Conditional Logic, Looping, and Arrays

What to Expect in This Module

- Conditional logic
- Basic looping
- Arrays
- For-each loop
- The switch statement

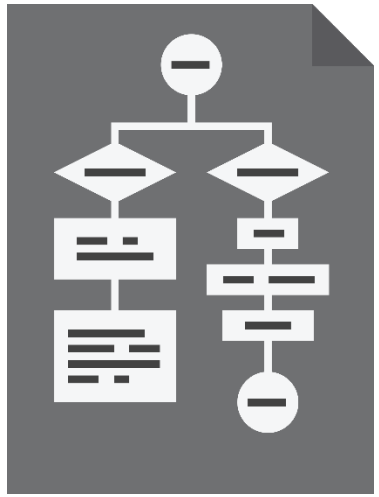
Conditional Logic

Relational operators

Conditional assignments

The if statement

Logical operators



Relational Operators

	Operator	Integer, Floating Point Example	Character Example	Boolean Example
Greater than	>	5 > 4	'c' > 'a'	*not available*
Greater than or equal to	>=	5 >= 4 4 >= 4	'c' >= 'a' 'a' >= 'a'	*not available*
Less than	<	4 < 5	'a' < 'c'	*not available*
Less than or equal to	<=	4 <= 5 4 <= 4	'a' <= 'c' 'a' <= 'a'	*not available*
Equal to	==	5 == 5	'c' == 'c'	true == true false == false
Not equal to	!=	4 != 5	'a' != 'c'	true != false

Conditional Assignment

- Assign a value to a variable based on the result of a condition

result = *condition* ? *true-value* : *false-value*;

```
int v1 = 7;
int v2 = 5;

int vMax = v1 > v2 ? v1 : v2;
System.out.println(vMax);
```

7

```
float students = 30;
float rooms = 4;

float studentsPerRoom = rooms == 0.0f ? 0.0f : students/rooms;
System.out.println(studentsPerRoom);
```

7.5

If-else Statement

- An if statement conditionally executes a statement

```
if (condition )
true-statement ;
else
  false-statement ;
```

- The optional else clause executes a statement when the if condition is false

```
int v1 = 10;
int v2 = 4;

if(v1 > v2)
  System.out.println("v1 is bigger");
else
  System.out.println("v1 is NOTbigger");
```

Chaining if-elseStatements

- If-else statements chained together are evaluated in order until one is true

```
if (condition-1 )
   true-statement-1 ;
else  if (condition-2 )
true-statement-2  ;
          .
          .
          .
else  if (condition-N )
true-statement-N  ;
else
  false-statement  ;
```

```
int v1 = 10;
int v2 = 40;

if(v1 > v2)
   System.out.println("v1 is bigger");
else if(v1 < v2)
   System.out.println("v2 is bigger");
else
   System.out.println("v1 and v2 are equal");
```

Block Statements

- A block statement groups statements into a compound statement

```
{
   statement-1;
   statement-2;
         .
         .
         .
   statement-N;
}
```

```
int v1=10, v2=4, diff;
if(v1 >v2) {
   diff =v1 – v2;
   System.out.println(“v1 is bigger”);
   System.out.println(diff);
}
else if(v2 >v1){
   diff =v2 – v1;
   System.out.println(“v2 is bigger”);
   System.out.println(diff);
}
else
   System.out.println(“v1 and v2 are equal”);
```

Block Statements and Variable Scope

- A variable declared within a block is not visible outside of the block
  - A variable's range of visibility is known as the variable's *scope*

```
float students =30.0;
float rooms = 4.0;

if(rooms >0.0) {
   System.out.println(students);
   System.out.println(rooms);
   float avg =students / rooms;

}

System.out.println(avg);
```

Logical Operators

	Operator	What Resolves to True
And	&	true & true
Or	|	false | true, true | false, true | true
Exclusive or (XOR)	^	false ^ true, true ^ false
Negation	!	false

```
int a = 20, b = 14, c = 5;
            true
      true      true
if (a > b && b > c)
  System.out.println("a is greater than c");
```

```
boolean done = false;
        true
if (!done)
  System.out.println("Keep going");
```

Conditional Logical Operators

	Operator	What Resolves to True
Conditional and	&&	true && true
Conditional or	||	false || true true || ----

- Resolve following conceptually similar rules as non-conditional and/or
- Only execute the right-side if needed to determine the result
  - && only executes right-side if left-side is true
  - || only executes right-side if left-side is false

Looping

While loop

Do-while loop

For loop

While Loop

- Repeatedly executes a statement as long as the condition is true
  - Condition checked at loop start
  - Statement may never execute

while (*condition*)
statement ;

```
int kVal = 5;
int factorial = 1;

while(kVal >1)
   factorial *= kVal--;

System.out.println(factorial);
```

```
int kVal = 5;
int factorial = 1;

while(kVal >1) {
   factorial *= kVal;
   kVal -= 1;

}
System.out.println(factorial);
```

