Statements and Control Flow

• The programs we have seen so far do exactly the same list of instructions every time
• What if we want to do different things for different inputs?
  – Do some action only if a specified condition is met
  – We need *conditional* statements

```java
if(value < 0)
  System.out.println(“Bad input”);
```
Statements and Control Flow

• What if we want to repeat some set of instructions some number of times?
  – Repeat an action some number of times
  – We need *iterative* statements

```java
while(i < 100)
    System.out.println(i++);
```
Statements

• *Declaration Statement*: type, followed by a comma-separated list of identifiers, followed by a semi-colon
  
  ```
  int foo, bar;
  String name = “Scott”;
  ```

• *Expression Statement*: expression followed by a semi-colon
  
  – *Assignment Expression*: `size = size + 5`
  – *Method Call Expression*: `System.out.println(…)`
  – Not all expressions can be part of an expression statement (more on this later)
Block Statement

- **Block Statement**: one or more statements inside braces, e.g.,

```
{  
    int a = 4;  // Statement  
    System.out.println(a);  // Statement
}  // Statement
```

- A Block Statement is a Statement
- Block Statements can contain Block Statements
- Variables declared within a block disappear when the block has finished executing
Empty Statement

• *Empty statement*: do-nothing statement ;
  – Is a statement, but does nothing
• Example:
  – Wait for a condition to become true
    while(notTimeYet)
    ;
    <other stuff>
  – Question: Which is clearer, that, or this:
    while(notTimeYet);
    <other stuff>
Boolean Expression

• Any expression that evaluates to true or false
  – true
  – false
  – Comparisons
  – Logical operations
# Relational Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal</td>
<td><code>==</code></td>
<td><code>a == b</code></td>
</tr>
<tr>
<td>Not equal</td>
<td><code>!=</code></td>
<td><code>a != b</code></td>
</tr>
<tr>
<td>Less than</td>
<td><code>&lt;</code></td>
<td><code>a &lt; b</code></td>
</tr>
<tr>
<td>Greater than</td>
<td><code>&gt;</code></td>
<td><code>a &gt; b</code></td>
</tr>
<tr>
<td>Less than or equal to</td>
<td><code>&lt;=</code></td>
<td><code>a &lt;= b</code></td>
</tr>
<tr>
<td>Greater Than or Equal to</td>
<td><code>&gt;=</code></td>
<td><code>a &gt;= b</code></td>
</tr>
</tbody>
</table>
Comparisons

• Comparisons (using relational operators) evaluate to true or false
• Example:
  ```java
  int a = 5, b = 7;
  boolean flag;
  flag = (a < b); // boolean expression
  System.out.println(flag);
  ```
Logical Operators

- Operations on logical values

<table>
<thead>
<tr>
<th>Name</th>
<th>Operator</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT</td>
<td>!</td>
<td>a = !(b == c)</td>
</tr>
<tr>
<td>AND</td>
<td>&amp;&amp;</td>
<td>a = (b &amp;&amp; c)</td>
</tr>
<tr>
<td>OR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Logical Operations

• Example 1
  int x, y;
  boolean b;
  x = Console.in.getInt();
  y = Console.in.getInt();
  b = (x == y);
  System.out.println(b);

• Example 2
  boolean b = (age >= 18 && age < 65);
  System.out.println("full fare adult is " + b);
  b = (age < 18 || age >= 65);
  System.out.println("reduced fare is" + b);
Operator Precedence and Associativity

• **Operator Precedence**
  – The order in which different operators are evaluated, i.e. who goes first
  – * has higher precedence than +, both higher than =
    ```
    int x = 3 + 4 * 5;  // x = 23, not 35!
    ```

• **Operator Associativity**
  – The order in which operators of the same precedence are applied
  – * and % have equal precedence, left to right associativity
    ```
    int y = 4 * 3 % 2;  // y = 0, not 4!
    ```
## Operator Precedence and Associativity

<table>
<thead>
<tr>
<th>Operators</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>( ) ++ (postfix) -- (postfix)</td>
<td>Left to right</td>
</tr>
<tr>
<td>+ (unary) - (unary) ++ (prefix) -- (prefix) !</td>
<td>Right to left</td>
</tr>
<tr>
<td>* / %</td>
<td>Left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>Left to right</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>Left to right</td>
</tr>
<tr>
<td>== !=</td>
<td>Left to right</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Left to right</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>= += -= *= /= etc.</td>
<td>Right to left</td>
</tr>
</tbody>
</table>
What if we want a different evaluation order?

• Parentheses ( ) have a higher precedence than just about everything else
  – They can be used to impose a different evaluation order

  int x = 3 + 4 * 5;    // x = 23
  int x = (3 + 4) * 5;  // x = 35
  int y = 4 * 3 % 2;    // y = 0
  int y = 4 * (3 % 2);  // y = 4
Conditional Statements

• Conditionally execute a statement based on the value of a boolean expression
  – **if statement** - decide whether or not to take a particular action
    • Execute a particular statement only if a given boolean expression is true
  – **if-else statement** - choose between two alternative actions
    • Execute one of two statements based on the value of a given boolean expression
Conditional Statements (cont.)

