



Chapter 4 - Control Statements

Computer Science AP

Lesson 4: Introduction to Control Statements

Review:

- The increment and decrement operators.
- Standard math methods.
- The if and if-else statements to make choices.
- while and for loops to repeat a process.
- Construct appropriate conditions for control statements using relational operators.
- Detect and correct common errors involving loops.

Lesson 4: Introduction to Control Statements

Vocabulary:

- control statements
- counter
- count-controlled loop
- flowchart
- infinite loop
- iteration
- off-by-one error
- overloading
- sentinel
- task-controlled loop

4.1 Extended Assignment Operators

- Plain assignment $=$
- Add, then assign $+=$
- Subtract, then assign $-=$
- Multiply, then assign $*=$
- Divide, then assign $/=$
- Modulus, then assign $\%=$

No space
between
operators

Example: Extended Assignment Operators

```
int z = 25;
```

```
z += 3;           //same as z = z + 3
```

```
z /= 5;           //same as z = z / 5;
```

```
z %= 2;           //same as z = z%2;
```

```
//also applies to String
```

```
String s = "hello";
```

```
s += " world";    //creates "hello world"
```

Increment and Decrement operators

`++` means increase by 1

`--` means decrease by 1

- Works with `int` and `double`

- Most useful for loops

Let's look at the Operator Precedence Chart.

Operator Precedence

Table C-1 shows the operator precedence. The operators shown in bold are not discussed in this book. To learn more about them, see the references on Sun's Web site (<http://www.sun.com>).

TABLE C-1

Operator precedence

OPERATOR	FUNCTION	ASSOCIATION
()	Parentheses	Left to right
[]	Array subscript	
.	Object member selection	
++	Increment	Right to left
--	Decrement	
+	Unary plus	
-	Unary minus	
!	Boolean negation	
~	Bitwise negation	
(type)	Type cast	
*	Multiplication	Left to right
/	Division	
%	Modulus	
+	Addition or concatenation	Left to right
-	Subtraction	
<<	Bitwise shift left	Left to right
>>	Bitwise shift right	
>>>	Bitwise shift right, sign extension	



Open with

OPERATOR	FUNCTION	ASSOCIATION
<	Less than	Left to right
<=	Less than or equal to	
>	Greater than	
>=	Greater than or equal to	
instanceOf	Class membership	
==	Equal to	Left to right
!=	Not equal to	
&	Boolean AND (complete)	Left to right
&	Bitwise AND	
^	Boolean exclusive OR	Left to right
^	Bitwise exclusive OR	
	Boolean OR (complete)	Left to right
	Bitwise OR	
&&	Boolean AND (partial)	Left to right
	Boolean OR (partial)	Left to right
?:	Ternary conditional	Right to left
=	Assign	Right to left
+=	Add and assign	
-=	Subtract and assign	
*=	Multiply and assign	
/=	Divide and assign	
%=	Modulo and assign	
<<=	Shift left and assign	
>>=	Shift right, sign extension, and assign	
>>>=	Shift right, no sign extension, and assign	
&=	Boolean or bitwise AND and assign	
=	Boolean or bitwise OR and assign	

Page 2 / 2

4.2 Methods of the Math class

METHOD	WHAT IT DOES
<code>Static int abs(int x)</code>	Returns the absolute value of an integer x.
<code>static double abs(double x)</code>	Returns the absolute value of a double x.
<code>static double pow(double base, double exponent)</code>	Returns the base raised to the exponent.
<code>static long round(double x)</code>	Returns x rounded to the nearest whole number. (Note: Returned value must be cast to an int before assignment to an int variable.)
<code>static int max(int a, int b)</code>	Returns the greater of a and b.
<code>static int min(int a, int b)</code>	Returns the lesser of a and b.
<code>static double sqrt(double x)</code>	Returns the square root of x.

