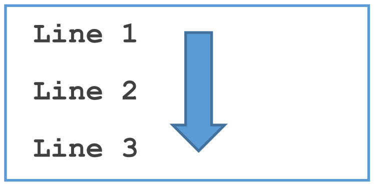
CS330: Programming Language Project (PLP)

Assignment 4: Control Flow

Control Flow is the order in which a computer executes the statements of a program, in Java there are three basic types of control flow structures (Control Structures).

Sequential: Is the default form of control flow in a program for Java. It executes statements of code one line after another in how it is written within the source code (Control Structures). This is the simplest form of control structure in Java. The illustration below is an example of how a sequential statement structure would function.



Sequential structure format:

```
//all lines of code will be read from top to bottom
Code line 1
Code line 2
Code line 3
```

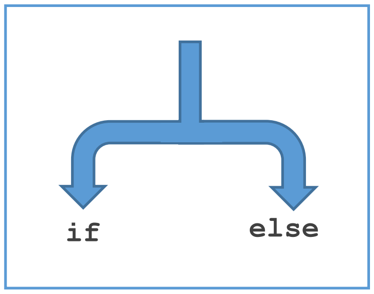
Sequential structure example:

```
public class Sequential {
    public static void main(String[] args) {
    //notice how every line of code is read from top to bottom without
    // any interruptions nor breaks

    Scanner myObj = new Scanner(System.in);  //Creates a Scanner object
    System.out.println("Enter your name");

    String name = myObj.nextLine();  // Read name input
    System.out.println("hello world my name " + name);
    // Output string + name input
  }
```

Selection: In order to alter the flow from its default sequential order of execution, a selection type of structure would "branch out" to certain statements, which happens if the state of a certain variable(s) match the desired condition. In this case a Boolean expression (i.e. true or false) would determine if the desire condition is met (**Control Structures**). If the condition is true then Java would continue unto the statement, if false Java would move onto the next line of code. The possible statements Java may or may not execute based on the met conditions are called branches (Downey& Mayfield, 2016, p.59). In Java there are three types of selection statements: *single selection, double selection,* and *multiple selection.*



These are the typical formats of the three types of selection statements:

Single Selection (if-statement):

```
if (boolean expression)
     [statement] //statement executed if true
```

Double Selection (if-else statement):

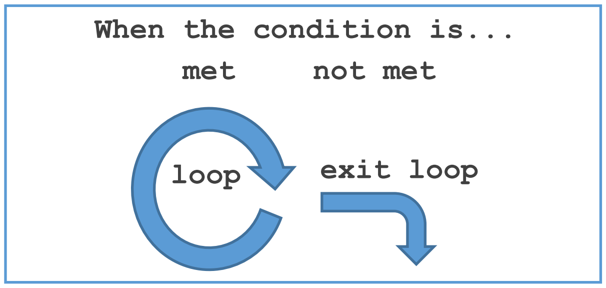
```
if (boolean expression)
    statement  //statement executed if true
else
    statement  //statement executed if false
```

Multiple Selection (switch-case statement):

```
switch (integral expression ) {

    case  integral expression :
        statement(s)
        break;
    case  integral expression :
        statement(s)
        break;
    case  integral expression :
        statement(s)
        break; //break is optional if there is a default
    default:
            statement(s)
}
```

Repetition: A repetition structure, also known as a loop, is a block of code that is repeated multiple times. It is similar to selection structure by the way certain conditions are needed to divert from the sequential structure (in this case, enter the loop) however it differs from selection structure by the way it continuously executes or exits the loop (**Control Structures**). Within each loop iteration the program examines whether or not the conditions are satisfied to either repeat or exit the loop. In Java there are three types of repetition statements: *while repetition statement, do...while repetition statement,* and *for repetition statement* (Deitel, Deitel, & Deitel, 2014).



These are the typical formats of the three repetition loops

while repetition statement (while loop):

```
while (boolean_expression)
//condition is met using if the Boolean expression is true
{
  [Block of code]
}
```

do-while repetition statement (do-while loop):

```
do
{
  [Block of code] //this block of code is excuted at least once
}
while (boolean_expression);
//condition is met using if the Boolean expression is true
```

for repetition statement (for loop):

```
for (initialCondition; boolean_Expression; iterativeStatement)
 {
   [Block of code]
  //condition is met using if the Boolean expression is true
  //Block of code is iterated i number of times
 }
```

Discussion Questions:

1. What types of conditional statements are available in your language?

In Java most of our conditional statements are typically the three selection statements stated earlier: if statements, if-else statements, and switch case statements.

if statement:

The if statement is the simplest conditional statement in Java. As stated earlier, if the condition is seen as true, the statement within the braces would be executed, if the condition is false the statement in the braces would be skip.

```
int number = 1;
// checks if number is greater than 0
if (number > 0) {
    System.out.println("The number is positive.");
}
```

View entire source code here
(this is a modified from the source)

if-else statement:

