Chapter 3:
Decision Structures

Starting Out with Java:
From Control Structures through Objects

Fifth Edition

by Tony Gaddis

PEARSON ALWAYS LEARNING

Troy University Resources

Chapter Topics

Chapter 3 discusses the following main topics:

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

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Chapter Topics

Chapter 3 discusses the following main topics:

- More about Variable Declaration and Scope
- The Conditional Operator
- The `switch` Statement
- The `printf` Method
- The `DecimalFormat` Class

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

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)

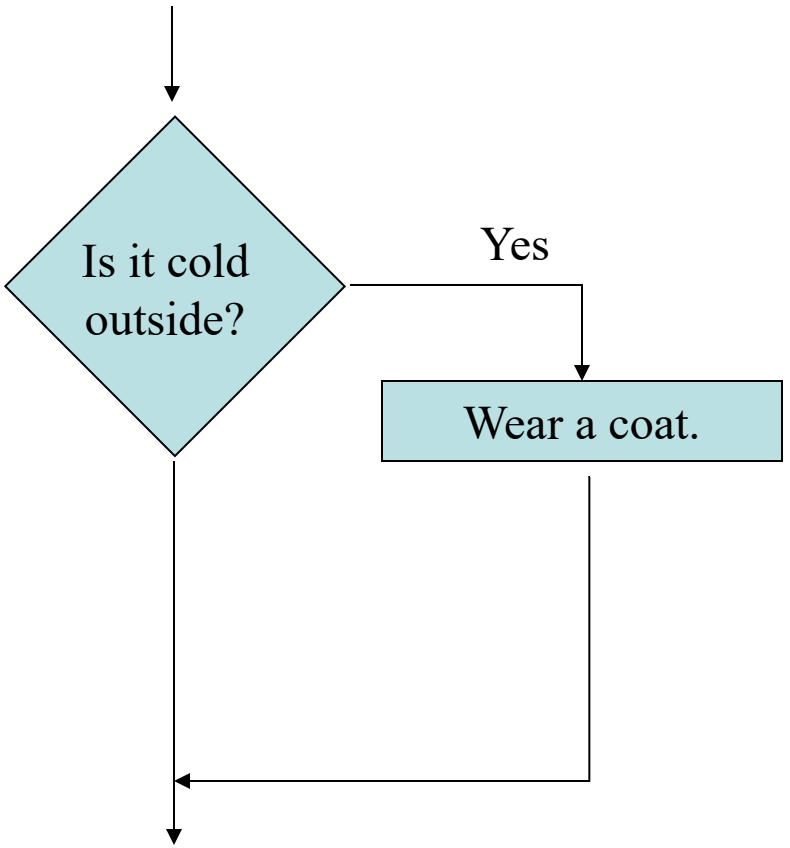
execute next statement.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Flowcharts

- If statements can be modeled as a flow chart.

