Java Programing 1

ITC 5102

Lecture 5: Decision Structure

Instructor: Parisa Pouladzadeh
Email: parisa.pouladzadeh@humber.ca

Topics

- The `if` Statement
- The `if-else` Statement
- Nested `if` Statements
- The `if-else-if` Statement
- Logical Operators
- Comparing `String` Objects

Topics (cont'd)

- More about Variable Declaration and Scope
- The Conditional Operator
- The `switch` Statement

The `if` Statement

The `if` statement decides whether a section of code executes or not.

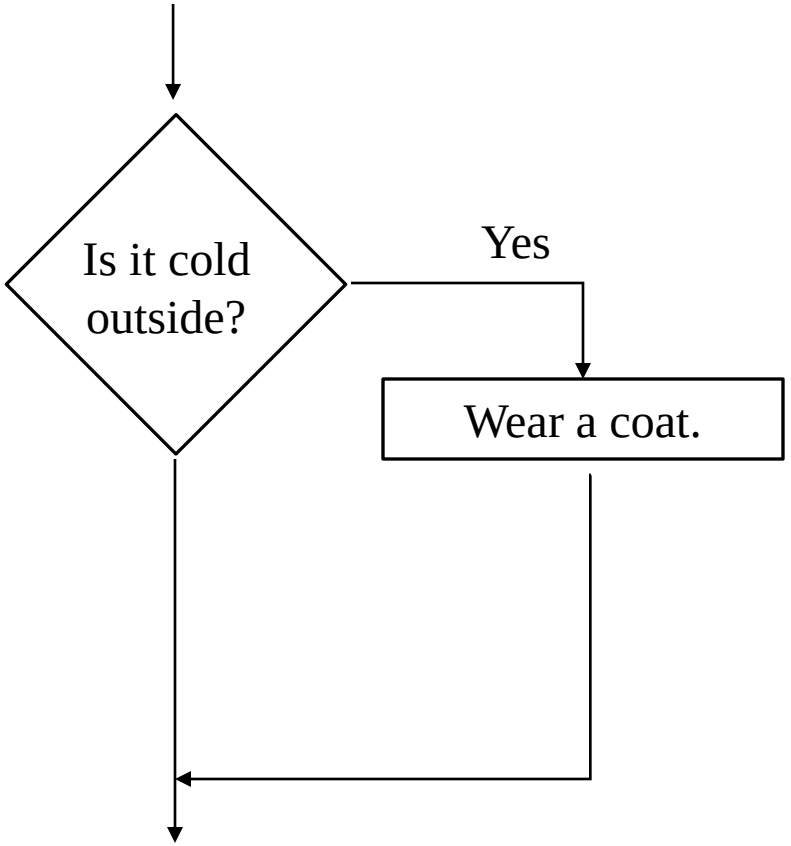
The `if` statement uses a `boolean` to decide whether the next statement or block of statements executes.

- *if (boolean expression is true)*
- *execute next statement.*

Flowcharts

If statements can be modeled as a flow chart.

