• Computing the Area of a Circle:

```java
import java.util.Scanner;
public class ComputeArea {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter a radius: ");
        double radius = input.nextDouble();
        double area = radius * radius * 3.14159;
        // Display results
        System.out.println("The area for the circle of radius " +
                        + " of radius " + radius + " is " + area);
    }
}
```
Motivation

If the user entered a negative value for radius in ComputeArea.java, then you don't want the program to compute the area, but to inform the user that their input was incorrect.
Computing the Area of a Circle:

```java
import java.util.Scanner;

public class ComputeArea {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter a radius: ");
        double radius = input.nextDouble();
        if(radius > 0){
            double area = radius * radius * 3.14159;
            System.out.println("The area for the circle" + " of radius " + radius + " is " + area);
        } else
        System.out.println("The radius is negative." + "The area cannot be computed.");
    }
}
```
The boolean Type and Operators

• Often in a program you need to compare values:
  e.g., if \( x \) is greater than \( y \) then ...
• Java provides six comparison operators (called relational operators) to compare two values: \(<\), \(\leq\), \(>\), \(\geq\), \(==\) and \(!=\)
• The result of the comparison is a Boolean value: true or false. For example,
  \[
  \text{boolean } b = (1 > 2);
  \]
b is false after the statement.
## Comparison Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>equal to</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
</tr>
</tbody>
</table>
One-way \textbf{if} Statements

\begin{verbatim}
if (boolean-expression) {
  statement(s);
}
\end{verbatim}

\begin{verbatim}
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println("The area +" for the circle of radius + radius + " is " + area);
}
\end{verbatim}
One-way if Statements

**Condition containment is necessary!**

\[
\text{if } i > 0 \{ \\
\quad \text{System.out.println("i is positive");} \\
\}
\]

(a) Wrong

\[
\text{if } (i > 0) \{ \\
\quad \text{System.out.println("i is positive");} \\
\}
\]

(b) Correct

**Block containment is not necessary for a single statement!**

\[
\text{if } (i > 0) \{ \\
\quad \text{System.out.println("i is positive");} \\
\}
\]

(a)

\[
\text{if } (i > 0) \\
\quad \text{System.out.println("i is positive");}
\]

(b) Equivalent
Two-way if Statement

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

- Boolean Expression
- true
- false
- Statement(s) for the true case
- Statement(s) for the false case
if...else Example

if (radius >= 0) {
    double area = radius * radius * 3.1415;
    System.out.println("The area for the" + " circle of radius " + radius + " is " + area);
} else
    System.out.println("Negative input");
Multiple Alternative if Statements

Indentation in Java is not required, but a good programming style.

```java
if (score >= 90.0)
    grade = 'A';
else
    if (score >= 80.0)
        grade = 'B';
    else
        if (score >= 70.0)
            grade = 'C';
        else
            if (score >= 60.0)
                grade = 'D';
            else
                grade = 'F';
```

Equivalent

```java
if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';
```

Indentation exception for cascading else if statements:
Suppose score is 70.0

if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';

The condition is false
Suppose score is 70.0

The condition is false

```java
if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';
```
Suppose score is 70.0

if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';
Trace if-else statement

Suppose score is 70.0

if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';

grade is C
Trace if-else statement

Suppose score is 70.0

```java
if (score >= 90.0) 
    grade = 'A';
else if (score >= 80.0) 
    grade = 'B';
else if (score >= 70.0) 
    grade = 'C';
else if (score >= 60.0) 
    grade = 'D';
else
    grade = 'F';
```

Exit the if statement
if ... else

Indentation mistakes can get us confused, so the rule is that the else clause matches the most recent if clause in the same block.

```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");
```

Wrong indentation

```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");
```

Correct indentation

This does not print anything!
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");
```

This code prints B.
Common Error

• Adding a semicolon at the end of an if clause is a common mistake:
  
  ```java
  if (radius >= 0);
  {
    area = radius*radius*PI;
    System.out.println( 
      "The area for the circle of radius " + 
      radius + " is " + area);
  }
  ```
  
  Wrong

• It is not a compilation error or a runtime error, it is a logic error because ";" is a statement (no-operation statement)

• This error often occurs when you use the next-line block style
What’s wrong here?

System.out.print("Enter your total cholesterol level: ");
int totalCholesterol = input.nextInt();

if (totalCholesterol >= 200)
  System.out.println("Your cholesterol is too high.");
  System.out.println("You need to lower that.");
else
  System.out.println("Good, eat away!");
What’s wrong here?

