scanner)
• Know how to \texttt{parseInt} or \texttt{parseDouble}
• Know why \texttt{double} is needed instead of \texttt{int}, e.g., division needs \texttt{double}
Lab 3

(Control Structures)

• Modify your Lab2 and add additional functionalities such as square root, sin, cos, max, and min. Your program must ask the user for what kind of operation would like to perform, and ONLY that operation must be executed in your program. Then return the answer in a dialog box.

• Observations

★ Division cannot divide by 0

★ Square root does not work for negative numbers sqrt, cos, sin only accept 1 number, the rest of the operations accept 2.

★ Any operation that is not in the list of operations should be properly handled, e.g., if user enters ‘hello’ as an operator a proper error message should display such as “hello is not a valid operator for this calculator”
Lab 3
After Lab 3 students:

- Know how to process `String` input
- Parse `int` and `double` types
- Handle different options as an input
- Handle small exceptions for input
- Use `nested-if`s and `else-if`
Lab 4 (Loops)

- Based on Lab 3, implement any kind of looping into your calculator, in such a way that the user can continuously work with your calculator application.

- Your calculator shall ask the user for the desired operation that would like to perform, followed by the answer.

- After the answer is displayed, you must ask the user "would you like to perform another operation?" If the user says "yes" repeat the procedure asked before. If the user says "no" the application should be closed, other answer should display an error message followed by the question of "would you like to perform another operation?".

- Observations:
  - If the user asks to perform division, and the second number is zero, then your program should tell the user: Cannot divide by zero, please enter the second number. Your program must keep asking for a non-zero number.
  - In similar fashion, when the user desired to perform the square root "sqrt", the number must be non-negative.
After Lab 4 students:

- Know how loops work (specially do-while)
- Compare Strings and its methods
- Extended version (Read and Write Files)
Lab 5 (Methods)

Lab 5 is based on Lab 4. Write the following methods on your calculator:

- **addition**: This method takes two real numbers (i.e., n1 and n2), and performs the binary addition of both numbers. It returns a "double", which represents the addition of n1 + n2

- **subtraction**: This method takes two real numbers (i.e., n1 and n2), and performs the binary subtraction of both numbers. It returns a "double", which represents the subtraction of n1 - n2

- **product**: This method takes two real numbers (i.e., n1 and n2), and performs the binary product of both numbers. It returns a "double", which represents the product of n1 * n2

- **division**: This method takes two real numbers (i.e., n1 and n2), and performs the binary operation division of both numbers. It returns a "double", which represents the division of n1 / n2
Lab 5
(Methods)

• **mySquareRoot**: This method takes two parameters: *n*: the number whose square root is desired to be obtain and *tol*: precision of the iterative method.

• This program uses Newton's method to find the square root of a positive number. This is an iterative method and the program keeps generating better approximation of the square root until two successive approximations have a distance less than the specified tolerance.

• **myFactorial**: This method takes an int value, and returns the factorial of this number using an iterative method (i.e., loop). The factorial of a non-negative integer *n*, denoted by *n!*, is the product of all positive integers less than or equal to *n*. For example, $5! = 5 \times 4 \times 3 \times 2 \times 1$.
After Lab 5 students:

• Know how to declare methods
• Know how to call methods from different places in the code
• Practice loops and conditional statements
• Incorporate mathematical background and algorithmic ideas
Lab 6

(Array Manipulation)

- Given as an input operations.txt containing the following math expressions

```
+ 45 3
^ 2 2
sqrt 16
- 15 7
* 34 5
+ 25 7
sqrt 33
! 8
```

- Modify your Lab 5, in such a way that you read your file operations.txt (Hint see Ch 4), and identify how many operations you performed over the file. E.g., in this example, there are 2 additions, 1 subtraction, 1 multiplication, 0 division, 2 square root, 1 power, 1 factorial. The total number of operations must be store it into an array, e.g.

```
• Addition       position 0
• Subtraction    position 1
• Product        position 2
• Division       position 3
• Power          position 4
• Square root    position 5
• Factorial      position 6
```
In short, for every line you read, you increment the index where that operation is located, perform the mathematical operation by invoking the methods, then write the answer into a file named answers.txt, that will look as follows:

- $45 + 3 = 48$
- $2^2 = 4$
- $\sqrt{16} = 4$
- $15 - 7 = 8$
- $34 \times 5 = 170$
- $25 + 7 = 32$
- $\sqrt{33} = 5.74$
- $8! = 40320$
Finally, write a method named: `writeTotal`. This method takes as a parameter an array of ints (containing the counters for all math operations). Then appends into the file containing the answers the total number of operations performed on file `operations.txt`.

- Total additions: 2
- Total subtractions: 1
- Total Multiplications: 1
- Total Divisions: 0
- Total Power: 1
- Total Square roots: 2
- Total Factorials: 1
- Total Sin: 0
- Total Cos: 0
After Lab 6, students:

- Know how to declare an array, and perform operations using array
- Know how to work with files (read/write)
- Know how to invoke argument-based methods
- Know the difference between global and local counters
Lab 7
(Exception Handling)

Based on Lab 6, incorporate ExceptionHandling. The following methods should throw the following exceptions:

- **mySquareRoot**: throws an `IllegalArgumentException` in case the number is less than zero.

- **myFactorial**: throws an `IllegalArgumentException` in case the number is less than 1.

- **division**: throws an `ArithmeticException` in case the second number, i.e., the divisor is zero.

- When the user is asked if he/she would like to perform another operation that is not identified within the cases that your calculator performs, then you should throw `IllegalStateException`, specifying that the input is not correct.

- Implement **two** Exceptions that potentially can be thrown in your program.

- In the main, handle the exceptions properly, e.g., using try/catch structures.
After Lab 7, students:

• Know how to create exception handling structures

• Know how to create their own exceptions

• Know how to *throw* and *catch* exceptions

• Reprised method invocation and method definition
Lab 8
(OOP)

- Implement a class named MathExpression.java
- A math expression can be either an unary or a binary expression (kind), either one or two arguments depending on the kind.
- Write a class with the methods depicted on the UML Figure.
- In addition, write accessors and mutators for each field.
- Write a program that reads the same file "operations.txt", and creates an object MathExpression.java based on each line of the file.
- Then store each MathExpression into an array of MathExpression objects.
- Write a method named performOperation(), that takes one parameter, which is an array of MathExpression
After Lab 8, students:

• Know how to define a blue print (class)
• Know how to create fields, constructors, methods
• Know how to create, implement, and invoke accessors and mutators
• Know how to read files, tokenize each line, extract object values and store them into an array
(Lab 9)
OOP + Inheritance
After Lab 9, students:

• Incorporate inheritance into Lab 8 with additional functionality, and separation of duties
• Apply polymorphism and method and constructor overloading
• Override method i.e., `toString()`
After CS1 course

• Students have the big picture about fundamentals of programming I

• Students incorporate different programming components to make “better” programs

• Students in CS II have good programming basics independently if they are UTEP, NMSU, EPCC, ECHS

• Students incorporate divide-n-conquer/separation of duties on their programs