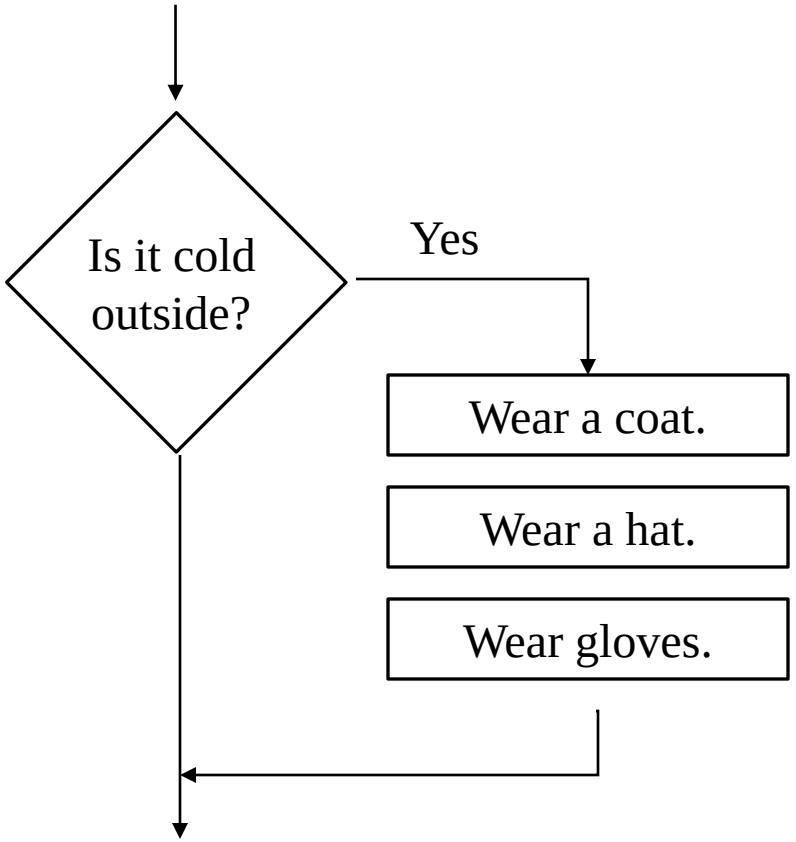
```
if (coldOutside)
  wearCoat();
```



Flowcharts (cont'd)

A block `if` statement may be modeled as:

```
if (coldOutside)
{
   wearCoat();
   wearHat();
   wearGloves();
}
```



Note the use of curly braces to block several statements together.

Relational Operators

In most cases, the `boolean` expression, used by the `if` statement, uses *relational operators.*

Relational Operator	Meaning
>	is greater than
<	is less than
>=	is greater than or equal to
<=	is less than or equal to
==	is equal to
!=	is not equal to

Boolean Expressions

A *boolean expression* is any variable or calculation that results in a *true* or *false* condition.

Expression	Meaning
`x > y`	Is x greater than y?
`x < y`	Is x less than y?
`x >= y`	Is x greater than or equal to y?
`x <= y`	Is x less than or equal to y.
`x == y`	Is x equal to y?
`x != y`	Is x not equal to y?

`if` Statements and Boolean Expressions

```
if (x > y)
  System.out.println("X is greater than Y");

if(x == y)
  System.out.println("X is equal to Y");

if(x != y)
{
  System.out.println("X is not equal to Y");
  x = y;
  System.out.println("However, now it is.");
}
```

Example: AverageScore.java

```java
import javax.swing.JOptionPane; // Needed for JOptionPane

/**
This program demonstrates the if statement.
*/

public class AverageScore
{
public static void main(String[] args)
{
double score1; // To hold score #1
double score2; // To hold score #2
double score3; // To hold score #3
double average; // To hold the average score
String input; // To hold the user's input

// Get the first test score.
input = JOptionPane.showInputDialog("Enter score #1:");
score1 = Double.parseDouble(input);

// Get the second score.
input = JOptionPane.showInputDialog("Enter score #2:");
score2 = Double.parseDouble(input);

// Get the third test score.
input = JOptionPane.showInputDialog("Enter score #3:");
score3 = Double.parseDouble(input);

// Calculate the average score.
average = (score1 + score2 + score3) / 3.0;

// Display the average score.
JOptionPane.showMessageDialog(null, "The average is " + average);

// If the score was greater than 95, let the user know
// that's a great score.
if (average > 95)
JOptionPane.showMessageDialog(null,"That's a great score!");

System.exit(0);
}
}
```

Programming Style and `if` Statements

An `if` statement can span more than one line; however, it is still one statement.

```
if (average > 95)
   grade = 'A';
```

is functionally equivalent to

```
if(average > 95) grade = 'A';
```

Programming Style and `if` Statements (cont'd)

Rules of thumb:

- The conditionally executed statement should be on the line after the `if` condition.
- The conditionally executed statement should be indented one level from the `if` condition.
- If an `if` statement does not have the block curly braces, it is ended by the first semicolon encountered after the `if` condition.

```
if (expression)
  statement;
```

No semicolon here.

Semicolon ends statement here.