Some observations on the Math class

- All the methods are static. Static methods belong to a class, not an object. You don't need to instantiate an object to call static methods
- There are two **abs** methods. This is an example of *overloaded* methods. *Same method name in the same class with different parameter lists*

4.2 Example :The sqrt Method

- This code segment illustrates the use of the sqrt method:

```
//compute its radius
// Use the formula  $a = \pi r^2$  , where a is the
// area and r is the radius
double area = 10.0, radius;
radius = Math.sqrt (area / Math.PI);
```

- Messages are usually sent to objects; however, if a method's signature is labeled **static**, the message instead is sent to the method's class.

4.2 The pow and sqrt methods

- The method signatures
 - `static double pow (double base, double exp)`
 - `static double sqrt (double x)`
- Their parameters are both double. If an `int` is used, it is automatically converted to `double` before the method is called.

4.2 The Random class

- A *random number generator* returns numbers chosen at random from a predesignated interval.
- Java's random number generator is implemented in the **Random** class and utilizes the methods **nextInt** and **nextDouble**

```
Random rgen = new Random();  
int num = rgen.nextInt(3); //0, 1, 2  
double dub = rgen.nextDouble(); //0.0 <= dub < 1.0
```

Alternate method to get random

- The method signature

static double Math.random()

- The statement to invoke the method

double dub = Math.random();

- What it does

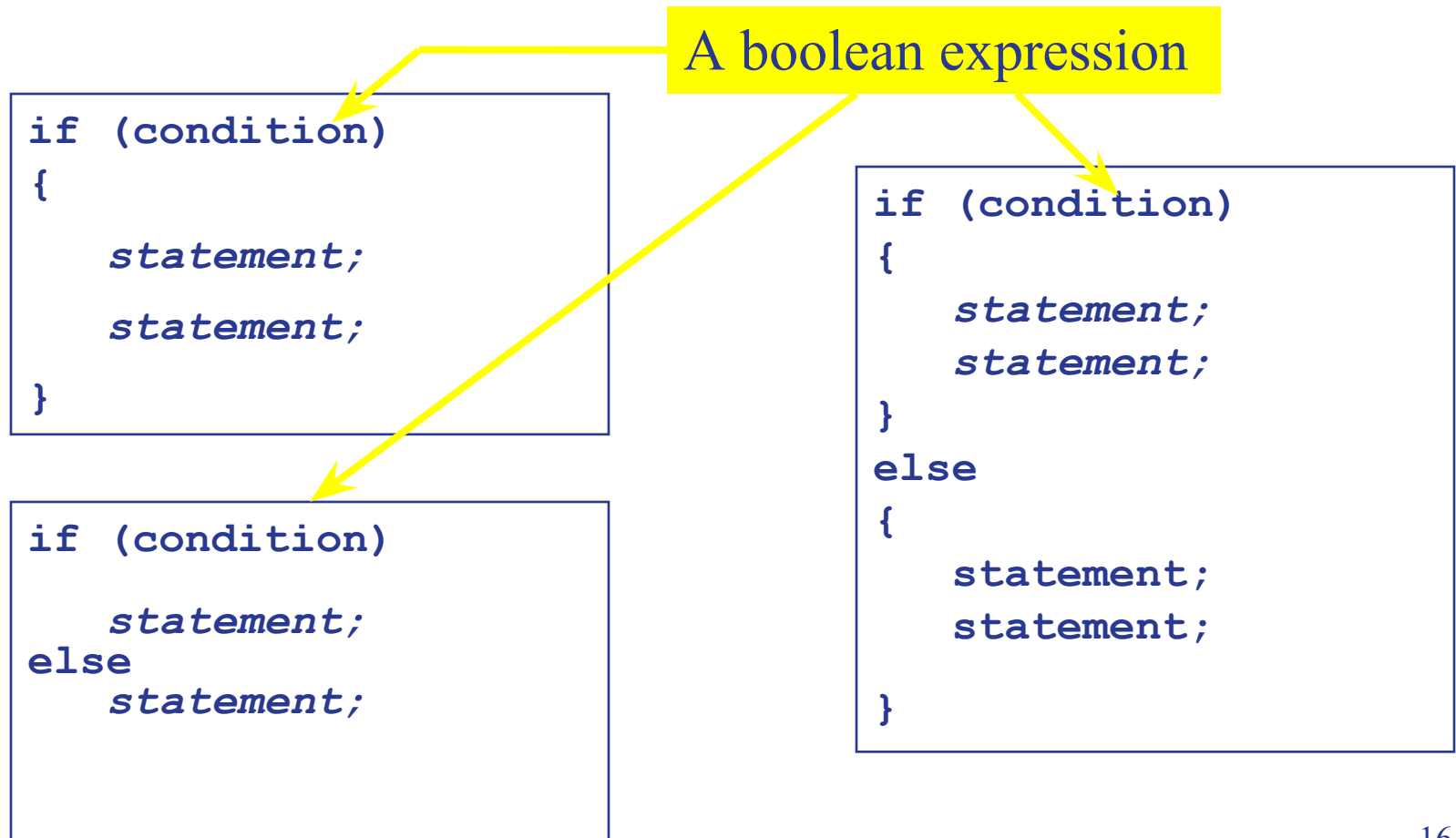
Returns a random double greater than or equal to 0.0 and less than 1.0

