Selections and Loops

Introduction to Programming and Computational Problem Solving - 2
CSE 8B
Lecture 4
Announcements

• Assignment 1 is due today, 11:59 PM
• Quiz 1 is Oct 7
• Assignment 2 will be released today
  – Due Oct 12, 11:59 PM
• Educational research study
  – Oct 7, weekly survey
• Reading
  – Liang
    • Chapters 3 and 5
Selections and loops

• Selections
  – Relational operators (e.g., less than, equal to)
  – Logical operators (e.g., not, and, or)
  – if statements
  – if-else statements
  – switch statements

• Loops
  – while loops
  – do-while loops
  – for loops
The boolean type and operators

• Often in a program you need to compare two values, such as whether \( i \) is greater than \( j \)
• Java provides six comparison operators (also known as relational operators) that can be used to compare two values
• The result of the comparison is a Boolean value: true or false
• For example
  ```java
  boolean b = (1 > 2);
  ```
# Relational operators

<table>
<thead>
<tr>
<th>Java Operator</th>
<th>Mathematics Symbol</th>
<th>Name</th>
<th>Example (radius is 5)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code></td>
<td><code>&lt;</code></td>
<td>less than</td>
<td><code>radius &lt; 0</code></td>
<td>false</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td><code>≤</code></td>
<td>less than or equal to</td>
<td><code>radius &lt;= 0</code></td>
<td>false</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td><code>&gt;</code></td>
<td>greater than</td>
<td><code>radius &gt; 0</code></td>
<td>true</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td><code>≥</code></td>
<td>greater than or equal to</td>
<td><code>radius &gt;= 0</code></td>
<td>true</td>
</tr>
<tr>
<td><code>==</code></td>
<td><code>=</code></td>
<td>equal to</td>
<td><code>radius == 0</code></td>
<td>false</td>
</tr>
<tr>
<td><code>!=</code></td>
<td><code>≠</code></td>
<td>not equal to</td>
<td><code>radius != 0</code></td>
<td>true</td>
</tr>
</tbody>
</table>
if statements

if (boolean-expression) {
    statement(s);
}

Braces are optional for a single statement; however, it is best practice (less error prone) to **always use braces**
if statements

```java
if (radius >= 0) {
    area = radius * radius * PI;
    System.out.println("The area for the circle of radius "+ radius + " is "+ area);
}
```

![Flowchart diagram for if statement]
if-else statements

if (boolean-expression) {
    statement(s)-for-the-true-case;
}
else {
    statement(s)-for-the-false-case;
}

Braces are optional for a single statement; however, it is best practice (less error prone) to **always use braces**.
### if-else statements

```java
if (radius >= 0) {
    area = radius * radius * 3.14159;
    System.out.println("The area for the "
    + "circle of radius " + radius +
    " is " + area);
}
else {
    System.out.println("Negative input");
}
```
Conditional operator

(boolean-expression) ? expression1 : expression2

if (x > 0) {
    y = 1;
} else {
    y = -1;
}

is equivalent to

y = (x > 0) ? 1 : -1;
Multiple if-else statements

```
score >= 90
  true
  grade is A

score >= 80
  false
  score >= 70
    false
    grade is D
    true
    grade is C

score >= 60
  false
  grade is F

```

CSE 8B, Fall 2022
Multiple if-else statements

```
if (score >= 90.0)
  System.out.print("A");
else
  if (score >= 80.0)
    System.out.print("B");
  else
    if (score >= 70.0)
      System.out.print("C");
    else
      if (score >= 60.0)
        System.out.print("D");
      else
        System.out.print("F");
```

Equivalent

```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

This is better
Nested statements

- The `else` clause matches the **most recent** `if` clause in the same block

```java
int i = 1, j = 2, k = 3;
if (i > j)
  if (i > k)
    System.out.println("A");
else
  System.out.println("B");
```

This is better with correct indentation.

Braces are optional for a single statement; however, it is best practice (less error prone) to **always use braces**

Nothing is printed
Nested statements

• To force the else clause to match the first if clause, you must add a pair of braces

