

Controlling Program Flow

The objectives of this chapter are:

- To explain boolean expressions
- To describe controlling program flow using *if* and *switch* statements
- To explain the three Java looping mechanisms
 - for
 - while
 - do while

Boolean Expressions

- Boolean expressions are those expressions which return either true or false.
- The return type of a boolean expression is boolean.
 - some other programming languages represent the return value of boolean expressions as integral values.
- Boolean expressions use the relational and equality operators as well as logical AND and OR.
 - >, >=, <, <=
 - !=, ==
 - &&, ||
- Boolean expressions can be used to control program flow.

Truth Tables

- Boolean algebra is based on Truth Tables
 - For A and B, both A and B must be true for the expression to evaluate to true.
 - For A or B, either A or B can be true and the expression will evaluate to true.

AND

B \ A	false	true
false	false	false
true	false	true

OR

B \ A	false	true
false	false	true
true	true	true

Truth Tables Cont'd

- Java also has an operator for logical NOT (!).
 - Use parentheses to aid readability

```
if ((x == 5) && !(y > 50))  
    statement1;
```

NOT	
A	
false	true
true	false

Expression "Short Circuiting"

- Boolean expressions are "Short Circuited"
- Because of the nature of the truth table for AND, if any of the operands are false, the whole expression evaluates to false.
 - Sub expressions need not be evaluated
- A similar case exists with OR. If any of the operands are true, the whole expression evaluates to true.
- In this example, if x is not equal to 5 the second expression (y>50) will not be evaluated because the first operand to && is false.

```
if ((x == 5) && (y > 50))  
    statement1;
```

Tips for Boolean Expressions

- One of the most common mistakes is using the assignment operator instead of testing equality.

mistake:

```
if (x = 5)
    statement1;
```

instead of

```
if (x == 5)
    statement1;
```

- Although logical AND (&&) has a higher precedence than logical OR (||), it is a good idea to use parentheses to aid readability.

```
if ( ((x == 5) && (y > 50)) || (x > 50) )
    statement1;
```

If statements

- The general form is:

```
if (boolean-expression)
    statement1;
else
    statement2;
```

- If the boolean expression evaluates to true, statement1 is executed. If false, statement2 is executed.
- The else clause is optional.
- If more than one statement is to be executed, the statements must be grouped in a block.

If statements using blocks

- When executing multiple statements:

```
if (boolean-expression)
{
    statement1;
    statement2;
    statement3;
}
else
{
    statement4;
    statement5;
    statement6;
}
```

- Statement indentation and placement of curly braces should enhance readability.

Nested if statements

- If statements can be nested. When doing so, it is recommended that curly braces and indentation be used to clearly show the structure of the code.

```
if (boolean-expression)
{
    if (boolean-expression)
    {
        statement1;
    }
    else
    {
        statement2;
    }
}
else
{
    statement3;
}
```

Ternary Operator -- ?:

- The ternary operator is a shortcut for the following code:

```
if (boolean-expression)
    result = value1;
else
    result = value2;
```

general syntax:

```
(boolean-expression)? value1 : value2
```

- If the boolean expression evaluates true, the whole expression evaluates to value1. Otherwise, it is value2.

```
z = (x > y)? x : y;
```

is equivalent to:

```
if (x > y)
    z = x;
else
    z = y;
```

Switch Statement

- Sometimes, using if-else statements can become unruly:

```
if (x == 1)
    statement1;
else if (x == 2)
    statement2;
else if (x == 3)
    statement3;
else if (x == 4)
    statement4;

[ ... ]
```

- Code structures like this are difficult to read and maintain.
- The switch statement is a better choice.

Switch Statement Cont'd

- General Syntax:

```
switch( integer_expression )
{
    case constant1:
        statement1;
                                break;
    case constant2:
        statement2;
                                break;
    case constant3:
        statement3;
                                break;
    default:
        statement4;
}
```

- The integer expression is evaluated
- Control passes to the case whose integer constant matches the value of the expression.
- If there is no match, control passes to the default case. If no default case is present, control leaves the switch statement.

Notes on the Switch statement

- The condition is an integer value.
- The case value must be known at compile time. It is good coding practice to use final variables as case constants.
- The default case is optional.
- **The break statement causes execution to leave the switch statement. *If the break statement is not present*, execution will continue to the next statement, even if it is within another case.**

Loops

- Loops are used to execute statements or blocks multiple times based on a looping condition.
- Java has three types of loops:
 - while loops
 - do while loops
 - for loops
- Care should be taken whenever a loop is used to avoid an endless loop.

while loops

- The while loop is the most basic loop in Java.

```
while (boolean-expression)
{
    statement1;
    [...]
}
```

- The loop body will continue to execute as long as the looping condition is true. The looping condition is tested upon entry and when the loop body is completed.
- If the loop body consists of a single statement, the curly braces are not necessary.
- If the looping condition is false upon entry, the loop body will not be executed.

do while loops

- The do-while loop is identical to the while loop except that the test is evaluated at the end of the loop.

```
do
{
    statement1;
    [...]
} while (boolean-expression);
```

- Because the looping condition is evaluated at the end of the loop body, the loop body is guaranteed to execute at least once.