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

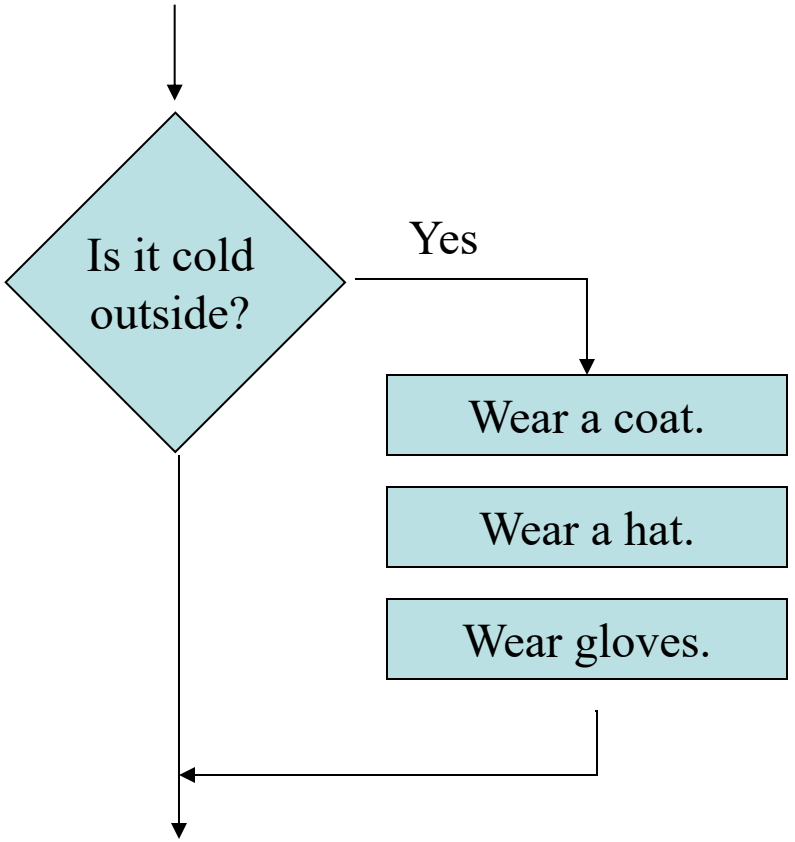


©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Flowcharts

- A block `if` statement may be modeled as:

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



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

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

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

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

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?

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

`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

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

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';
```

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Programming Style and `if` Statements

- 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.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

Block `if` 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.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Block `if` Statements

- 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.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

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!");
```

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Comparing Characters

- Characters can be tested with relational operators.
- Characters are stored in memory using the Unicode character format.
- Unicode is stored as a sixteen (16) bit number.
- Characters are *ordinal*, meaning they have an order in the Unicode character set.
- Since characters are ordinal, they can be compared to each other.

```
char c = 'A';
if(c < 'Z')
  System.out.println("A is less than Z");
```

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

`if-else` Statements

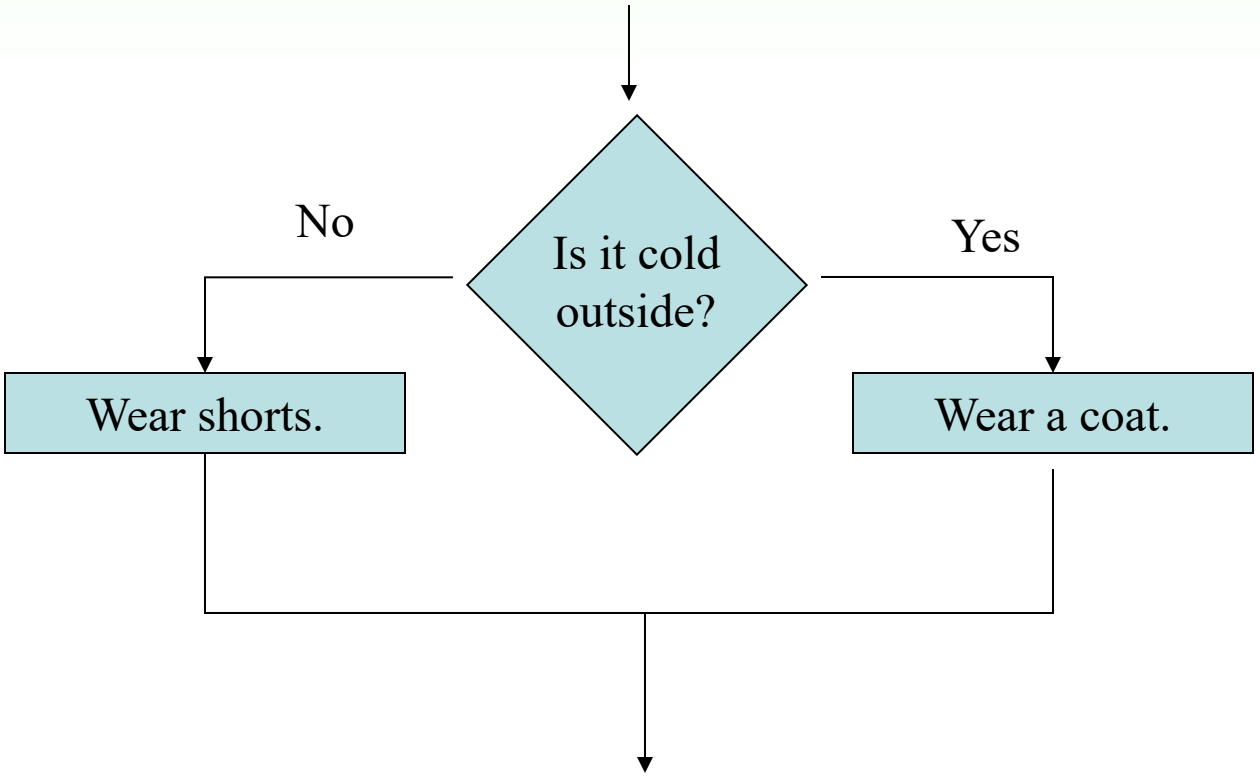
- The `if-else` statement adds the ability to conditionally execute code when the `if` condition is false.