Having Multiple Conditionally-Executed Statements

- Conditionally executed statements can be grouped into a block by using curly braces `{ }` to enclose them.
- If curly braces are used to group conditionally executed statements, the `if` statement is ended by the closing curly brace.

```
if (expression)
{
  statement1;
  statement2;
}
```

Curly brace ends the statement.

Having Multiple Conditionally-Executed Statements (cont'd)

- Remember that when the curly braces are not used, then only the next statement after the `if` condition will be executed conditionally.

```
if (expression)
  statement1;
  statement2;
  statement3;
```

Only this statement is conditionally executed.

Flags

A flag is a `boolean` variable that monitors some condition in a program.

When a condition is true, the flag is set to `true`.

The flag can be tested to see if the condition has changed.

```
if (average > 95)
   highScore = true;
```

Later, this condition can be tested:

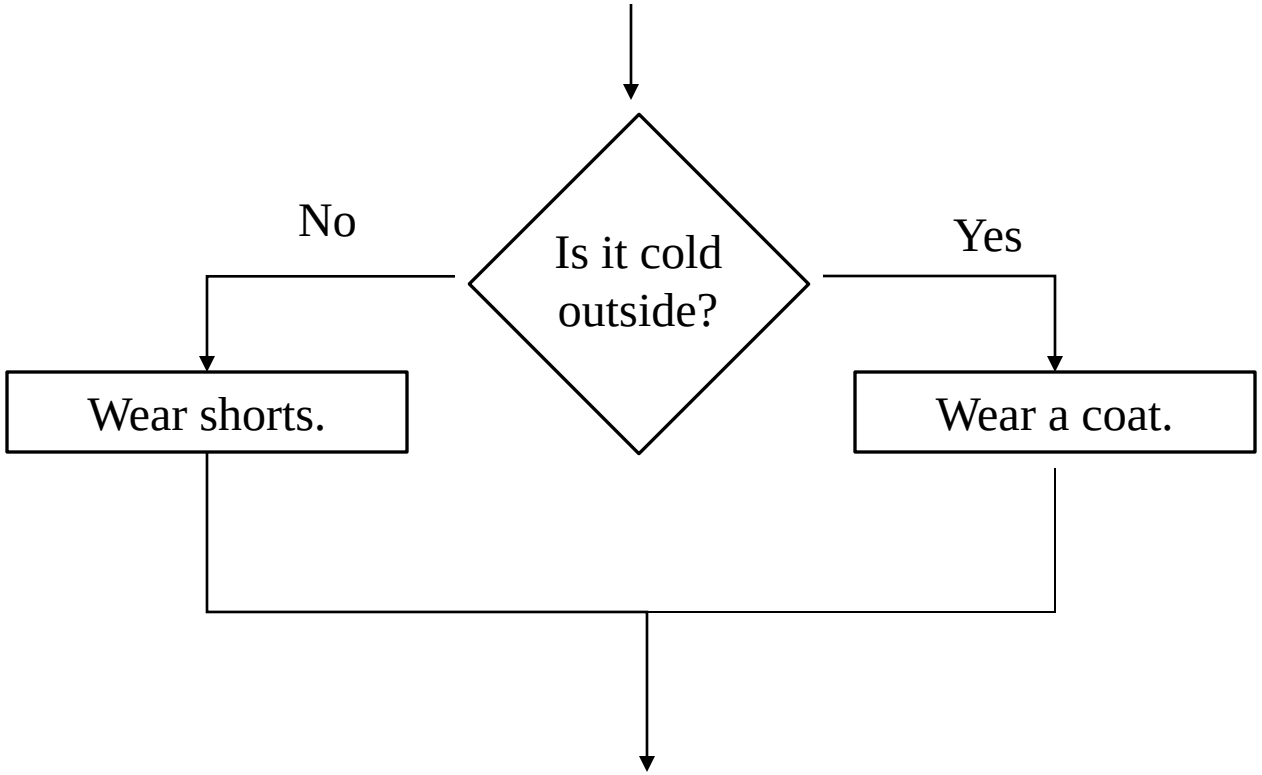
```
if (highScore)
   System.out.println("That's a high score!");
```