Do-while Loop

- Repeatedly executes a statement as long as the condition is true
  - Condition checked at loop *end*
  - Statement always executes at leastonce

```
do
   statement ;
while (condition ) ;
```

```
int iVal = 155;0;
do {
   System.out.print(iVal);
   System.out.print(" * 2 =");
   iVal *= 2;
   System.out.println(iVal);
} while(iVal <100);
```

```
5 * 2 = 10
10 * 2 = 20
20 * 2 = 40
40 * 2 = 80
80 * 2 = 160
```

```
150 * 2 = 300
```

For Loop

- Repeatedly executes a statement as long as the condition is true
  - Condition checked at loop start
  - Provides simplified notation for loop control values

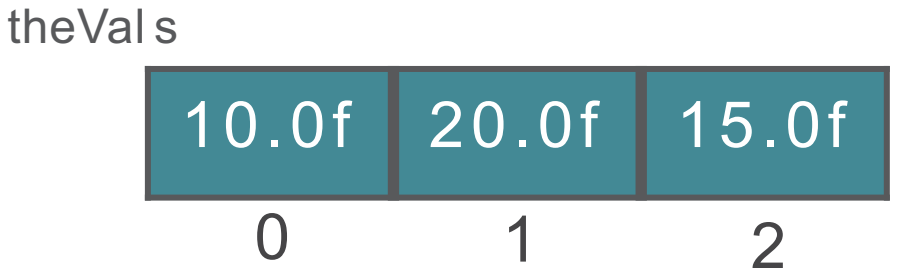
```
for (initialize ; condition ; update )
    statement ;
```

```
int iVal =1;
while(iVal <100){
   System.out.println(iVal);
   ival *= 2;
}
```

```
for( int iVal = 1; iVal < 100; iVal *= 2) {
    System.out.println(iVal);
}
```

Arrays

- Provides an ordered collection of elements
  - Each element accessed via an index
  - Indexes range from 0 to number-of-elements minus 1
  - Number of elements can be found via array's *length* value



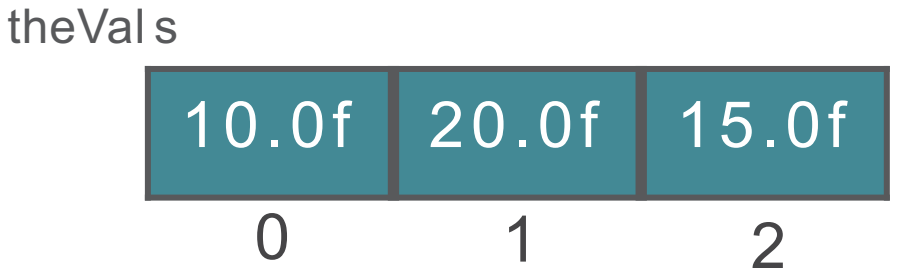

```
float[] theVals = new float[3];
theVals[0] = 10.0f;
theVals[1] = 20.0f;
theVals[2] = 15.0f;

float sum = 0.0f;
for(int i =0; i <theVals.length; i++)
   sum+= theVals[i];

System.out.println(sum);
```

Arrays

- Provides an ordered collection of elements
  - Each element accessed via an index
  - Indexes range from 0 to number-of-elements minus 1
  - Number of elements can be found via array's *length* value




```
float[] theVals = { 10.0f, 20.0f, 15.0f };




float sum = 0.0f;
for(int i =0; i <theVals.length; i++)
   sum+= theVals[i];

System.out.println(sum);
```

For-each Loop

- Executes a statement once for each member in an array
  - Handles getting collection length
  - Handles accessing each value

for (*loop-variable-declaration* :*array*)
    statement ;

```
float[] theVals ={ 10.0f, 20.0f, 15.0f };
float sum = 0.0f;
for( float currentVal: theVals){
   sum += currentVal;
}
System.out.println(sum);
```

Switch

- Transfers control to a statement based on a value
  - Simplifies testing against multiple possible matches
  - Only primitive types supported are char and integers
  - A match can execute more than one statement
  - Use break to avoid “falling through”
  - Can optionally include default to handle any unmatched values

```
switch (test-value ) {
    case value-1:
        statements
    case value-2:
        statements
    .
    .
    .
    case value-n:
        statements
    default:
        statements
}
```

Switch Example

```
int iVal = 2150;
switch(iVal % 2) {
  case 0:
     System.out.print(iVal);
     System.out.println(" is even");
     break;
 case 1:
    System.out.print(iVal);
    System.out.println(" is odd");
    break;
 default:
    System.out.println("oops it broke");
    break;
}
```

```
10 is even
10 is odd
oops it broke
```

```
10 is even
```

```
25 is odd
```

Summary

- Use the if-else statement to provide conditional logic
  - If-else statements can be chained together
- Block statements use brackets to group statements
  - Variables declared within a block are not visible outside of the block
- Both while and do-while loops execute as long as a condition is true
  - The do-while loop body always executes at least once
- The for loop provides simplified notation for loop initialization and control
- For-each statement handles details of executing once for each array member
- Switch statement simplifies notation of testing against multiple matches