AP COMPUTER SCIENCE
JAVA CONCEPTS V: FLOW CONTROL STATEMENTS

PAUL L. BAILEY

1. Flow Control Overview

1.1. **Flow Control Keywords.** A subset of the Java keywords are used inside blocks of code to control the flow of execution. We call these *flow control* keywords, and they are:

```
break       do          if
case        else        return
continue    for         switch
default     goto        while
```

These keywords are combined to make various types of flow control statements.

1.2. **Boolean Expressions.** A *boolean expression* is an expression that evaluates to a boolean data type. This occurs when the last operator to be evaluated is a boolean operator.

There are two types of boolean operators we would like to consider: *relational* and *logical*.

Type	Operator	Function	Inputs
Relational	`==`	Equal to	All
Relational	`!=`	Not equal to	All
Relational	`<=`	Less than or equal to	Numeric
Relational	`<`	Less than	Numeric
Relational	`>=`	Greater than or equal to	Numeric
Relational	`>`	Greater than	Numeric
Logical	`!`	Not (unary)	Boolean
Logical	`||`	Or	Boolean
Logical	`&&`	And	Boolean

Date: September 21, 2019.

2. Decision Making Statements

2.1. **if statement.** The general format of an if statement is

```
if (expression)
{
    statement(s)
}
```

The if statement IS part of the AP Java subset.

Expression must evaluate to a boolean value. If the expression is true, the statement or block of code is executed; otherwise, the block of code is skipped. The block of code may be replaced by a single statement.

2.2. **if-else statement.** The general format of an if statement is

```
if (expression)
{
    statement(s)
}
else
{
    statement(s)
}
```

The if-else statement IS part of the AP Java subset.

Expression must evaluate to a boolean value. If the expression is true, the statement or block of code following `if (expression)` is executed; otherwise, the block of code following `else` is executed. Either block of code may be replaced by a single statement.

Example:

```
// Example if-else statement
// Return the absolute value of a difference

if (a < b)
{
    m = b - a;
}
else
{
    m = a - b;
}
```

2.3. **switch statement.** The general format of the switch statement is

```
switch (value)
{
    case v1 :
        statement(s)
    case v2 :
        statement(s)
    ...
    default :
        statement(s)
}
```

The switch statement IS NOT part of the AP Java subset.

The `value` can be of one of the "switchable" types, which include byte, short, char, int, and String. The value is tested against each case. If the switch value equals the case value, the corresponding statements are executed. The statements of subsequent cases are also executed, until a `break` statement is encountered. When `break` is encountered, execution resumes after the closing brace of the `switch` statement.

If none of the cases are satisfied, the optional `default` statements are executed.

```
\\ Example switch statement
\\ Convert a number to the name of the corresponding month

String s;
switch (m)
{
    case 1: s = "January"; break;
    case 2: s = "February"; break;
    case 3: s = "March"; break;
    case 4: s = "April"; break;
    case 5: s = "May"; break;
    case 6: s = "June"; break;
    case 7: s = "July"; break;
    case 8: s = "August"; break;
    case 9: s = "September"; break;
    case 10: s = "October"; break;
    case 11: s = "November"; break;
    case 12: s = "December"; break;
    default: s = "Unknown";
}
```

3. Looping Statements

3.1. **while statement.** The general format of the while statement is

```
while (expression)
{
    statement(s)
}
```

The while statement IS part of the AP Java subset.

Expression must evaluate to a boolean value. If the expression is false, the block of code is skipped. If the expression is true, the block of code is executed, after which the expression is evaluated again. This continues until the expression evaluates to false.

```
// Example while statement
// Compute the sum of the first 10 positive integers

int s = 0;
int k = 0;
while (k <= 10)
{
    s += k;
    k += 1;
}
```

3.2. **do-while statement.** The general form of a do-while statement is

```
do
{
    statement(s)
}
while (expression);
```

The do-while statement IS NOT part of the AP Java subset.

The block of code is executed, and then the expression is evaluated. If expression is true, the block is executed again. This continues until expression is false.

```
// Example do-while statement
// Compute the sum of the first 10 positive integers

int s = 0;
int k = 1;
do
{
    s += k;
    k += 1;
}
while (k <= 10);
```

3.3. **for statement.** The general form of a for statement is

```
for (initialization; termination; increment)
{
    statement(s)
}
```

This is equivalent to

```
initialization;
while (termination)
{
    statement(s)
    increment;
}
```

The for statement IS part of the AP Java subset.

Initialization, termination, and increment are expressions.

- The initialization expression initializes the loop; it's executed once, as the loop begins.
- When the termination expression evaluates to false, the loop terminates.
- The increment expression is invoked after each iteration through the loop.

```
// Example for statement
// Compute the sum of the first 10 positive integers

int s = 0;
int k = 1;
for (k = 1; k <= 10; k++)
{
    s += k;
}
```

3.4. **for each statement.** This is an alternate form of use of the `for` keyword, and is referred to in the AP Java subset as "enhanced for". It allows the programmer to iterate through an array or collection with less code. The syntax is

```
for (type variable : collection)
{
    statement(s)
}
```

The for each statement IS part of the AP Java subset.

For example, the code

```
char array = { 'H', 'E', 'L', 'L', 'O' };
for (int i = 0; i < arr.length; i++)
{
    char letter = array[i];
    System.out.println(letter);
}
```

can instead be written using a for each statement as

```
char array = { 'H', 'E', 'L', 'L', 'O' };
for (char letter : array)
{
    System.out.println(letter);
}
```

When using a for each statement, the collection should not be modified within the body of the loop.

4. Branching Statements

4.1. **return statement.** The general form of the return statement is

```
return value;
```

The return statement IS part of the AP Java subset.

The return statement exits a method and returns the value to the calling method. If the method in which the return is found is void, then the value is absent. A return statement will terminate any loops in which it occurs, and exit the method.

```
\\ Example return statement
\\ Find the remainder by brute force

while (n > m)
{
    n = n - m;
}
return n;
```

A return may be placed inside the body of a loop; this will terminate the loop, and will exit the method.

4.2. **break statement.** The break statement is

```
break;
```

The break statement IS NOT part of the AP Java subset.

The break statement can be used to terminate switch, while, do-while, and for statements. When `break` is encountered, execution resumes after the closing brace of the enclosing switch, while, do-while, or for statement.

```
// Example break statement
// Divide a number by leading nonzero entries in an array

int number = 12345;
while (index < array.length)
{
    if (array[index] == 0)
    {
        break;
    }
    number = number / array[index];
}
```

There is also a "labeled break" statement in Java. This does not exist in C, C++, or C#, and is not in the AP Java subset, so we skip it here.

4.3. **continue statement.** The continue statement is

```
continue;
```

The continue statement IS NOT part of the AP Java subset.

The continue statement can be used inside a loop to skip the rest of the block, but continue looping. When the `continue` is encountered, execution resumes at the top of the loop.

```
\\ Example continue statement
\\ Divide a number by ALL nonzero entries in an array

int number = 12345;
while (index < array.length)
{
    if (array[index] == 0)
    {
        continue;
    }
    number = number / array[index];
}
```

Department of Mathematics, BASIS Scottsdale
Email address: paul.bailey@basised.com