CS330: Programming Language Project (PLP)

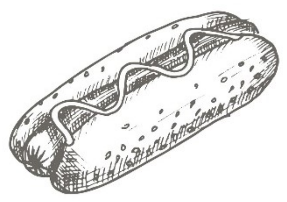
Assignment 6: Naming, Scope, and Bindings

What's in a name? That which we call a rose by any other name would smell as sweet.

-William Shakespeare

Naming plays an important aspect in programing languages because it enables programmers to identify variables, constants, operations, data types, and so on, "rather than using low-level hardware components" like an address (Vrajitoru, 2020). Valid names can vary by programming languages, some such as Java utilize camelCase, while other languages utilize foo_bar or lisp-name as a type of naming convention. In some languages reserve words such as if, else, and while, cannot be utilized as identifiers since these characters are named with a fixed purpose in the language already (w3schools, 2020).

Binding is the operation of associating two things, such as a name and the entity that name represents (Vrajitoru, 2020). I have created a simple example of binding based off the widely debated question: Is a hot dog a sandwich?



For example, someone who sees a hotdog as a sandwich, would bind the concept of a hotdog to the entity of a sandwich. The illustration below shows an arrow, which represents the binding operation of associating a hotdog to a sandwich.

$$\begin{tikzcd} \text{Hotdog} \arrow[r, "\text{binding}"] & \text{Sandwich} \end{tikzcd}$$

However this is a more traditional example of binding in programing languages:

```
int i = 5;
```

This line tells us a number of things about `i`. Firstly, it tells us that there is a binding of `i`'s type attribute to an integer (`int`). Secondly, the value of `i` is assigned to the number 5. So within this single line of code there are two types of bindings that effect `i`.

Binding time of an attribute is how an attribute is computed, which can be at translation time or at execution time (Vrajitoru, 2020). The terms **static** and **dynamic** generally described if the attributes are bounded before run time or during run time. Static binding occurs before run time and remains unchanged while the program runs (Vrajitoru, 2020). Dynamic binding occurs during run time *or* changes can occur while the program runs (Vrajitoru, 2020).

Scope is the region of the program where name-to-object binding is active (Vrajitoru, 2020). Different languages have different scoping rules, for example, in Java, the scope is where variables are both created and accessed, more practically the scope area in Java tends to be between each set of curly brackets (Geeks for Geeks, 2020). Down below I have created an example to better illustrate the concept of the scope for Java.

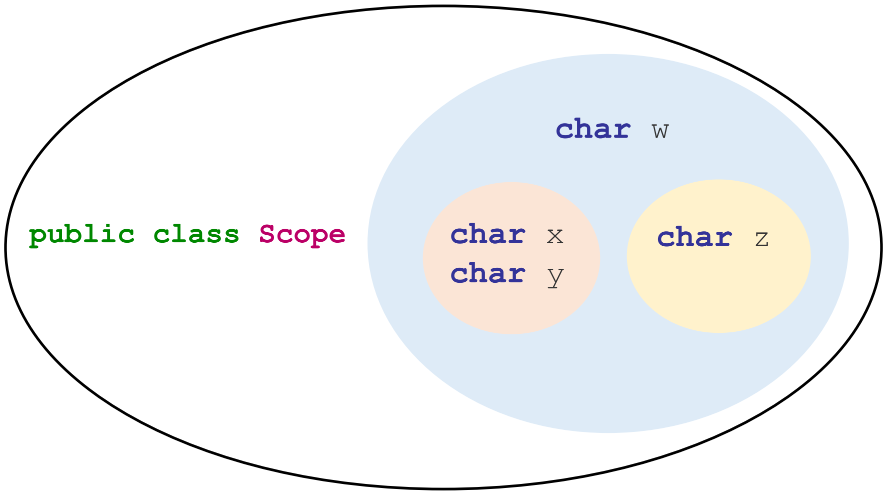
```java
public class Scope
{
    public static void main(String args[])
    {
        char w = 'd'; //notice the declaration of w
        {
            char w = 'd';
            char x = 'a';
            char y = 'b';
            System.out.println(x);
        }

        {
            char w = 'd';
            char z = 'c';
            System.out.println(z);
            //System.out.println(x);
           //would cause an error
          //because x is not in the scope
        }
        char w = 'd';

    }
}
```

The scope of x and y

The scope of z

The scope of w

view source code here



Discussion Questions:

1. Declare a variable (say, x) in the main body of your program. Then declare one inside of a for loop. Is there a conflict? Is the old variable overwritten, or do you now have two variables of the same name?