`if-else` Statements

The `if-else` statement adds the ability to conditionally execute code when the `if` condition is false.

```
if (expression)
        statementOrBlockIfTrue;
else
        statementOrBlockIfFalse;
```

`if-else` Statement Flowcharts



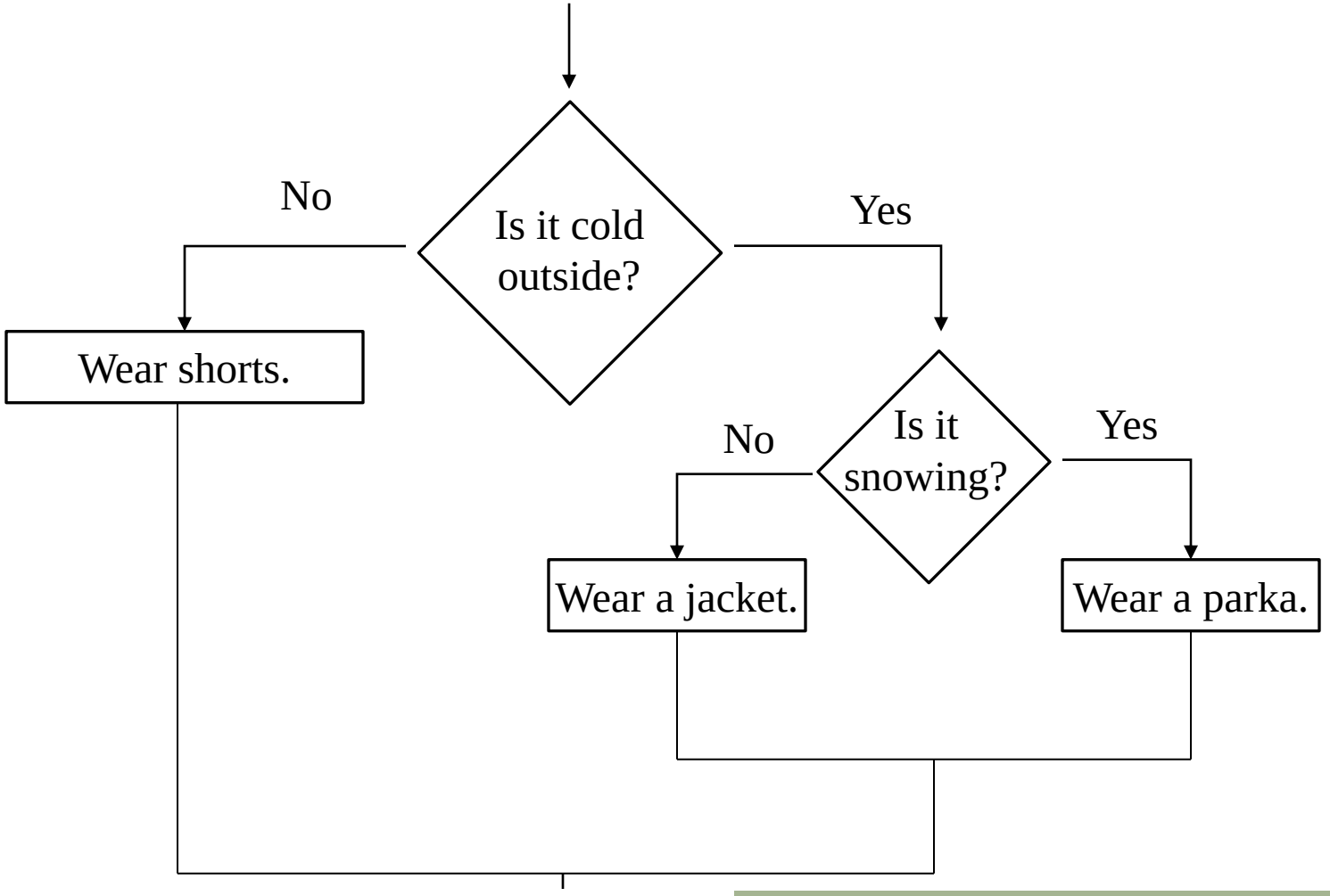
Nested `if` Statements

If an `if` statement appears inside another `if` statement (single or block) it is called a *nested `if` statement*.

