

Java – Control Statement Introduction to Class

SE 206

Control Statements

- Java's control statements are nearly identical to those in C/C++.
- There are a few differences – especially in the **break** and **continue** statements.
- The control statements discussed here:
 - if
 - if - else
 - if - else if
 - switch
 - ?
 - while
 - for

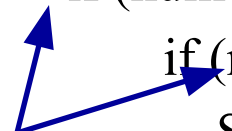
if statement

- What does the following do?

```
Scanner stdin = new Scanner(System.in);
System.out.print("Enter an integer number: ");
int value1 = stdin.nextInt();
System.out.print("Enter another integer number: ");
int value2 = stdin.nextInt();
if (value2 < value1) {
    int rememberValue1 = value1;
    value1 = value2;
    value2 = rememberValue1;
}
System.out.println("The numbers in sorted order are "
+ value1 + " and then " + value2);
```

If-then-else precedence

if (number != 0)
if (number > 0)
System.out.println("positive");
Which if does this
else refer to? else
System.out.println("negative");



If-else-if

□ Consider

```
if (number == 0) {  
    System.out.println("zero");  
}  
else if (number > 0) {  
  
    System.out.println("positive");  
}  
else {  
    System.out.println("negative");  
}
```

Finding the minimum value using “?” notation

- Consider:

// z is to hold the minimum of x and y

if (x < y)

z = x;

else


z = y;

Notice no braces!



- Another way to do this:

z = (x < y) ? x : y;



The ?: notation

- Only works when both “cases” return a value!
 - Meaning when both “cases” are expressions
 - Example: $z = (x < y) ? x : y;$
 - Thus, you can’t put a print statement in there!
- Can be difficult to read

```
System.out.println ((number != 0) ? ((number > 0) ? "positive" :  
    "negative") : "zero");
```

```
if (number != 0)  
    if (number > 0)  
        System.out.println("positive");  
    else  
        System.out.println("negative");  
else  
    System.out.println("zero");
```

A switch statement example

```
if (a == '0')
    System.out.println ("zero");
else if (a == '1')
    System.out.println ("one");
else if (a == '2')
    System.out.println ("two");
else if (a == '3')
    System.out.println ("three");
else if (a == '4')
    System.out.println ("four");
else
    System.out.println ("five+");
```

```
switch (a) {
    case '0':
        System.out.println ("zero");
        break;
    case '1':
        System.out.println ("one");
        break;
    case '2':
        System.out.println ("two");
        break;
    case '3':
        System.out.println ("three");
        break;
    case '4':
        System.out.println ("four");
        break;
    default:
        System.out.println ("five+");
        break;
```

```
}
```


Why use Switch statement

- The task is often more readable with the switch than with the if-else-if

Testing for vowel-ness

```
switch (ch) {  
    case 'a': case 'A':  
    case 'e': case 'E':  
    case 'i': case 'I':  
    case 'o': case 'O':  
    case 'u': case 'U':  
        System.out.println("vowel");  
        break;  
    default:  
        System.out.println("not a vowel");  
}
```

← The break causes an exiting of the switch

↖ Handles all of the other cases

Java looping

- Options
 - while
 - do-while
 - for

- Allow programs to control how many times a statement list is executed

Averaging

- Problem
 - Extract a list of **positive numbers** from **standard input** and produce their **average**
 - Numbers are **one per line**
 - A **negative number** acts as a *sentinel* to indicate that there are **no more numbers** to process

- **Sample run**

Enter positive numbers one per line.

Indicate end of list with a negative number.


4.5

0.5

1.3

-1

Average 2.1



```

public class NumberAverage {
    // main(): application entry point
    public static void main(String[] args) {
        // initialize variables
        int valuesProcessed = 0;
        double valueSum = 0;
        // set up the input
        Scanner stdin = new Scanner (System.in);
        // prompt user for values
        System.out.println("Enter positive numbers 1 per line.\n"
            + "Indicate end of the list with a negative number.");
        // get first value
        double value = stdin.nextDouble();
        // process values one-by-one
        while (value >= 0) {
            // add value to running total
            valueSum += value;
            // processed another value
            ++valuesProcessed;
            // prepare next iteration - get next value
            value = stdin.nextDouble();
        }
        // display result
        if (valuesProcessed > 0){
            // compute and display average
            double average = valueSum / valuesProcessed;
            System.out.println("Average: " + average);
        }
        else{
            // indicate no average to display
            System.out.println("No list to average");
        }
    }
}

```

for vs. while

- A for statement is almost like a while statement

```
for ( ForInit; ForExpression; ForUpdate ) Action
```

is ***ALMOST*** the same as:

```
ForInit;  
while ( ForExpression ) {  
    Action;  
    ForUpdate;  
}
```

- This is not an absolute equivalence!
 - We'll see when they are different below

Variable declaration

- You can declare a variable in any block:

```
while ( true ) {  
    int n = 0;  
    n++;  
    System.out.println (n);  
}  
System.out.println (n);
```

Variable n gets created
(and initialized) each time

Thus, println() always
prints out 1

Variable n is not
defined once while
loop ends

As n is not defined
here, this causes
an error

for vs. while

- An example when a for loop can be directly translated into a while loop:

```
int count;  
for ( count = 0; count < 10; count++ ) {  
    System.out.println (count);  
}
```

- Translates to:

```
int count;  
count = 0;  
while (count < 10) {  
    System.out.println (count);  
    count++;  
}
```


for vs. while

- An example when a for loop CANNOT be directly translated into a while loop: only difference

```
for ( int count = 0; count < 10; count++ ) {  
    System.out.println (count);  
}
```

- Would (mostly) translate as: count is **NOT** defined here

```
int count = 0;  
while (count < 10) {  
    System.out.println (count);  
    count++;  
}
```

count **IS** defined here

Nested loops

```
int m = 2;
int n = 3;
for (int i = 0; i < n; ++i) {
    System.out.println("i is " + i);
    for (int j = 0; j < m; ++j) {
        System.out.println("  j is " + j);
    }
}
```

do-while: Picking off digits

□ Consider

```
System.out.print("Enter a positive number: ");  
int number = stdin.nextInt();  
do {  
    int digit = number % 10;  
    System.out.println(digit);  
    number = number / 10;  
} while (number != 0);
```

□ Sample behavior

```
Enter a positive number: 1129  
9  
2  
1  
1
```

while vs. do-while

- If the condition is false:
 - while will not execute the action
 - do-while will execute it once

```
while ( false ) {  
    System.out.println ("foo");  
}
```

never executed



```
do {  
    System.out.println ("foo");  
} while ( false );
```

executed once



Loop controls

The continue keyword

- The continue keyword will immediately start the next iteration of the loop
 - The rest of the current loop is *not* executed

```
for ( int a = 0; a <= 10; a++ ) {  
    if ( a % 2 == 0 ) {  
        continue;  
    }  
    System.out.println (a + " is odd");  
}
```

- Output:
1 is odd
3 is odd
5 is odd
7 is odd
9 is odd

The break keyword

- The break keyword will immediately stop the execution of the loop
 - Execution resumes after the end of the loop

```
for ( int a = 0; a <= 10; a++ ) {  
    if ( a == 5 ) {  
        break;  
    }  
    System.out.println (a + " is less than five");  
}
```

- Output: 0 is less than five
1 is less than five
2 is less than five
3 is less than five
4 is less than five