- **switch statement**: choose among several alternative actions
  - Execute one of a set of statements based on a specified value (not a boolean expression)

- **while statement**: repeat an action as long as a specified condition is true
  - Repeatedly execute a statement as long as the given boolean expression is true

- **for statement**: execute an action a specified number of times
  - Repeatedly execute a statement as long as the given boolean expression is true
if Statement

• Used to decide whether or not to take a particular action

   if(<boolean expression>)
       <statement>

• If the boolean expression is true, the then statement is executed, otherwise it is not
Flowchart For An if Statement

- True
- Boolean Expression
- Statement

False

The rest of the program
if Statements in Action

if(value > 50)
    System.out.println(“Warning, value too big!”);

if(y != 0)
    z = x / y;

if(item.price < 100 && cashOnHand >= item.price) {
    item.purchase( );
    cashOnHand -= item.price;
}
Example: Bubblesort

• Given three numbers, place them in increasing order

• Algorithm:

1. Put the three numbers in $a$, $b$, and $c$
2. If $b$ is less than $a$, swap $a$ and $b$
3. If $c$ is less than $b$
   1. Swap $b$ and $c$
   2. If $b$ is less than $a$, swap $a$ and $b$
Bubblesort (1)

1. Put the three numbers in \(a\), \(b\), and \(c\)

2. If \(b\) is less than \(a\), swap \(a\) and \(b\)

3. If \(c\) is less than \(b\)
   1. swap \(b\) and \(c\)
   2. if \(b\) is less than \(a\), swap \(a\) and \(b\)

\[
\begin{array}{|c|}
\hline
  a & 24 \\
\hline
  b & 0 \\
\hline
  c & 37 \\
\hline
\end{array}
\]
Bubblesort (1)

1. Put the three numbers in \( a, b, \) and \( c \)

2. If \( b \) is less than \( a \), swap \( a \) and \( b \)

3. If \( c \) is less than \( b \)
   1. swap \( b \) and \( c \)
   2. if \( b \) is less than \( a \), swap \( a \) and \( b \)
Bubblesort (1)

1. Put the three numbers in $a$, $b$, and $c$
2. If $b$ is less than $a$, swap $a$ and $b$
3. If $c$ is less than $b$
   1. swap $b$ and $c$
   2. if $b$ is less than $a$, swap $a$ and $b$
Bubblesort (1)

1. Put the three numbers in a, b, and c

2. If b is less than a, swap a and b

3. If c is less than b
   1. swap b and c
   2. if b is less than a, swap a and b
Bubblesort (1)

- a: 0
- b: 24
- c: 37
Bubblesort (2)

1. Put the three numbers in a, b, and c

2. If b is less than a, swap a and b

3. If c is less than b
   1. swap b and c
   2. if b is less than a, swap a and b
Bubblesort (2)

<table>
<thead>
<tr>
<th>a</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>17</td>
</tr>
<tr>
<td>c</td>
<td>6</td>
</tr>
</tbody>
</table>

1. Put the three numbers in \(a, b,\) and \(c\)

2. **If \(b\) is less than \(a\), swap \(a\) and \(b\)**

3. If \(c\) is less than \(b\)
   1. swap \(b\) and \(c\)
   2. if \(b\) is less than \(a\), swap \(a\) and \(b\)**
Bubblesort (2)

1. Put the three numbers in $a$, $b$, and $c$

2. If $b$ is less than $a$, swap $a$ and $b$

3. If $c$ is less than $b$
   1. swap $b$ and $c$
   2. if $b$ is less than $a$, swap $a$ and $b$
Bubblesort (2)

1. Put the three numbers in \( a, b, \) and \( c \)
2. If \( b \) is less than \( a \), swap \( a \) and \( b \)
3. If \( c \) is less than \( b \)
   1. swap \( b \) and \( c \)
   2. if \( b \) is less than \( a \), swap \( a \) and \( b \)

\[
\begin{array}{c}
\text{a} & 17 \\
\text{b} & 35 \\
\text{c} & 6 \\
\end{array}
\]
Bubblesort (2)

1. Put the three numbers in \( a, b, \) and \( c \)
2. If \( b \) is less than \( a \), swap \( a \) and \( b \)
3. If \( c \) is less than \( b \)
   1. swap \( b \) and \( c \)
   2. if \( b \) is less than \( a \), swap \( a \) and \( b \)
Bubblesort (2)

1. Put the three numbers in $a$, $b$, and $c$
2. If $b$ is less than $a$, swap $a$ and $b$
3. If $c$ is less than $b$
   1. swap $b$ and $c$
   2. if $b$ is less than $a$, swap $a$ and $b$
Bubblesort (2)