The nested `if` is executed only if the outer `if` statement results in a true condition.

See example: LoanQualifier.java

Nested `if` Statement Flowcharts



```
import javax.swing.JOptionPane; // Needed for JOptionPane class

/**

This program demonstrates a nested if statement.
*/

public class LoanQualifier
{
public static void main(String[] args)
{
double salary; // Annual salary
double yearsOnJob; // Years at current job

String input; // To hold string input

// Get the user's annual salary.
input = JOptionPane.showInputDialog("Enter your " + "annual salary.");
salary = Double.parseDouble(input);

// Get the number of years at the current job.
input = JOptionPane.showInputDialog("Enter the number of " + "years at your current job.");
yearsOnJob = Double.parseDouble(input);

// Determine whether the user qualifies for the loan.
if (salary >= 30000)
{
if (yearsOnJob >= 2)
{
JOptionPane.showMessageDialog(null, "You qualify " + "for the loan.");
}
else
{
JOptionPane.showMessageDialog(null, "You must have " + "been on your current job for at least " + "two years to qualify.");
}
}
else
{
JOptionPane.showMessageDialog(null, "You must earn " +
"at least $30,000 per year to qualify.");
}

System.exit(0);
}
}
```

`if-else` Matching

Curly brace use is not required if there is only one statement to be conditionally executed.

However, sometimes curly braces can help make the program more readable.

`if-else-if` Statements

`if-else-if` statements can become very complex.

Imagine the following decision set.

- *if it is very cold, wear a heavy coat,*
- *else, if it is chilly, wear a light jacket,*
- *else, if it is windy wear a windbreaker,*
- *else, if it is hot, wear no jacket.*

`if-else-if` Statements

```
if (expression)
  statement or block
else if (expression)
    statement or block
    // Put as many else ifs as needed here
else
    statement or block
```

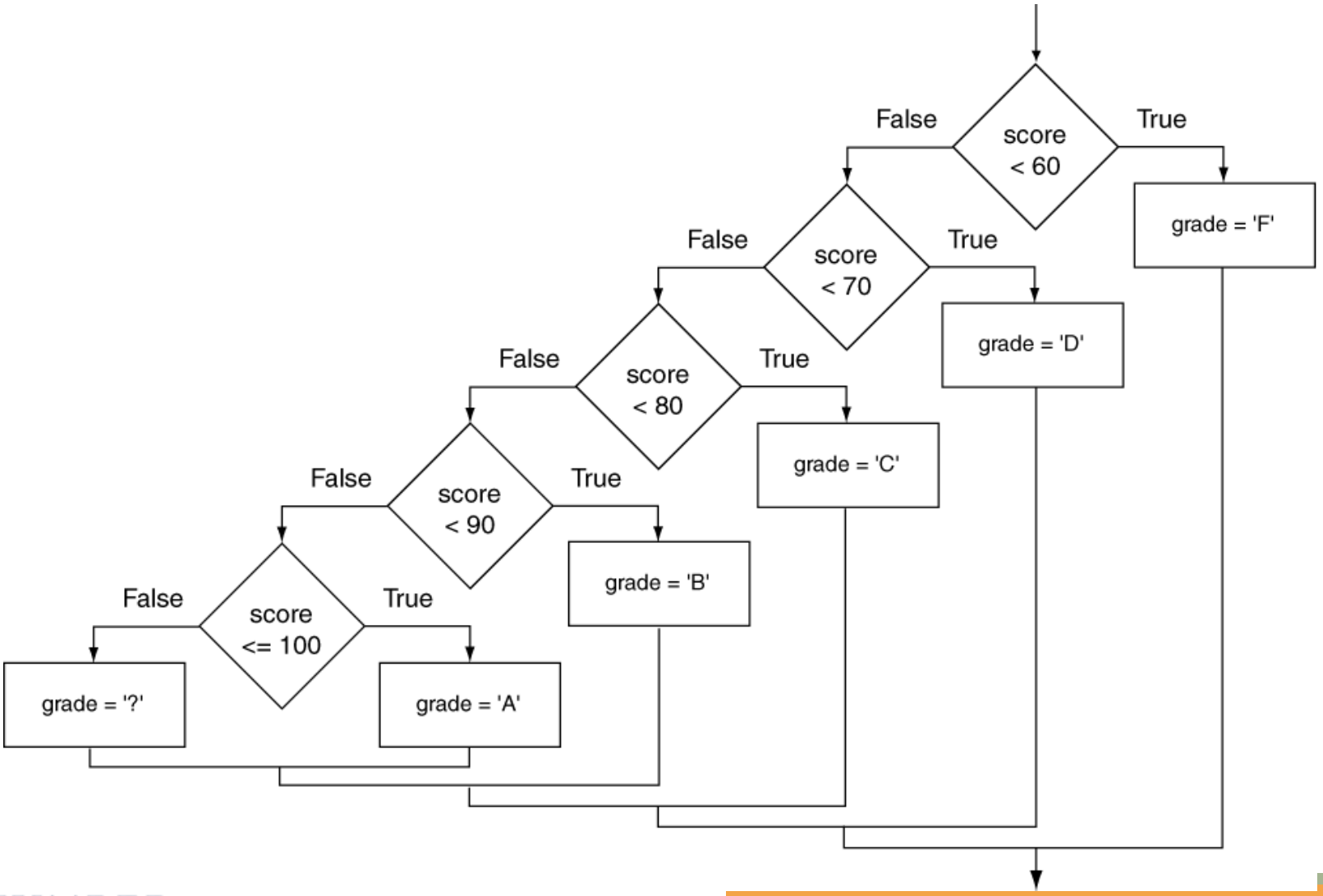
Care must be used since `else` statements match up with the immediately preceding unmatched `if` statement.