Comparing both methods

```
System.out.println ((int) (Math.random() * 10));
```

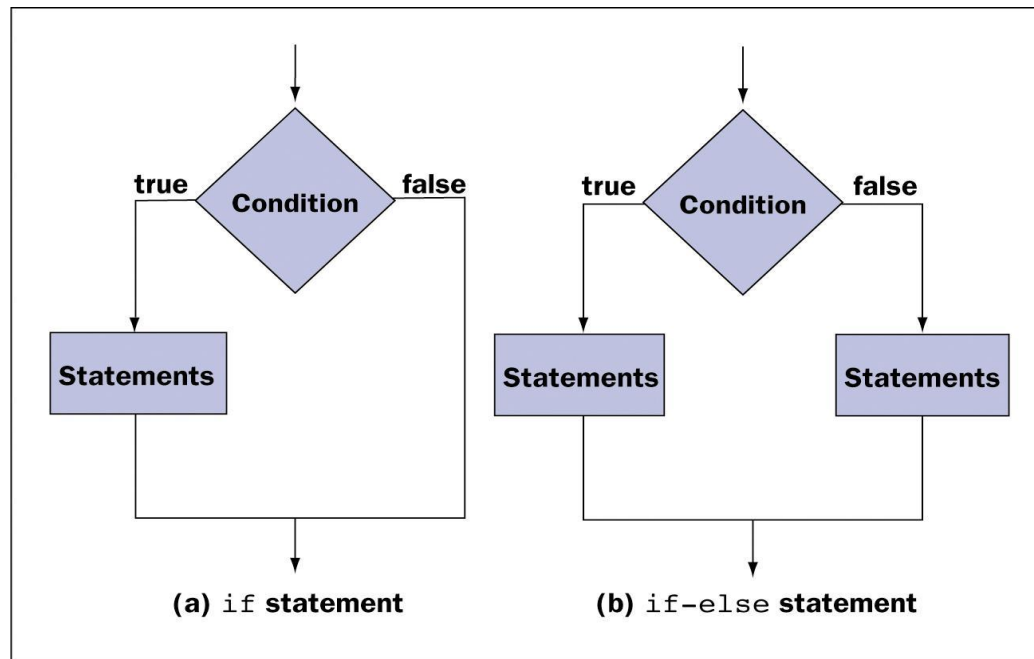
```
Random r = new Random();  
System.out.println (r.nextInt(10));
```

4.4 The If Statement



If-then-else Statement – Flow Chart

- Flowcharts for the `if` and `if-else` statements.



Relational Operators

- Greater than $>$
- Greater than or equal to $>=$
- Less than $<$
- Less than or equal to $<=$
- Equal to $==$
- Not equal to $!=$

Let's look at the Operator Precedence Chart again

Operator Precedence

Table C-1 shows the operator precedence. The operators shown in bold are not discussed in this book. To learn more about them, see the references on Sun's Web site (<http://www.sun.com>).