```java
int i = 1;
int j = 2;
int k = 3;
if (i > j) {
    if (i > k)
        System.out.println("A");
} else
    System.out.println("B");
```

B is printed

Braces are optional for a single statement; however, it is best practice (less error prone) to **always use braces**
Less error prone

if (number % 2 == 0)
    even = true;
else
    even = false;

Equivalent

boolean even = number % 2 == 0;

if (even == true)
    System.out.println("It is even.");

Equivalent

if (even)
    System.out.println("It is even.");
# Logical operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
<td>logical negation</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
<td>logical conjunction</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>exclusive or (xor)</td>
<td>logical exclusion</td>
</tr>
</tbody>
</table>
Truth table for operator `!`:

<table>
<thead>
<tr>
<th><code>p</code></th>
<th><code>!p</code></th>
<th>Example: <code>age = 24</code> and <code>weight = 140</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
<td><code>!(age &gt; 18)</code> is false, because <code>(age &gt; 18)</code> is true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td><code>!(weight == 150)</code> is true, because <code>(weight == 150)</code> is false</td>
</tr>
</tbody>
</table>
Truth table for operator `&&`

<table>
<thead>
<tr>
<th>( p_1 )</th>
<th>( p_2 )</th>
<th>( p_1 \land p_2 )</th>
<th>Example: ( \text{age} = 24 \text{ and weight} = 140 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>(( \text{age} \leq 18 ) \land (( \text{weight} &lt; 140 )) is false, because both conditions are false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>(( \text{age} \leq 18 ) \land (( \text{weight} \geq 140 )) is false, because (( \text{age} \leq 18 )) is false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>(( \text{age} &gt; 18 ) \land (( \text{weight} &gt; 140 )) is false, because (( \text{weight} &gt; 140 )) is false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>(( \text{age} &gt; 18 ) \land (( \text{weight} \geq 140 )) is true, because both conditions are true</td>
</tr>
</tbody>
</table>

Example:

- \( \text{age} = 24 \) and \( \text{weight} = 140 \)
- \( \text{age} = 24 \) and \( \text{weight} < 140 \)
- \( \text{age} = 24 \) and \( \text{weight} \geq 140 \)
- \( \text{age} = 24 \) and \( \text{weight} > 140 \)
## Truth table for operator $\lor$

<table>
<thead>
<tr>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$p_1 \lor p_2$</th>
<th>Example: age = 24 and weight = 140</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>(age &gt; 34) $\lor$ (weight $\geq$ 150) is false, because both conditions are false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>(age &gt; 34) $\lor$ (weight $\leq$ 140) is true, because (weight $\leq$ 140) is true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>(age &gt; 14) $\lor$ (weight $\geq$ 150) is false, because (age &gt; 14) is true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>(age &gt; 14) $\lor$ (weight $\leq$ 140) is true, because both conditions are true</td>
</tr>
</tbody>
</table>
### Truth table for operator ^

<table>
<thead>
<tr>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$p_1 \land p_2$</th>
<th>Example: $\text{age} = 24 \text{ and } \text{weight} = 140$</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>$(\text{age} &gt; 34) \land (\text{weight} &gt; 140)$ is false, because both conditions are false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>$(\text{age} &gt; 34) \land (\text{weight} \geq 140)$ is true, because $(\text{age} &gt; 34)$ is false and $(\text{weight} \geq 140)$ is true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>$(\text{age} &gt; 14) \land (\text{weight} &gt; 140)$ is true, because $(\text{age} &gt; 14)$ is true and $(\text{weight} &gt; 140)$ is false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>$(\text{age} &gt; 14) \land (\text{weight} \geq 140)$ is false, because both conditions are true</td>
</tr>
</tbody>
</table>
Short-circuit operators

• && and || are short-circuit operators
  • p1 && p2
    – If p1 or p2 is false, then p1 && p2 is false
    – p1 is evaluated first
      • If p1 is true, then p2 is evaluated
      • If p1 is false, then p2 is not evaluated
  • p1 || p2
    – If p1 or p2 is true, then p1 || p2 is true
    – p1 is evaluated first
      • If p1 is true, then p2 is not evaluated
      • If p1 is false, then p2 is evaluated
switch statements

• When the value in a case statement matches the value of the switch-expression, the statements starting from this case are executed until either a break statement or the end of the switch statement is reached