See example:

TestGrade.java, TestResults.java

```java
import javax.swing.JOptionPane; // Needed for JOptionPane

/**
This program asks the user to enter a numeric test
score and displays a letter grade for the score. The
program uses an if-else-if statement to determine
the letter grade.
*/

 public class TestResults
 {
 public static void main(String[] args)
 {
 int testScore; // Numeric test score
 String input; // To hold the user's input

 // Get the numeric test score.
 input = JOptionPane.showInputDialog("Enter your numeric " +
 "test score and I will tell you the grade: ");
 testScore = Integer.parseInt(input);

 // Display the grade.
 if (testScore < 60)
 JOptionPane.showMessageDialog(null, "Your grade is F.");
 else if (testScore < 70)
 JOptionPane.showMessageDialog(null, "Your grade is D.");
 else if (testScore < 80)
 JOptionPane.showMessageDialog(null, "Your grade is C.");
 else if (testScore < 90)
 JOptionPane.showMessageDialog(null, "Your grade is B.");
 else
 JOptionPane.showMessageDialog(null, "Your grade is A.");

 System.exit(0);
 }
 }
```

`if-else-if` Flowchart



Logical Operators

Java provides two binary *logical operators* (`&&` and `||`) that are used to combine `boolean` expressions.

Java also provides one *unary* (`!`) logical operator to reverse the truth of a `boolean` expression.

Logical Operators

Operator	Meaning	Effect
&&	**AND**	Connects two `boolean` expressions into one. Both expressions must be true for the overall expression to be true.
||	**OR**	Connects two `boolean` expressions into one. One or both expressions must be true for the overall expression to be true. It is only necessary for one to be true, and it does not matter which one.
!	**NOT**	The ! operator reverses the truth of a `boolean` expression. If it is applied to an expression that is true, the operator returns false. If it is applied to an expression that is false, the operator returns true.

The && Operator

The logical AND operator (`&&`) takes two operands that must both be `boolean` expressions.

The resulting combined expression is true if (and *only* if) both operands are true.