Loops - Examples

- What will these loops output?

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

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

```
int x = 0;
do
{
    System.out.println(x);
    x = x+1;
} while (x<10) ;
```

Loops - Common Errors

- What are the errors in these loops?

endless loop

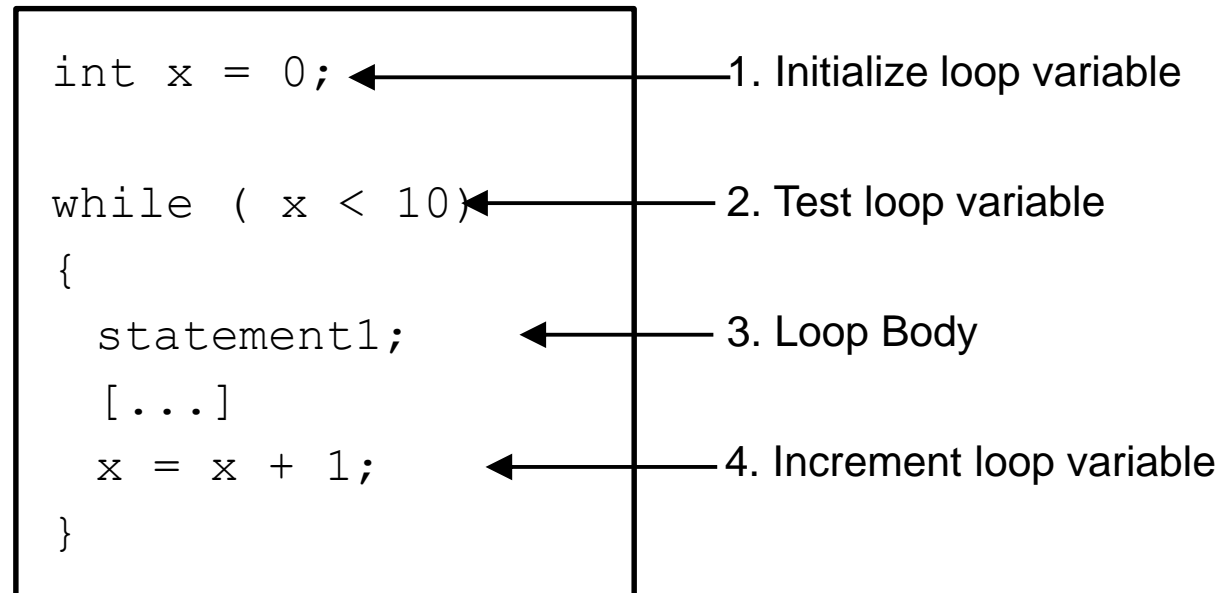
```
int x = 0;
while (x < 10);
{
    System.out.println(x++);
}
```

endless loop

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

Loop Components

- Each loop has 4 main components



for loop

- The syntax of the for loop makes the 4 parts of the loop explicit.

```
int x = 0; ← 1. Initialize loop variable  
  
while ( x < 10) ← 2. Test loop variable  
{  
    statement1; ← 3. Loop Body  
    [...]   
    x = x + 1; ← 4. Increment loop variable  
}
```

syntax:

```
for (i = 0; i < 10;  
    i++)  
{  
    statement1;  
    [...]   
}
```

Notes about loops

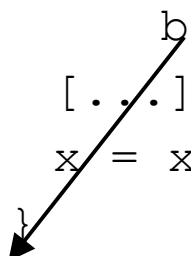
- The test condition on the for loop is the same as the while loop. The loop body is executed while the condition is true.
- **The initialization and increment portions of the for loop are optional. However, the semicolons must be present.**
- **If the test condition is omitted, the test is always true.**
- for loops are generally used when the number of times the loop is to be executed is known.
 - Do not adjust the looping variable within the loop
- while and do-while loops are used when the number of times the loop is to be executed is not known.
 - The focus is "while" this condition is true.

break revisited

- We previously see the break statement used in switch statements.
 - break can also be used with loops.
- The break statement will cause the flow of execution to break out of the current loop.
- If loops are nested, break will cause control to leave the inner-most loop.

```
int x = 0;

while ( x < 10)
{
    if (y > 100)
        break;
    [...]
    x = x + 1;
}
```

A black arrow originates from the 'break;' statement and points diagonally down and to the left, ending at the closing curly brace of the while loop, illustrating that the loop is exited immediately.


continue

- continue is similar to break.
- continue causes execution to go back to the loop test condition. If the test condition is true, the loop will be executed again. If not, the loop body is exited.

```
int x = 0;

while (x < 10)
{
    if (y > 100)
        continue;

    [...]
    x = x + 1;
}
```



Notes about break and continue

- Do not overuse break and continue.
- break and continue are structured goto statements.
 - The overuse of break and continue usually indicates a poor design.
 - Re-design and re-write is usually the best solution.
- Overuse of if and switch statements generally indicates a procedural solution.

Review

- What are boolean expressions?
- How are boolean expressions used to change program flow?
- What is short circuiting? Why is it important?
- What is the general form of an if statement?
- What is the purpose of the switch statement?
- Name the three types of loops in Java
- How are while and do while loops different?
- What do the break and continue statements do?