The if else statement is another type of conditional statement with multiple possible statements to branch into. The current condition of the statement at the time would determine which one of the statements would be executed. What makes this different from the if statement is that if the condition is false, there would still be a statement to execute rather than skip over.

```
int number = 0;
// checks if number is greater than 0
if (number > 0) {
    System.out.println("The number is positive.");
}
//By default 0 or negative
else {
    System.out.println("The number is 0 or negative")
```

View entire source code here

switch-case statement:

The switch-case statement can be described as a multi-way branch statement because there are multiple cases to choose from based on the value of the expression (GeeksforGeek, 2019). The expression can be these type of primitive datatypes: byte, short, char, and int (GeeksforGeek, 2019). It differs from if and if-else statement because of the multiple paths of execution. The statements that would be executed are the ones that fulfill the desire conditions.

```
int favcuisine = 4;
String cuisine;

// switch statement to check day
switch (favcuisine) {
    case 1:
        cuisine = "Mexican";
        break;
            case 2:
            cuisine = "Chinese";
            break;
        case 3:
            cuisine = "Thai";
            break;
        case 4: // matched value of Cuisine
            cuisine = "Indian";
            break;
        default:
            cuisine = "Invalid Cuisine";
            break;
}
System.out.println("Eliana's favourite Cuisine is " + cuisine);
```

View entire source code here

2. Does your language use short-circuit evaluation? If so, make sure that your code includes an example.

Short-circuit evaluation is the evacuation of a partial evaluation of a logical expression. As an example, if a student named Java had two options in passing a class: going to class or doing the assignments, Java would deemed going to class as a way of passing and ignored the assignments portion of the class, in that manner is how Java would also typically handle a short circuit evaluation. The reason a short circuit evaluation would happen is because the result of the evaluation is clear prior to the completion of the entire evaluation. A short circuit evaluation would avoid “unnecessary work” (similar to our student named Java) and would “lead to efficient processing” overall (Geeks-forGeek, 2020).

There are two types of operators that would lead to a short circuit evaluation: the AND (&&) operator and the OR (||) operator (GeeksforGeek, 2020).

AND (&&) operator:

In the && operator, if a false occurs *prior to the completion* of the evaluation, the *whole statement itself will always be deemed as false*, regardless of how many trues appear prior to the false or the amount of trues in total of that statement.

```
int number1 = 3, number2 = 4, number3 = 9;
boolean result;
// All expression must be true from result to be true
result = (number1 > number2) && (number3 > number1
// result will be false because (number1 > number2) is false
System.out.println(result);
```

View entire source code here

OR (||) operator:

Within the || operator, if a true occurs *prior to the completion* of the evaluation, the whole statement itself would always be deemed as true. *This statement will always be true regardless* of how many false appear prior to the true statement or the number of falses in comparison to that one true in the statement.

```
int number1 = 3, number2 = 4, number3 = 9;
boolean result;

// At least one expression needs to be true for the result
//to be true
result = (number1 > number2) || (number3 > number1);

// result will be true because (number3 > number1) is true
System.out.println(result);
```

View entire source code here

3. How does your programming language deal with the “dangling else” problem?

The dangling else problem is when the computer views a nested if statement to be ambiguous while interpreting the source-code (Lin). Consider the following statements:

Statement 1:

```
//else is indented with the inner if statement
int x=17;
int y=12;

if(x>5)
    if(y>10)
        System.out.println("First case");
    else
        System.out.println("Second case");
```

View entire source code

Statement 2:

```
//else is indented with the outer if statement
int x=17;
int y=12;

if(x>5)
    if(y>10)
        System.out.println("First case");
    else
        System.out.println("Second case");
```

View entire source code

Java does not pay attention to indentation, so the if statement Java would pick over the other would be the if statement that was called first. This type of ambiguity of having to pick one if statement over the other for the Java compiler (translator), is how the dangling else problem is formed within Java. How this is possible for Java is that the Java compiler would ignore white space characters (i.e. space and tab) and deemed them as insignificant (Cheung, 2019). The Java compiler would only focus on syntactical correctness which is why it would view both statement of code as the same.

The solution to the dangling else would be to add brackets in order for the next if statement to be evaluated because this would then force the complier to notice the second if statement.

Solution:

```
//adding brackets would allow the Java compiler to notice the latest
//if statement
int x=17;
int y=12;

if(x>5){
    if(y>10)
    System.out.println("First case");
    }
else
    System.out.println("Second case");
```

View entire source code

4. Does your language include multiple types of loops? If so, what are they and how do they differ from each other?

Java includes 3 types of loops: while loops, do-while loops, and for loops. And these types of loops are under the repetition structures within Java's control flow. As stated earlier, these types of loops would repeated a block of code multiple times in succession as long as the criteria defined within the loop structure is met. Despite these loops being defined as repetition structures, how each loop implements repetition varies.

while loop:

How a while loop would function is that a block of code would continuously be performed for n amount of times if the conditions of the while loop are met (Downey& Mayfield, 2016, p.89). A condition of a while loop is expressed as met or not met is through a Boolean expressions (i.e. True or False). If the condition of the while loop is true, the block of code would continuously be performed until the condition is false. If the condition of the while loop was false prior to the loop, the blocks of code would not be performed.

```
int x = 1, sum = 0;

// Exit when x becomes greater than 10
while (x <= 10) {
    sum = sum + x; // Total sum of x
    x++; // incrementing the value of x for the next iteration
}
```