See example: LogicalAnd.java

Expression 1	Expression 2	Expression1 && Expression2
true	false	false
false	true	false
false	false	false
true	true	true

```
import javax.swing.JOptionPane; // Needed for JOptionPane class

/**
This program demonstrates the logical && operator.
*/

public class LogicalAnd
{
public static void main(String[] args)
 {
 double salary; // Annual salary
 double yearsOnJob; // Years at current job
 String input; // To hold string input

 // Get the user's annual salary.
 input = JOptionPane.showInputDialog("Enter your " +
 "annual salary.");
 salary = Double.parseDouble(input);

 // Get the number of years at the current job.
 input = JOptionPane.showInputDialog("Enter the number of " +
 "years at your current job.");
 yearsOnJob = Double.parseDouble(input);

 // Determine whether the user qualifies for the loan.
 if (salary >= 30000 && yearsOnJob >= 2)
 {
 JOptionPane.showMessageDialog(null, "You qualify " +
 "for the loan.");
 }
 else
 {
 JOptionPane.showMessageDialog(null, "You do not " +
 "qualify for the loan.");
 }

 System.exit(0);
 }
 }
```

The || Operator

The logical OR operator (`||`) takes two operands that must both be `boolean` expressions.

The resulting combined expression is false if (and *only* if) both operands are false.

Example: LogicalOr.java

Expression 1	Expression 2	Expression1 || Expression2
true	false	true
false	true	true
false	false	false
true	true	true

```java
import javax.swing.JOptionPane; // Needed for JOptionPane class

/**
This program demonstrates the logical || operator.
*/

public class LogicalOr
{
public static void main(String[] args)
 {

 double salary; // Annual salary
 double yearsOnJob; // Years at current job
 String input; // To hold string input

 // Get the user's annual salary.
 input = JOptionPane.showInputDialog("Enter your " +
 "annual salary.");
 salary = Double.parseDouble(input);

 // Get the number of years at the current job.
 input = JOptionPane.showInputDialog("Enter the number of " +
 "years at your current job.");
 yearsOnJob = Double.parseDouble(input);

 // Determine whether the user qualifies for loan.
 if (salary >= 30000 || yearsOnJob >= 2)
 {
 JOptionPane.showMessageDialog(null, "You qualify " +
 "for the loan.");
 }
 else
 {
 JOptionPane.showMessageDialog(null, "You do not " +
 "qualify for the loan.");
 }

 System.exit(0);
 }
 }
```

The ! Operator

The `!` operator performs a logical NOT operation.

If an *`expression`* is true, `!`*`expression`* will be false.

```
if (!(temperature > 100))
    System.out.println("Below the maximum temperature.");
```

If `temperature > 100` evaluates to false, then the output statement will be run.

Expression 1	!Expression1
true	false
false	true

Short Circuiting

Logical AND and logical OR operations perform *short-circuit evaluation* of expressions.

Logical AND will evaluate to false as soon as it sees that one of its operands is a false expression.

Logical OR will evaluate to true as soon as it sees that one of its operands is a true expression.

Order of Precedence

The ! operator has a higher order of precedence than the && and || operators.

The && and || operators have a lower precedence than relational operators like < and >.

Parenthesis can be used to force the precedence to be changed.

Order of Precedence

Order of Precedence	Operators	Description
1	**(unary negation) !**	Unary negation, logical NOT
2	*** / %**	Multiplication, Division, Modulus
3	**+ -**	Addition, Subtraction
4	**< > <= >=**	Less-than, Greater-than, Less-than or equal to, Greater-than or equal to
5	**== !=**	Is equal to, Is not equal to
6	**&&**	Logical AND
7	**||**	Logical NOT
8	**= += -= *= /= %=**	Assignment and combined assignment operators.

Variable Scope

In Java, a local variable does not have to be declared at the beginning of the method.

The scope of a local variable begins at the point it is declared and terminates at the end of the method.

When a program enters a section of code where a variable has scope, that variable has *come into scope*, which means the variable is visible to the program.