```java
switch (switch-expression) {
    case value1:  statement(s)1;
        break;
    case value2: statement(s)2;
        break;
    ...
    case valueN: statement(s)N;
        break;
    default: statement(s)-for-default;
}
```
switch statements

• The switch-expression must yield a value of char, byte, short, int or String type and must always be enclosed in parentheses
• The value1, ..., and valueN must have the same data type as the value of the switch-expression
• The resulting statements in the case statement are executed when the value in the case statement matches the value of the switch-expression
• Note that value1, ..., and valueN are constant expressions (i.e., they cannot contain variables in the expression, such as 1 + x)

```java
switch (switch-expression) {
    case value1: statement(s)1;
                  break;
    case value2: statement(s)2;
                  break;
    …
    case valueN: statement(s)N;
                  break;
    default: statement(s)-for-default;
}
```
switch statements

• The keyword break is optional, but it should be used at the end of each case in order to terminate the remainder of the switch statement
  – If the break statement is not present, the next case statement will be executed

• The default case, which is optional, can be used to perform actions when none of the specified cases matches the switch-expression

```java
switch (switch-expression) {
    case value1: statement(s)1;
                break;
    case value2: statement(s)2;
                break;
    ...
    case valueN: statement(s)N;
                break;
    default: statement(s)-for-default;
}
```

The default case is optional; however, it is best practice (less error prone) to **always have a default case**
switch statements

- status is 0: Compute tax for single filers
  - break
- status is 1: Compute tax for married jointly or qualifying widow(er)
  - break
- status is 2: Compute tax for married filing separately
  - break
- status is 3: Compute tax for head of household
  - break
- default: Default actions
switch statements

switch (status) {
    case 0:  compute taxes for single filers;
             break;
    case 1:  compute taxes for married file jointly;
             break;
    case 2:  compute taxes for married file separately;
             break;
    case 3:  compute taxes for head of household;
             break;
    default: System.out.println("Error: invalid status");
             System.exit(1);
}

The default case is optional; however, it is best practice (less error prone) to **always have a default case**.
switch statements

switch (day) {
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
        System.out.println("Weekday");
        break;
    case 0:
    case 6:
        System.out.println("Weekend");
}
Operator precedence

- ( ), var++, var--
- ++var, --var, +, - (unary plus and minus), ! (not)
- (type) casting
- *, /, % (multiplication, division, and remainder)
- +, - (binary addition and subtraction)
- <, <=, >, >= (relational operators)
- ==, != (equality)
- ^ (exclusive or)
- && (and)
- || (or)
- =, +=, -=, *=, /=, %= (assignment operators)
Operator associativity

• When two operators with the same precedence are evaluated, the associativity of the operators determines the order of evaluation

• All binary operators except assignment operators are left-associative

  \[ a - b + c - d \text{ is equivalent to } ((a - b) + c) - d \]

• Assignment operators are right-associative

  \[ a = b += c = 5 \text{ is equivalent to } a = (b += (c = 5)) \]
Operator precedence and associativity

• The expression in the parentheses is evaluated first
  – Parentheses can be nested, in which case the expression in the inner parentheses is executed first

• When evaluating an expression without parentheses, the operators are applied according to the precedence rule and the associativity rule

• If operators with the same precedence are next to each other, their associativity determines the order of evaluation
Loops

• while loops
• do-while loops
• for loops
while loops

• Executes statements repeatedly while the condition is true