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

- See example: Division.java

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

`if-else` Statement Flowcharts



©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

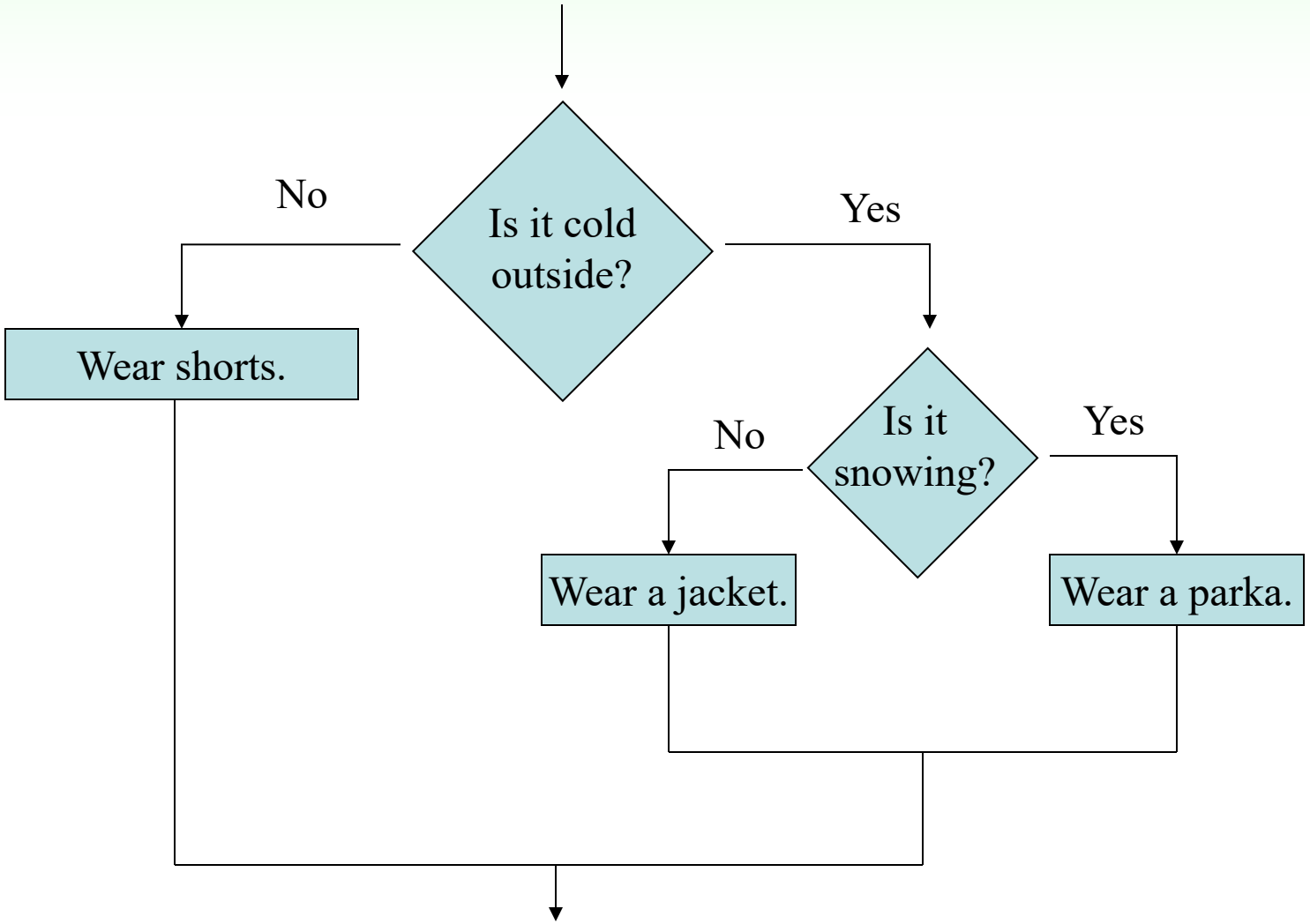
Troy University Resources

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

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Nested `if` Statement Flowcharts



©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

Nested `if` Statements

```
if (coldOutside)
{
     if (snowing)
     {
          wearParka();
     }
     else
     {
          wearJacket();
     }
}
else
{
     wearShorts();
}
```