See example: VariableScope.java

```
import javax.swing.JOptionPane; // Needed for JOptionPane

/**
This program demonstrates how variables may be declared in various locations
throughout a program.
*/

public class VariableScope
{
 public static void main(String[] args)
 {
 // Get the user's first name.
 String firstName;
 firstName = JOptionPane.showInputDialog("Enter your " +
 "first name.");

 // Get the user's last name.
 String lastName;
 lastName = JOptionPane.showInputDialog("Enter your " +
 "last name.");

 JOptionPane.showMessageDialog(null, "Hello, " + firstName +
 " " + lastName);
 System.exit(0);
 }
 }
```

The Conditional Operator

The *conditional operator* is a ternary (three operand) operator.

You can use the conditional operator to write a simple statement that works like an `if-else` statement.

The format of the operators is:

```
expression1 ? expression2 :
expression3
```

The conditional operator can also return a value.

The Conditional Operator

The conditional operator can be used as a shortened `if-else` statement:

```
x > y ? z = 10 : z = 5;
```

This line is functionally equivalent to:

```
if(x > y)
   z = 10;
else
   z = 5;
```

The Conditional Operator

Many times the conditional operator is used to supply a value.

```
number = x > y ? 10 : 5;
```

This is functionally equivalent to:

```
if(x > y)
    number = 10;
else
    number = 5;
```

See example: ConsultantCharges.java

The `switch` Statement

The `if-else` statement allows you to make true / false branches.

The `switch` statement allows you to use an ordinal value to determine how a program will branch.

The `switch` statement can evaluate a `char`, `byte`, `short`, `int`, or string value and make decisions based on the value.

The `switch` Statement

- The `switch` statement takes the form:

```
switch (testExpression)
{
  case Value_1:
    // place one or more statements here
    break;
  case Value_2:
    // place one or more statements here
    break;
    // case statements may be repeated
    //as many times as necessary
  default:
    // place one or more statements here
}
```

The `switch` Statement

- The *`testExpression`* is a variable or expression that gives a `char`, `byte`, `short`, `int` or string value.

```
switch (testExpression)
{
    …
}
```

- The `switch` statement will evaluate the *`testExpression`*.

- If there is an associated `case` statement that matches that value, program execution will be transferred to that `case` statement.

The `switch` Statement

Each `case` statement will have a corresponding `case` value that must be unique.

```
case value_1:
    // place one or more statements here
    break;
```

If the *`testExpression`* matches the `case` value, the Java statements between the colon and the `break` statement will be executed.

The `case` Statement

The `break` statement ends the `case` statement.

The `break` statement is optional.

If a `case` does not contain a `break`, then program execution continues into the next `case`.

- See example: NoBreaks.java
- See example: PetFood.java

The `default` section is optional and will be executed if no *CaseExpression* matches the *SwitchExpression*.

See example: SwitchDemo.java

```java
import java.util.Scanner; // Needed for Scanner class

/**
This program demonstrates the switch statement.
*/

public class SwitchDemo {
public static void main(String[] args) {
int number; // A number entered by the user

 // Create a Scanner object for keyboard input.
Scanner keyboard = new Scanner(System.in);

 // Get one of the numbers 1, 2, or 3 from the user.
System.out.print("Enter 1, 2, or 3: ");
 number = keyboard.nextInt();

 // Determine the number entered.
switch (number) {
case 1:
System.out.println("You entered 1.");

break;
case 2:
System.out.println("You entered 2.");
break;
case 3:
System.out.println("You entered 3.");
break;
default:
System.out.println("That's not 1, 2, or 3!");
}
}
}
```

```java
import java.util.Scanner; // Needed for Scanner class

/**
This program demonstrates the switch statement.
*/

public class NoBreaks
{
public static void main(String[] args)
 {
 int number; // A number entered by the user

 // Create a Scanner object for keyboard input.
 Scanner keyboard = new Scanner(System.in);

 // Get one of the numbers 1, 2, or 3 from the user.
 System.out.print("Enter 1, 2, or 3: ");
 number = keyboard.nextInt();

 // Determine the number entered.
 switch (number)
 {
 case 1:
 System.out.println("You entered 1.");
 case 2:
 System.out.println("You entered 2.");
 case 3:
 System.out.println("You entered 3.");
 default:
 System.out.println("That's not 1, 2, or 3!");
 }
 }
 }
```