   There is a conflict in Java's case, this is because redefining variables in a nested scope would cause an error. Down below is an example of Java code with a variable `x` being redefined in the for loop statement which is the "problem line" for the whole entire source code.

```
class Loop
{
    public static void main(String args[])
    {
        int x = 1;
        for (int i =0; i <= 4; i++){
            int x = 3; //This line would cause an error
            System.out.println(var);
        }
    }
}
Output: error: variable declaration not allowed here
```

view source code here

2. What if the other x is inside of a method?

   This is not again possible because variable `x` was already defined if it is defined as `int x`. This is because `int x` was already declared in the same class that the method is under. However if `x` is declared without the binding of a similar data type as its class, then the value of `x` would print the value assigned within the method, in this case I have commented that line out in the code below.

```
public class Main{
   public static void main(String []args){
      int x = 7;
      var(x);
   }
   public static void var(int x){
      //x = 11; //the output would be 11
      int x = 11; //this will not work since variable x already defined
      System.out.println(x);
   }
}

Output: error: variable x is already defined in method var(int)
```

view source code here

3. Can you have variables that are globally accessible? What are the rules for creating them?

A global variable is a variable that has accessed in multiple scopes, languages like Python and C++ have accessed to global variables (Java Global Variables). However in the case of Java, Java does not have any concept of globally accessible variables, variables in Java are either in the scope of a class or a method, which can be seen in illustration above on page 2. However one loop-hole to make a variable more accessible and “global like” is by creating static variables also known as class variables (Java Global Variables). Down below is Java code that utilizes static variables.

```
public class Global {
  public static String a = "You can access this here";
  public static String b = "You can access this there";
  public static String c = "Even here!";

  public static void main(String[] args) {
  String newA = Global.a;
  String newB = Global.b;
  System.out.println(newA);
  System.out.println(newB);
  int i = 0;
  while (i < 1) {
    System.out.println(c);
    i++;
  }

}
}
```

Output:

```
        You can access this here
        You can access this there
        Even here!
```

view source code here

4. Are some variables passed by value while others are passed by reference? Which ones are which?

All variables in Java are passed by value (Albano, 2017). To illustrate this in more detail let us look at the specifics of how Java stores variables. Within the Java Virtual Machine are two sections of memory space: stack spaced memory and heap spaced memory (Kumar, 2020). Java's stack spaced memory is referenced in a Last-In-First-Out type of manner, so whenever a method is invoked within the code a new block is created into the stack. What is stored within its heap spaced memory are objects, and these objects have global access within the program (Kumar). When a stack is formed and creates any changes in the heap memory. These changes can happen to existing objects because these values within the methods are copied into the heap memory.

5. If you run this code (or the equivalent) in your language, what is the output? What does that tell you about how the language handles assignments?

The output of the following in Java is "cat, dog, dog, and dug." It tells me that variables within the same scope can reassign each other. The output also illustrates the fact that it is possible to reassign or modify a declared variable. I came to this conclusion because `char a` printed the same output as `char b` once `a` was assigned to be equal to `b`, this was also after the original declaration of `char a`. I know that the modification of a variable was also possible in Java because [ith] value of an array, in this case [1], modified `char b` from dog to dug when `b[1]` was reassigned 'u'.

```
public class Main{

     public static void main(String []args){
            char [] a = {'c','a','t'};
            char [] b = {'d','o','g'};
            System.out.println(a); //print cat
            System.out.println(b); //print dog
            a=b;
            System.out.println(a);//print dog
            b[1] = 'u';
            System.out.println(b);//print dug
     }
}
```

```
Output:
        cat
        dog
        dog
        dug
```

view source code here

Works Cited

Albano, J. (2017, October 20). *Passing by Value vs. Passing by Reference in Java.* Retrieved November 01, 2020, from

https://dzone.com/articles/pass-by-value-vs-reference-in-java

Geeks for Geeks. (2020). *Scope of Variables In Java*. Retrieved October 29, 2020, from

https://www.geeksforgeeks.org/variable-scope-in-java/

Java Global Variable: Declaration & Examples (n.d.) Retrieved October 28, 2020, from

https://study.com/academy/lesson/java-global-variable-declaration-examples.html#:~:text=A%20global%20varia-ble%20is%20one,be%20part%20of%20a%20class

Kumar, P. (2020, July). *Java Heap Space vs Stack – Memory Allocation in Java*. Retrieved November 02, 2020, from

https://www.journaldev.com/4098/java-heap-space-vs-stack-memory

Vrajitoru, D. (2020). *Name, Scope, Binding*. Retrieved October 28, 2020, from

https://www.cs.iusb.edu/~danav/teach/c311/c311_3_scope.html

w3schools. (n.d.). *Java Keywords*. Retrieved October 30, 2020, from

https://www.w3schools.com/java/java_ref_keywords.asp