```java
while (loop-continuation-condition) {
    // loop-body
    Statement(s);
}
```
while loops

```java
int count = 0;
while (count < 100) {
    System.out.println("Welcome to Java");
    count++;
}
```
Ending a loop with a sentinel value

- Often the number of times a loop is executed is not predetermined
- You may use an input value to signify the end of the loop
- Such a value is known as a sentinel value
- For example, a program reads and calculates the sum of an unspecified number of integers. The input 0 signifies the end of the input.
do-while loops

• Execute the loop body first, then checks the loop continuation condition

do {
   // Loop body
   Statement(s);
} while (loop-continuation-condition);
for loops

- A concise syntax for writing loops

```java
for (initial-action; loop-continuation-condition; action-after-each-iteration) {
// loop body
Statement(s);
}
```
for loops

int i;
for (i = 0; i < 100; i++) {
    System.out.println("Welcome to Java!");
}
for loops

• The initial-action in a for loop can be a list of zero or more comma-separated expressions
• The action-after-each-iteration in a for loop can be a list of zero or more comma-separated statements
• However, it is best practice (less error prone) not to use comma-separated expressions and statements

```java
for (int i = 0, j = 0; (i + j < 10); i++, j++) {
    // Do something
}
```
Loops and floating-point accuracy

• Remember, calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy.

• As such, **do not use floating-point values for equality checking in a loop control**

```java
double sum = 0;
double item = 1;
while (item != 0) {  // No guarantee item will be 0
    sum += item;
    item -= 0.1;
}
System.out.println(sum);
```
Infinite loops

• If the loop-continuation-condition in a for loop is omitted, it is implicitly true

for (; ; ) {
  // Do something
}

(a)  

Equivalent

while (true) {
  // Do something
}

(b)
Loops

• The three forms of loop statements, while, do-while, and for, are expressively equivalent
  – You can write a loop in any of these three forms

\[
\text{while (loop-continuation-condition) \{ \\
  // Loop body \\
\}} \\
\]

\[\begin{align*}
  (a) \quad & \text{Equivalent} \\
  (b) \quad & \text{for ( ; loop-continuation-condition; ) \{ \\
  \quad \// Loop body \\
  \}}
\end{align*}\]

\[
\text{for (initial-action; \\
  loop-continuation-condition; \\
  action-after-each-iteration) \{ \\
  // Loop body; \\
\}} \\
\]

\[\begin{align*}
  (a) \quad & \text{Equivalent} \\
  (b) \quad & \text{initial-action; \\
  while (loop-continuation-condition) \{ \\
  \quad // Loop body; \\
  \quad action-after-each-iteration; \\
  \}}
\end{align*}\]
Loops

• Use the loop form that is most intuitive and comfortable
  – A for loop may be used if the number of repetitions is known
  – A while loop may be used if the number of repetitions is not known
  – A do-while loop can be used to replace a while loop if the loop body must be executed before testing the continuation condition
break

• Immediately terminate the loop

```java
public class TestBreak {
    public static void main(String[] args) {
        int sum = 0;
        int number = 0;

        while (number < 20) {
            number++;
            sum += number;
            if (sum >= 100)
                break;
        }

        System.out.println("The number is "+number);
        System.out.println("The sum is "+sum);
    }
}
```
public class TestContinue {
  public static void main(String[] args) {
    int sum = 0;
    int number = 0;
    
    while (number < 20) {
      number++;
      if (number == 10 || number == 11)
        continue;
      sum += number;
    }
    
    System.out.println("The sum is " + sum);
  }
}
Nested loops

• Loops can be nested
• For example, nested for loops are often used to handle two-dimensional data

```java
for (int i = 0; i < numRows; i++) {
    // Handle i-th row
    for (int j = 0; j < numColumns; j++) {
        // Handle j-th column on i-th row
    }
}
```
Next Lecture

• Methods
• Reading
  – Liang
    • Chapter 6