Running Head: PROCEDURAL VS. OOP

Week 1 Discussion: Procedural Programming versus Object Oriented Programming

Jered McClure

Walden University

Week 1 Discussion: Procedural Programming versus Object Oriented Programming

Procedural programming is a type of code writing whereby the programmer lays the code out in a step-by-step fashion. “Procedures – simply contain a series of computational steps to be carried out. Any given procedure might be called at any point during a program’s execution, including by other procedures or itself” (Network Dictionary, 2007, p. 387). In essence, the programmer writes a bit of code and then references that code throughout the application. However, the code’s lifetime only exists within the scope of that application’s runtime.

What this means is, if that particular code needs to be used elsewhere, it would need to be copied in full to the new application. This in itself is not an issue, and in small applications can be a boon as the code is easily referenced. However, “when changes are made to the main procedure, those changes can cascade to the sub procedures… eventually impacting all procedures in the pyramid” (Dag, 2009). In essence, as application projects begin to expand, editing and maintaining code can become time consuming, or impossible.

For instance, take this code:

/\*

\* This is an example class that procedurally plays with colors.

\*/

**public** **class** variablesAreFun {

**public** **static** **void** main(String[] args) {

String firstColor, secondColor, thirdColor;

firstColor = "red";

secondColor = "green";

thirdColor = "blue";

{//code referencing variables

firstColor = firstColor + thirdColor;

{//more code referencing variables changing them

secondColor = firstColor;

{

firstColor = "dinosaur";

{//even more code referencing variables and

//changing them again

secondColor = "Atlantis";

}

{//hey lets reference those variables and

//change them some more

**int** someNumber1 = 2732568 \* 2;

thirdColor = "" + someNumber1;

}

}

}

}

System.*out*.println("My colours should be: red green blue");

System.*out*.println("My colors are: " + firstColor + " " +

secondColor + " " + thirdColor);

//My colours should be: red green blue

//My colors are: dinosaur Atlantis 5465136

//what happened?!!

}

}

While this code is indeed simplified, it displays the drawbacks of procedural code writing quite well. At each block, an action is taken against the color variables for the class. This causes those variables to change and mutate such that, at the end, their stored data is nothing as was to be expected.

At the small scale this is not an issue. The programmer simply goes through and adjusts each of the blocks so that the output is what is expected. However, on a large application project, this task could take a massive amount of time and effort, and in the end, may be unsolvable as the interactions within the different blocks may cause system failure if the data is changed. Imagine if there were ten thousand lines of code, if one of these variables are used anywhere in that code, they would have to be checked to make sure that it matched as expected.

Object Oriented Programming (OOP), on the other hand, offers a simple route to rectify this issue. By creating a someColors() object, all the variables could be encapsulated within their own black box. For instance:

/\*

\* This is an example object that plays with colors.

\*/

**public** **class** someColors {

//declare my class variables, make them private

**private** String firstColor, secondColor, thirdColor;

//Constructor

**public** someColors(String first, String second, String third){

firstColor = first;

secondColor = second;

thirdColor = third;

}

//Default constructor

**public** someColors(){

**this**("","","");

}

//update firstColor

**public** **void** updateFirstColor(String first){

firstColor = first;

}

//update secondColor

**public** **void** updateSecondColor(String second){

secondColor = second;

}

//update thirdColor

**public** **void** updateThirdColor(String third){

thirdColor = third;

}

//update all colors

**public** **void** updateAllColors(String first, String second, String third){

firstColor = first;

secondColor = second;

thirdColor = third;

}

//return the first color as String

**public** String getFirstColor(){

**return** firstColor;

}

//return the second color as String

**public** String getSecondColor(){

**return** secondColor;

}

//return the third color as String

**public** String getThirdColor(){

**return** thirdColor;

}

//output the colors to the system

**public** **void** printColors(){

System.*out*.println(firstColor + " " +

secondColor + " " + thirdColor);

}

}

This object could then be called using:

/\*

\* This is an example class that objectively plays with colors.

\*/

**public** **class** variablesAreFun{

**public** **static** **void** main(String[] args) {

someColors colors = **new** someColors("red", "green", "blue");

{//change my colors around

colors.updateFirstColor("dinosaur");

{//this looks pretty

colors.updateSecondColor("Atlantis");

{//Numbers are art!

colors.updateThirdColor("5465136");

}

}

}

colors.updateAllColors("red", "green", "blue");

System.*out*.print("My colors are: ");

colors.printColors();

//Output: My colors are: red green blue

//That's what I thought!

}

}

What this means is that the someColors() object can be used multiple times throughout any other class, and that at any time the programmer can change the colors to be exactly what they need them to be. That is, the object “colors” is maintained within a black box whereby it can only be modified through direct method interfaces, which are well defined. Finally, if anything is wrong in the someColors() class, it only needs to be changed once to propagate throughout any other class that uses that object. This saves both time and effort, as there is no need to go through multiple instances of code to make sure it is being used correctly.

Reference

Dag, L. R. (2009). *OO Concepts*. Retrieved September 3, 2012, from Bilkent University: http://www.ctp.bilkent.edu.tr/~russell/java/LectureNotes/1\_OOConcepts.htm

Network Dictionary. (2007). Procedure Programming. *Network Dictionary*, 387.