```java
System.out.print("Enter your total cholesterol level: ");
int totalCholesterol = input.nextInt();

if (totalCholesterol >= 200)
    System.out.println("Your cholesterol is too high.");
    System.out.println("You need to lower that.");
else // COMPILER ERROR HERE: this else does not match any if
    System.out.println("Good, eat away!");
```
What’s wrong here?

System.out.print("Enter your total cholesterol level: ");
int totalCholesterol = input.nextInt();

if (totalCholesterol >= 200) {
    // Now it is correct
    System.out.println("Your cholesterol is too high.");
    System.out.println("You need to lower that.");
} else
    System.out.println("Good, eat away!");
System.out.print("Enter your total cholesterol level:");
int totalCholesterol = input.nextInt();

if (totalCholesterol >= 200)
    System.out.println("Your cholesterol is too high.");
    System.out.println("You need to lower that.");
Why is this worse?

System.out.print("Enter your total cholesterol level:");
int totalCholesterol= input.nextInt();

if (totalCholesterol>= 200)
    System.out.println("Your cholesterol is too high.");
System.out.println("You need to lower that.");
// No compiler error
// It is a Bug/logical error because it says to lower
// that even if it is fine.
Why is this worse?

```java
System.out.print("Enter your total cholesterol level: ");
int totalCholesterol = input.nextInt();

if (totalCholesterol >= 200) {
    // correct
    System.out.println("Your cholesterol is too high.");
    System.out.println("You need to lower that.");
}
```
What about complex conditions?

- For example: Computing Taxes: the US federal income tax is calculated based on the filing status and taxable income (we need multiple/complex logical conditions)

- There are four filing statuses: single filers, married filing jointly, married filing separately, and head of household combined with earnings.

<table>
<thead>
<tr>
<th>Marginal Tax Rate</th>
<th>Single (0)</th>
<th>Married Filing Jointly or Qualified Widow(er) (1)</th>
<th>Married Filing Separately (2)</th>
<th>Head of Household (3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>$0 – $8,350</td>
<td>$0 – $16,700</td>
<td>$0 – $8,350</td>
<td>$0 – $11,950</td>
</tr>
<tr>
<td>15%</td>
<td>$8,351 – $33,950</td>
<td>$16,701 – $67,900</td>
<td>$8,351 – $33,950</td>
<td>$11,951 – $45,500</td>
</tr>
<tr>
<td>35%</td>
<td>$372,951+</td>
<td>$372,951+</td>
<td>$186,476+</td>
<td>$372,951+</td>
</tr>
</tbody>
</table>
## Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>exclusive or</td>
</tr>
</tbody>
</table>
Truth Table for Operator !

<table>
<thead>
<tr>
<th>p</th>
<th>!p</th>
<th>Example (assume age = 24, gender = 'F')</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
<td>!(age &gt; 18) is false, because (age &gt; 18) is true.</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>!(gender != 'F') is true, because (grade != 'F') is false.</td>
</tr>
</tbody>
</table>
### Truth Table for Operator `&&`

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>p2</td>
<td>p1 &amp;&amp; p2</td>
<td>Example (assume age = 24, gender = 'F')</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>----------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>(age &gt; 18) &amp;&amp; (gender == 'F') is true, because (age &gt; 18) and (gender == 'F') are both true.</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>(age &gt; 18) &amp;&amp; (gender != 'F') is false, because (gender != 'F') is false.</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td></td>
</tr>
</tbody>
</table>
Truth Table for Operator \( \lor \lor \)

<table>
<thead>
<tr>
<th>( p_1 )</th>
<th>( p_2 )</th>
<th>( p_1 \lor p_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

Example (assume age = 24, gender = 'F')

\[ (age > 34) \lor (gender == 'F') \] is true, because \( (gender == 'F') \) is true.

\[ (age > 34) \lor (gender == 'M') \] is false, because \( (age > 34) \) and \( (gender == 'M') \) are both false.
### Truth Table for Operator \(^\)\n
<table>
<thead>
<tr>
<th>p1</th>
<th>p2</th>
<th>p1 (^) p2</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>

**Example (assume age = 24, gender = 'F')**

\((age > 34) \(^\) (gender == 'F')\) is true, because \((age > 34)\) is false but \((gender == 'F')\) is true.

\((age > 34) \(^\) (gender == 'M')\) is false, because \((age > 34)\) and \((gender == 'M')\) are both false.
What is the result?

boolean result;
result = (5 <= 9);
result = !(5 <= 9);
result = (3.9 > 3.19);
result = ('a' == 'A');
result = (5 <= 9 && 8 > 9);
result = (5 <= 9 || 8 > 9);
Logical Operators Examples

System.out.println("Is " + number + " divisible by 2 and 3? "+ ((number % 2 == 0) && (number % 3 == 0)));

System.out.println("Is " + number + " divisible by 2 or 3? "+ ((number % 2 == 0) || (number % 3 == 0)));

System.out.println("Is " + number + " divisible by 2 or 3, but not both? "+ ((number % 2 == 0) ^ (number % 3 == 0)));
Determining Leap Year

This program first prompts the user to enter a year as an int value and checks if it is a leap year.

A year is a leap year if it is divisible by 4 but not by 100, or it is divisible by 400.

\[
(\text{year} \mod 4 == 0 \land \land \text{year} \mod 100 != 0 ) \\
\lor \lor \text{year} \mod 400 == 0
\]
The unconditional & and | Operators

• The & operator works exactly the same as the && operator, and the | operator works exactly the same as the || operator with one exception:
  • the & and | operators always evaluate both operands
The unconditional & and | Operators

If x is 1, what is x after these expressions:

1. \((x > 1) \&\& (x++ < 10)\)  
2. \((x > 1) \& (x++ < 10)\)  
3. \((1 == x) \|\| (10 > x++)\)  
4. \((1 == x) \| (10 > x++)\)
switch Statements