1. Put the three numbers in $a$, $b$, and $c$

2. If $b$ is less than $a$, swap $a$ and $b$

3. If $c$ is less than $b$
   1. swap $b$ and $c$
   2. if $b$ is less than $a$, **swap $a$ and $b$**
Bubblesort (2)

a  6
b  17
c  35
// SortInput.java - sort three numbers
import tio.*; // use the package tio

class SortInput {
    public static void main (String[] args) {
        int a, b, c, temp;

        // Get three numbers from the user
        System.out.println("type three integers:");
        a = Console.in.readInt();
        b = Console.in.readInt();
        c = Console.in.readInt();

        // If b is less than a, swap a and b
        if (b < a) {
            temp = a;
            a = b;
            b = temp;
        }
    }
}
// If c is less than b, swap b and c
if (c < b) {
    temp = b;
    b = c;
    c = temp;
}

// if (the new) b is less than a, swap a and b
if (a > b) {
    temp = a;
    a = b;
    b = temp;
}

System.out.print("The sorted order is : ");
System.out.println(a + ", " + b + ", " + c);
If-else Statement

• Used to choose between two alternative actions

```plaintext
if(<boolean expression>)
  <statement>
else
  <statement>
```

• If the boolean expression is true, the *then* statement is executed, otherwise the *else* statement is executed
Flowchart For An if-else Statement

- Boolean Expression
- True
  - Then Statement
- False
  - Else Statement

The rest of the program
If-else Statements in Action

```java
if (x < y)
    min = x;
else
    min = y;
System.out.println("min = " + min);
```

```java
if (y == 0)
    System.out.println("Divide by zero error!");
else
    z = x / y;
```
Details

• Any statement can be a then or an else statement
  – Expression Statement, Block Statement, Conditional Statement (including if or if-else Statements), etc.

• Common errors
  – Look at the ones listed in the book
    • They are exactly right
if-else-if-else

- If you string if-elses together, each if-else is the statement for the previous else

  if(<boolean expression>)
  <statement>
  else if(<boolean expression>)
  <statement>
  else <statement>
  <etc.>
Dangling else

• An *else* always binds to the nearest previous unmatched *if* in its block

```java
if(<boolean expression1>) {  // if 1
    if(<boolean expression2>)    // if 2
        <statement>
    else                                       // binds to “if 2”
        <statement>
}  
```
Dangling else

• An *else* always binds to the nearest unmatched *if* in its block

```java
if(<boolean expression1>) {  // if 1
  if(<boolean expression2>)     // if 2
    <statement>
}  // binds to “if 1”
else
  <statement>
```
while Statement

• Repeat some action as long as a specified condition is true

  while(<boolean expression>)
  <statement>

• Repeatedly execute <statement> until <boolean expression> is false
  – May not execute <statement> at all
Flowchart For A while Statement

Boolean Expression

Statement

True

False

The rest of the program
while Statements in Action

```java
int value = 0;
while(value < 5)
    System.out.println(value++);
```

```java
char c = 'a';
while(c != 'x') {
    c = Console.in.getChar();
    System.out.println(c);
}
```
for Statement

- Repeat some action as long as a specified condition is true
  
  for(<init>; <boolean>; <update>)
  
  <statement>
- <init> - executed once, at the beginning
- <boolean> - checked each time through the loop, before <statement>
- <update> - executed each time, after <statement>
- for statements are the same as while statements, except that init and update are explicitly included
Flowchart For A for Statement

1. Initialization
2. Boolean Expression
   - True: Statement
   - False: Update
3. The rest of the program
for Statements in Action

for(int value = 0; value < 5; value++)
    System.out.println(value);

for(char c = 'a';  c != 'x'; System.out.print(c))
    c = Console.in.getChar();

for(int i = 1, j = 1, t; i < 100; j = t + j) {
    System.out.println(i);
    t = i;
    i = j;
}
break and continue

• break and continue interrupt the flow of control in a while loop, for loop, or switch statement

• break
  – Jumps out of a while or for loop
    • With nested loops, jumps out of innermost one only
  – Causes a switch statement to terminate
    • If you omit it after a case, control drops into the next case

• continue
  – Terminates the current iteration of a loop
break Statement in Action

char c = 'a';
while(c != 'x') {
    c = Console.in.getchar();
    System.out.print(c);
}

while(true) {
    c = Console.in.readChar();
    System.out.println(c);
    if(c == 'x')
        break;
}
continue Statement in Action

for(int i = 0; i < 100; i++) {
    if(i % 2 == 1)
        continue;
    System.out.println(i);
}
switch Statement

- Choose among several alternative actions

```java
switch(<controlling expression>) {   // integer variable or value
    case value1:                 // if <ce> == value1, do <statement1>
        <statement1>
        break;
    case value2 :                 // if <ce> == value2, do <statement2>
        <statement2>
        break;
    <… more cases here …>
    case valuenn :                // if <ce> == valuenn, do <statementn>
        <statementn>
        break;
}
```
switch Statements in Action

```java
switch(dayOfWeek) {
    case 1:
        System.out.println("Sunday");
        break;
    case 2:
        System.out.println("Monday");
        break;
    <etc.>
    default:
        System.out.println("Huh?");
        break;
}
```