TABLE C-1

Operator precedence

OPERATOR	FUNCTION	ASSOCIATION
()	Parentheses	Left to right
[]	Array subscript	
.	Object member selection	
++	Increment	Right to left
--	Decrement	
+	Unary plus	
-	Unary minus	
!	Boolean negation	
~	Bitwise negation	
(type)	Type cast	
*	Multiplication	Left to right
/	Division	
%	Modulus	
+	Addition or concatenation	Left to right
-	Subtraction	
<<	Bitwise shift left	Left to right
>>	Bitwise shift right	
>>>	Bitwise shift right, sign extension	

Open with ▾

OPERATOR	FUNCTION	ASSOCIATION
<	Less than	Left to right
<=	Less than or equal to	
>	Greater than	
>=	Greater than or equal to	
instanceOf	Class membership	
==	Equal to	Left to right
!=	Not equal to	
&	Boolean AND (complete)	Left to right
&	Bitwise AND	
^	Boolean exclusive OR	Left to right
^	Bitwise exclusive OR	
	Boolean OR (complete)	Left to right
	Bitwise OR	
&&	Boolean AND (partial)	Left to right
	Boolean OR (partial)	Left to right
?:	Ternary conditional	Right to left
=	Assign	Right to left
+=	Add and assign	
-=	Subtract and assign	
*=	Multiply and assign	
/=	Divide and assign	
%=	Modulo and assign	
<<=	Shift left and assign	
>>=	Shift right, sign extension, and assign	
>>>=	Shift right, no sign extension, and assign	
&=	Boolean or bitwise AND and assign	
=	Boolean or bitwise OR and assign	

Examples

- Suppose $a = 3$, $b = 7$, $c = 10$, $d = -20$

<code>a == b</code>	<code>//evaluates to false</code>
<code>a != b</code>	<code>//evaluates to true</code>
<code>a + b < c + d</code>	<code>//what has precedence</code>
<code>a < b < c</code>	<code>// ??</code>
<code>a == b == c</code>	<code>// ??</code>

Do you understand the error message?

The operator `<` is undefined for the argument type(s) boolean, int

The operator `==` is undefined for the argument type(s) boolean, int

Logical Operators

- **!** **Not**
- **&&** **And**
- **||** **Or**

Let's look at the Operator Precedence Chart.
One More Time!

Operator Precedence

Table C-1 shows the operator precedence. The operators shown in bold are not discussed in this book. To learn more about them, see the references on Sun's Web site (<http://www.sun.com>).

TABLE C-1

Operator precedence

OPERATOR	FUNCTION	ASSOCIATION
()	Parentheses	Left to right
[]	Array subscript	
.	Object member selection	
++	Increment	Right to left
--	Decrement	
+	Unary plus	
-	Unary minus	
!	Boolean negation	
~	Bitwise negation	
(type)	Type cast	
*	Multiplication	Left to right
/	Division	
%	Modulus	
+	Addition or concatenation	Left to right
-	Subtraction	
<<	Bitwise shift left	Left to right
>>	Bitwise shift right	
>>>	Bitwise shift right, sign extension	



e.pdf

OPERATOR	FUNCTION	ASSOCIATION
<	Less than	Left to right
<=	Less than or equal to	
>	Greater than	
>=	Greater than or equal to	
instanceOf	Class membership	
==	Equal to	Left to right
!=	Not equal to	
&	Boolean AND (complete)	Left to right
&	Bitwise AND	
^	Boolean exclusive OR	Left to right
^	Bitwise exclusive OR	
	Boolean OR (complete)	Left to right
	Bitwise OR	
&&	Boolean AND (partial)	Left to right
	Boolean OR (partial)	Left to right
?:	Ternary conditional	Right to left
=	Assign	Right to left
+=	Add and assign	
-=	Subtract and assign	
*=	Multiply and assign	
/=	Divide and assign	
%=	Modulo and assign	
<<=	Shift left and assign	
>>=	Shift right, sign extension, and assign	
>>>=	Shift right, no sign extension, and assign	
&=	Boolean or bitwise AND and assign	
=	Boolean or bitwise OR and assign	