```java
switch (var) {
    case 0:    ...;
                break;
    case 1:    ...;
                break;
    case 2:    ...;
                break;
    case 3:    ...;
                break;
    default:   ...;
}
```
switch Statement Flow Chart

- var is 0
- break
- Compute tax for married file separately
- break
- var is 1
- break
- var is 2
- break
- var is 3
- break
- default
- break
- Next Statement
switch Statement Rules

char, byte, short, int, String

value1, ..., and valueN are constant expressions of the same data type as the value of the switch-expression constant = they cannot contain variables in the expression, such as x+y

```
switch (switch-expression) {
    case value1:    statement(s)1;
                    break;

    case value2:    statement(s)2;
                    break;

    ...

    case valueN:    statement(s)N;
                    break;

    default:        statement(s);
}
```
The `switch` statement in programming allows for multiple cases to be evaluated in a single `if` statement. Here are the key rules:

- **break** is optional, but it terminates the remainder of the `switch` statement.
- **default** is optional and executed when none of the specified cases match the `switch` expression.

The syntax for a `switch` statement is as follows:

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

Execution proceeds in sequential order, following the first case that matches the `switch` expression. Once a case is matched and its `break` statement is executed, the remaining cases are not evaluated.
Trace switch statement

Suppose ch is 'a':

```java
switch (ch) {
    case 'a':    System.out.println(ch);
    case 'b':    System.out.println(ch);
    case 'c':    System.out.println(ch);
}
```
Trace switch statement

```java
switch (ch) {
    case 'a':
        System.out.println(ch);
    case 'b':
        System.out.println(ch);
    case 'c':
        System.out.println(ch);
}
```

ch is 'a':
Trace switch statement

```java
switch (ch) {
    case 'a': System.out.println(ch);
    case 'b': System.out.println(ch);
    case 'c': System.out.println(ch);
}
```

Execute this line
Trace switch statement

```java
switch (ch) {
    case 'a': System.out.println(ch);
    case 'b': System.out.println(ch);
    case 'c': System.out.println(ch);
}
```

Execute this line
Trace switch statement