View entire source code here

do-while loop:

In the case of a do-while loop, the block of code could be executed at least one time (Downey& Mayfield, 2016, p.97). This is because the while loop is underneath the block of code, and would have declare to either repeat the block of code or not depending if the conditions are met or not.

```
int x = 1, sum = 0;

do {
    sum = sum + x;
    x++; // incrementing the value of x for the
         //next iteration
}
// Exit when x becomes greater than 10
while (x <= 10);
```

View entire source code here

for loop:

The for loop can be also expressed as a counting loop, because it is a loop that would rely on a counting variable (Downey& Mayfield, 2016, p.96). This counting variable is typically known as the amount of iterations, or repetitions, for the block of code. The for loop differs from both of the while and do while statements because it has a defined termination point. The termination point can be considered the point where the conditions of the loop are not met, and the program would continue onto the next lines of code in a sequential manner.

```
for(count = 1; count <= num; count++){
    total = total + count;
    }
System.out.println("Sum of numbers from 1 to 10 is: "+total);
```

View entire source code here

5. Can you use break or continue statements (or something similar) to exit loops?

The Java program utilizes both break or continue statements to exit loops of the program.

break:

In Java, a break statement would be a hard exit out of the loop and the program would then follow unto the next lines of code in a sequential manner (Perkins, 1999). How a break statement is typically executed is if it meets the conditions within one of Java's selection structures (such as a nested if statement), if the condition is met for the statement to break then Java would then immediately terminate the loop and continue on to the follow lines of code.

```
for (int i = 1; i <= 10; ++i) {

    // if the value of i is 8 the loop terminates
    if (i == 8) {
        break;
    }
```

View entire source code here

continue:

A continue statement would be a way to force an earlier iteration of the loop rather than just an immediate termination of the loop (Perkins, 1999). Similarly to how a break statement is executed, if the conditions for a continue statement are met within one of Java's selection structures (such as a nested if statement), then Java would force out of the current iteration of the loop, and remain within the loop until it is complete.

```
while (i <=10) {
// Since i=4, it will skip 4 and print everything else
    if (i == 4) {
    i++;
    continue;
  }
```

View entire source code here

6. If your language supports switch or case statements, do you have to use “break” to get out of them? Can you use “continue” to have all of them evaluated?

   In the case of a switch statement, Java could only break out of them. Utilizing break in a switch statement would terminate the statement. If no break occurs, the flow of control will flow through each case once Java runs into a break in the program (GeeksforGeeks, 2019). A break statement is however optional if there is a *default case* at the end of the switch (GeeksforGeeks, 2019).

   The continue statement however does not work well with the switch statement since it works best with forcing the program out of the current iteration unto the next incremental step (Perkins, 1999). Continue statements typically function well if only it is applied to loops, anything outside of a loop would cause this error:

   [*continue outside of loop*]

7. Is there anything special in terms of control flow that your language does that isn't addressed in this assignment? If so, what is it and how does it work? Make sure to include an example of it in your code as well.

   One control flow term that hasn’t been addressed in Java is the return statement. It is important to note there are two types of forms of return methods: one that returns a value and one to stop a program execution. In the case of control flow we will go over the latter. One reason someone might want to utilize this method is due to a detected an error condition (Downey& Mayfield, 2016, p.62).

```java
boolean gone = true;
System.out.println("You will see this!");
if (gone)
    return;
// The Java complier will skip every statement now
System.out.println("You won’t see this! Secret.");
```

View entire source code here

Works Cited

Cheung, Shun Yan (2019, October 26). *The dangling-else ambiguity*. Retrieved October 04, 2020, from http://www.mathcs.emory.edu/~cheung/Courses/170/Syllabus/06/if4.html

Control Structures. (n.d.). Retrieved October 03, 2020, from https://courses.cs.vt.edu/csonline/ProgrammingLanguages/Lessons/ControlStructures/index.html

Deitel, P., Deitel, H., & Deitel, A. (2014). Android™ How to Program, Second Edition. Retrieved October 6, 2020, from https://www.oreilly.com/library/view/androidtm-how-to/9780133802092/app03lev2sec2.html

Downey, A. B., & Mayfield, C. (2016). *Think Java*. Sebastopol, CA: O'Reilly Media.

GeeksforGeeks (2019, October 01). Switch Statement in Java. Retrieved October 05, 2020, from https://www.geeksforgeeks.org/switch-statement-in-java/?ref=lbp

GeeksforGeeks. (2020, May 01). *Short Circuiting in Java with Examples*. Retrieved October 04, 2020, from https://www.geeksforgeeks.org/short-circuiting-in-java-with-examples/

Lin, C. (n.d.). *Incremental Java: Dangling else*. Retrieved October 03, 2020, from https://www.cs.umd.edu/~clin/MoreJava/ControlFlow/dangling.html

Perkins, H. (1999). Control Flow Statements. Retrieved October 06, 2020, from https://courses.cs.washington.edu/courses/cse341/99wi/java/tutorial/java/nutsandbolts/while.html