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

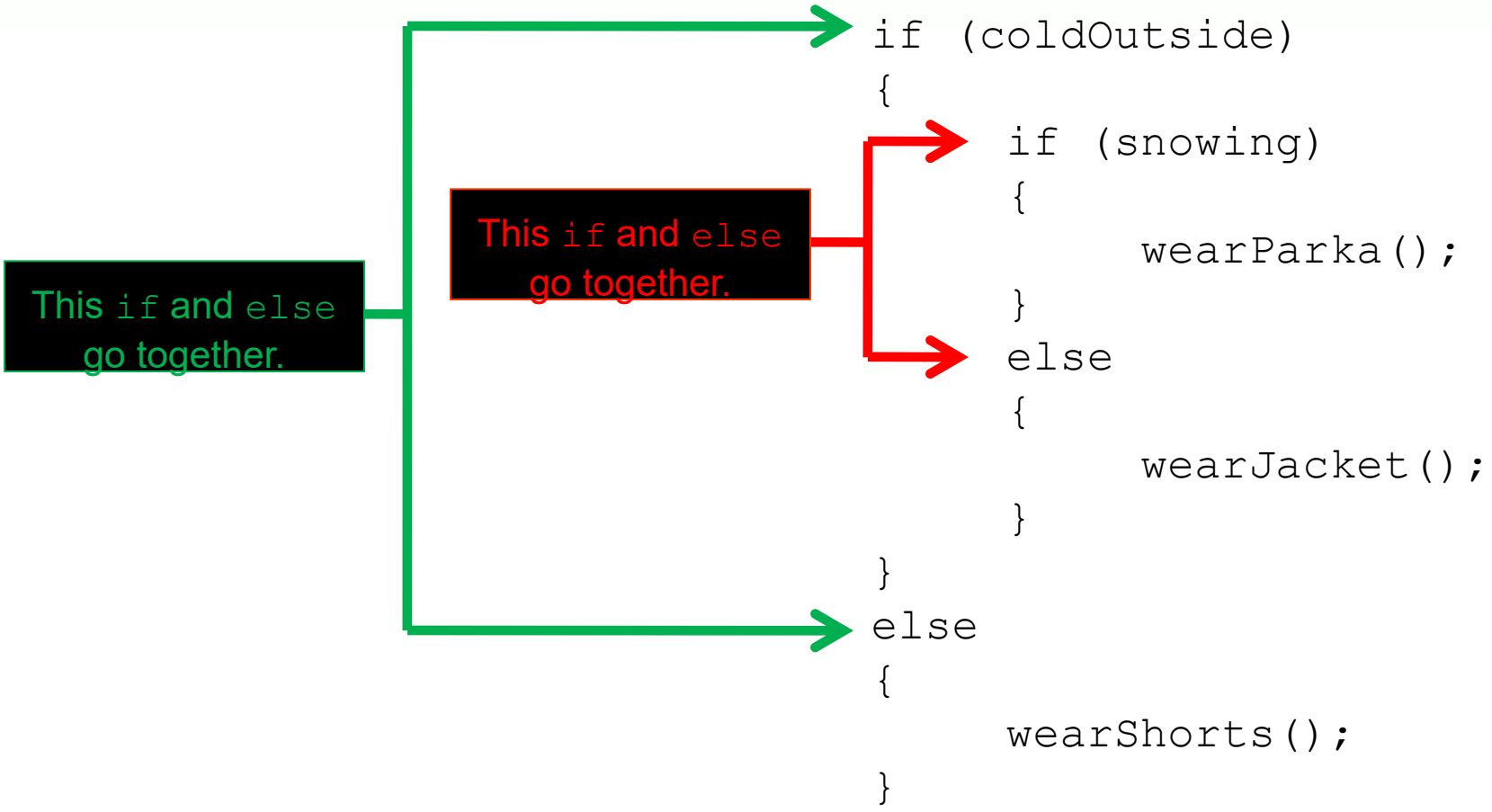
`if-else` Matching

- 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.
- Additionally, proper indentation makes it much easier to match up else statements with their corresponding `if` statement.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

Alignment and Nested `if` Statements



©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

`if-else-if` Statements

```
if (expression_1)
{
   statement;
   statement;
   etc.
}
else if (expression_2)
{
   statement;
   statement;
   etc.
}
```

If `expression_1` is true these statements are executed, and the rest of the structure is ignored.

Otherwise, if `expression_2` is true these statements are executed, and the rest of the structure is ignored.

Insert as many else if clauses as necessary

```
else
{
   statement;
   statement;
   etc.
}
```

These statements are executed if none of the expressions above are true.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