4.5 The While Loop

Continue executing the loop while condition is true

A boolean expression



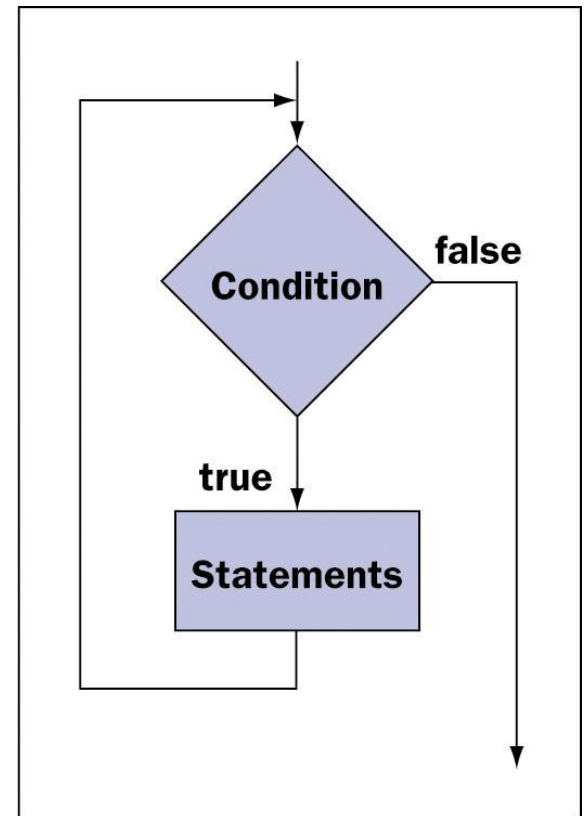
```
while (condition )  
    statement;
```

```
while (condition ) {  
    statement;  
    statement;  
}
```

There must be a statement in the body of the loop that changes the result of the boolean expression or it will be an infinite loop

The While Loop – Flow chart

- Provides a looping mechanism that executes statements repeatedly as long as some condition remains true
- Every “pass” through the body of the loop is an *iteration*



Example – Count-controlled

```
//Find sum between start and end using  
//a designated increment value  
int ctr, start=10, end=100, sum=0, increment=7;  
ctr = start;  
while (ctr <= end) {  
    sum += ctr;  
    ctr += increment;  
}  
System.out.print (sum);
```

Example –Event-controlled

```
//Display the first value for which  $1 + 2 + \dots + n$   
//is greater than 1 million  
int sum = 0;  
int number = 0;  
while (sum <= 1000000) {  
    number ++;  
    sum += number;  
}  
System.out.print (number);
```

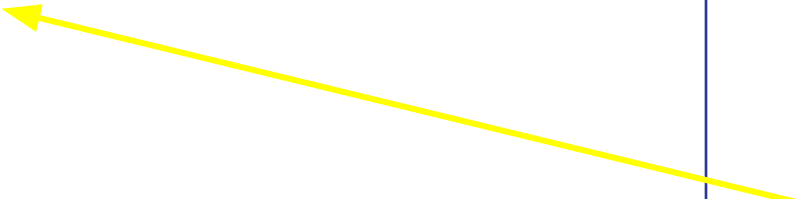
Your Turn!

1. List three components of a while loop.
2. What happens if the condition of the while loop is false from the outset?
3. Describe the purpose of the following.

```
Scanner read = new Scanner (System.in);  
S.O.P ("Enter a positive integer");  
int x=read.nextInt();  
while (x < 0) {  
    S.O.P ("Enter a Positive Integer");  
    x = read.nextInt();  
}
```



The for Loop (a fixed-iteration loop)

```
for (initialize; test; update) {  
    statement1;  
    statement2;  
    statement3;  
}
```



If the loop control variable is declared and initialized within the for statement, its scope and lifetime is only within the for loop

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



When body of the loop contains only one statement, curly brackets are optional

Declaring the loop control variable

//Example 1 - declaring outside of loop

```
int j;
```

```
for (j = 0 ; j < 10 ; j++)
```

```
    System.out.print (j);
```

//Example 2 - declaring inside of loop

//Preferred method because k is within the scope

//of the loop

```
for (int k = 0; k < 10; k++)
```

```
    System.out.print (k);
```

4.7 Nested loops

- The inside loop must complete all iterations before the next iteration of the outside loop
- The `break` statement will terminate a loop early. It is usually a condition of an `if` statement. It will only terminate the loop that it resides in.