```
switch (ch) {
    case 'a':    System.out.println(ch);
    case 'b':    System.out.println(ch);
    case 'c':    System.out.println(ch);
}
```
switch (ch) {
    case 'a':    System.out.println(ch);
    case 'b':    System.out.println(ch);
    case 'c':    System.out.println(ch);
}
Suppose ch is 'a':

```java
switch (ch) {
    case 'a':
        System.out.println(ch);
        break;
    case 'b':
        System.out.println(ch);
        break;
    case 'c':
        System.out.println(ch);
}
```
Trace switch statement 2

```java
switch (ch) {
    case 'a':
        System.out.println(ch);
        break;
    case 'b':
        System.out.println(ch);
        break;
    case 'c':
        System.out.println(ch);
}
```

Execute this line
Trace switch statement 2

```
switch (ch) {
    case 'a':  System.out.println(ch);
        break;
    case 'b':  System.out.println(ch);
        break;
    case 'c':  System.out.println(ch);
}
```

Execute this line
switch (ch) {
    case 'a':    System.out.println(ch);
                 break;
    case 'b':    System.out.println(ch);
                 break;
    case 'c':    System.out.println(ch);
}

Conditional Operator

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

is equivalent to

\ y = (x > 0) \ ? \ 1 : \ -1;

(\text{boolean-expression}) \ ? \ \text{expression1} : \ \text{expression2}

System.out.println( \\
(num \% 2 == 0)? num + " \ is even" : num + " \ is odd"); \\
System.out.println( num + \\
\ ((num \% 2 == 0)? " \ is even" : " \ is odd"));
Operator Precedence

- var++, var--
- +, - (Unary plus and minus), ++var, --var
- (type) Casting
- ! (Not)
- *, /, % (Multiplication, division, and remainder)
- +, - (Binary addition and subtraction)
- <, <=, >, >= (Comparison)
- ==, !=; (Equality)
- ^ (Exclusive OR)
- && (Conditional AND) Short-circuit AND
- || (Conditional OR) Short-circuit OR
- =, +=, -=, *=, /=, %= (Assignment operator)

All binary operators except assignment operators are left-associative.
Example

Applying the operator precedence and associativity rule, the expression \(3 + 4 \times 4 > 5 \times (4 + 3) - 1\) is evaluated as follows:

\[
\begin{align*}
3 + 4 \times 4 &> 5 \times (4 + 3) - 1 \\
3 + 4 \times 4 &> 5 \times 7 - 1 \\
3 + 16 &> 5 \times 7 - 1 \\
3 + 16 &> 35 - 1 \\
19 &> 35 - 1 \\
19 &> 34 \\
\text{false}
\end{align*}
\]

(1) inside parentheses first
(2) multiplication
(3) multiplication
(4) addition
(5) subtraction
(6) greater than
Operator Associativity

All binary operators except assignment operators are **left-associative**.

\[ 10 - 5 - 4 = (10 - 5) - 4 = 5 - 4 = 1 \]

The assignment operators are **right-associative**:

\[ a = b + = c = 5; \text{ is equivalent to } a = (b + = (c = 5)) \]
ChangeMaker Example Revisited

```java
System.out.print("Input change amount (1-99): ");
originalAmount = scanner.readInt;
if (originalAmount< 1 || originalAmount> 99)
    System.out.println("ERROR: Out of range.");
else{
    numQuarters = originalAmount/ 25;
    remainder = originalAmount% 25;
    numDimes = remainder / 10;
    remainder = remainder % 10;
    numNickels = remainder / 5;
    remainder = remainder % 5;
    if (numQuarters!= 0) // Do not print if zero
        System.out.println(numQuarters+ " quarters");
    if (numDimes!= 0)// Do not print if zero
        System.out.println(numDimes+ " dimes");
    if (numNickels!= 0)// Do not print if zero
        System.out.println(numNickels+ " nickels");
    if (numPennies!= 0)// Do not print if zero
        System.out.println(numPennies+ " pennies");
}
```
Nested ifs:

```java
if (numQuarters!= 0){ // Do not print if zero
    System.out.print(numQuarters+ " quarter");
    if (numQuarters== 1) // Do not print s if one
        System.out.println( );
    else
        System.out.println("s"); // print s if more
}
```