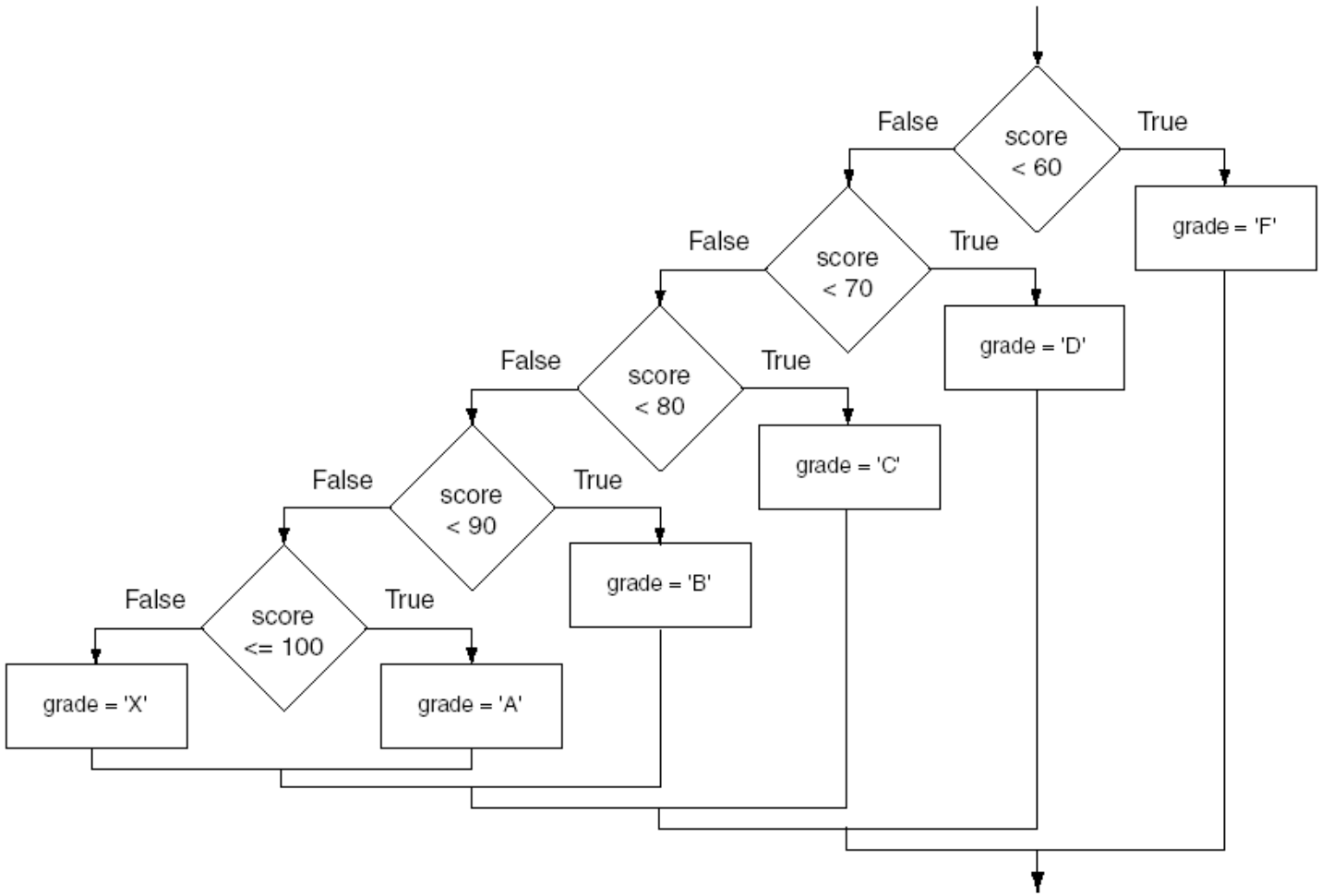
Troy University Resources

`if-else-if` Statements

- Nested `if` statements can become very complex.
- The `if-else-if` statement makes certain types of nested decision logic simpler to write.
- Care must be used since `else` statements match up with the immediately preceding unmatched `if` statement.
- See example: TestResults.java

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

`if-else-if` Flowchart



©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

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.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

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.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

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**

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

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

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

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

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

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.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

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.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

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.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

Comparing `String` Objects

- In most cases, you cannot use the relational operators to compare two `String` objects.
- Reference variables contain the address of the object they represent.
- Unless the references point to the same object, the relational operators will not return true.
- See example: StringCompare.java
- See example: StringCompareTo.java

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Ignoring Case in String Comparisons

- In the `String` class the `equals` and `compareTo` methods are case sensitive.
- In order to compare two `String` objects that might have different case, use:
  - `equalsIgnoreCase`, or
  - `compareToIgnoreCase`
- See example: SecretWord.java

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

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

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

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.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