4.8 Input files

```
Scanner reader =  
    new Scanner (new File ("mystuff.txt"));  
  
while (reader.hasNext() ) {  
    int inputValues = reader.nextInt();  
}
```

4.8 Output Files

```
PrintWriter writer = new PrintWriter  
                        (new File ("output.txt"));  
  
writer.print("Writing to my file");  
  
writer.close();    //required after outputs  
                  // have been completed
```

4.9 Find the error

```
int sum;  
  
int i = 3;  
while (i < 100) {  
    sum = sum + i;  
    i = i + 1;  
}  
System.out.println (sum);
```

Find the error

```
//find product of all positive odd
//integers < 100
int product = 1, j = 1;
while (j < 99) {
    product *= j;
    j += 2;
}
System.out.println (product);
```

Find the error

```
//find the same product  
int product = 1;  
int k = 3;  
while ( k != 100 ) {  
    product = product * k;  
    k += 2;  
}  
System.out.println (product);
```

Sample AP Questions

If addition had higher precedence than multiplication, the value of the expression

$$2 * 3 + 4 * 5$$

would be which of the following?

- A. 14
- B. 26
- C. 50
- D. 70
- E. 120

The expression

! (a && b)

is equivalent to which of the following expressions?

A. **(!a) && (!b)**

B. **(!a) || (!b)**

C. **! (a || b)**

D. **(a || b)**

E. **(a || b) && (a && b)**

deMorgan's Law

$\neg (x \ \&\& \ y)$ is the same as $\neg x \ || \ \neg y$

$\neg (x \ || \ y)$ is the same as $\neg x \ \&\& \ \neg y$

$x \ || \ (y \ \&\& \ z)$ same as $(x \ || \ y) \ \&\& \ (x \ || \ z)$

$x \ \&\& \ (y \ || \ z)$ same as $(x \ \&\& \ y) \ || \ (x \ \&\& \ z)$

Consider the following code segment:

```
int x = 0;
boolean y = true;
if ( y && (x != 0) && (2/x == 0) )
    System.out.println ("success");
else
    System.out.println ("failure");
```

Which of the following statements about this code segment is true?

- a. There will be an error when the code is compiled because the first && operator is applied to a non-boolean expression
- b. There will be an error when the code is compile because a boolean variable (y) and an int variable (x) appear in the same if-statement condition.
- c. There will be an error when the code is executed because of an attempt to divide by zero
- d. The code will compile and execute without error; the output will be “success”
- e. The code will compile and execute without error; the output will be “failure”

Short-Circuit Evaluation

- Expressions involving logical AND and OR are evaluated from left to right and evaluation stops as soon as the final value is known

- `(5 > 0) || x < 7` `x < 7` is never evaluated

- `(5 < 0) && x < 7` `x < 7` is never evaluated

Assume that the following definitions have been made and that the variable x has been initialized.

```
int x;  
boolean result;
```

Consider the following code segments:

Segment I	<pre>result = (x%2 == 0);</pre>	Segment III
Segment II	<pre>if (x%2 == 0) result = true; else result = false;</pre>	<pre>if ((x*2)/2 == x) result = true; else result = false;</pre>

Which of these code segments sets `result` to `true` if `x` is even, and to `false` if `x` is odd?

- a. I only b. II only c. III only d. I and II e. I and III

Homework, but not this weekend



Do now: Draw out your initials on graph paper using an asterisks or blank spaces in each box of the graph paper. It must be at least 10 squares high and as wide as you need. Leave a blank column between your first and last initial. Count how many asterisks and spaces are in each row.

By Oct. 18: Read Chapter 4 pages 105 – 145

Skip Case Study pages 130 – 133

Skip Graphics – Pages 137 - 143