The Conditional Operator

- The format of the operators is:

```
BooleanExpression ? Value1 : Value2
```

- This forms a conditional expression.
- If *BooleanExpression* is true, the value of the conditional expression is *Value1*.
- If *BooleanExpression* is false, the value of the conditional expression is *Value2*.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

The Conditional Operator

- Example:

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

- This line is functionally equivalent to:

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

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

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

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

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 an *integer* type or *character* type variable and make decisions based on the value.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

The `switch` Statement

- The `switch` statement takes the form:

```
switch (SwitchExpression)
{
  case CaseExpression:
    // place one or more statements here
    break;
  case CaseExpression:
    // place one or more statements here
    break;

    // case statements may be repeated
    //as many times as necessary
  default:
    // place one or more statements here
}
```

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

The `switch` Statement

```
switch (SwitchExpression)
{
  ...
}
```

- The `switch` statement will evaluate the *SwitchExpression*, which can be a `byte`, `short`, `int`, `long`, or `char`. If you are using Java 7, the *SwitchExpression* can also be a string.
- If there is an associated `case` statement that matches that value, program execution will be transferred to that `case` statement.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

The `switch` Statement

- Each `case` statement will have a corresponding *CaseExpression* that must be unique.

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

- If the *SwitchExpression* matches the *CaseExpression*, the Java statements between the colon and the `break` statement will be executed.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

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

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

The `printf` Method

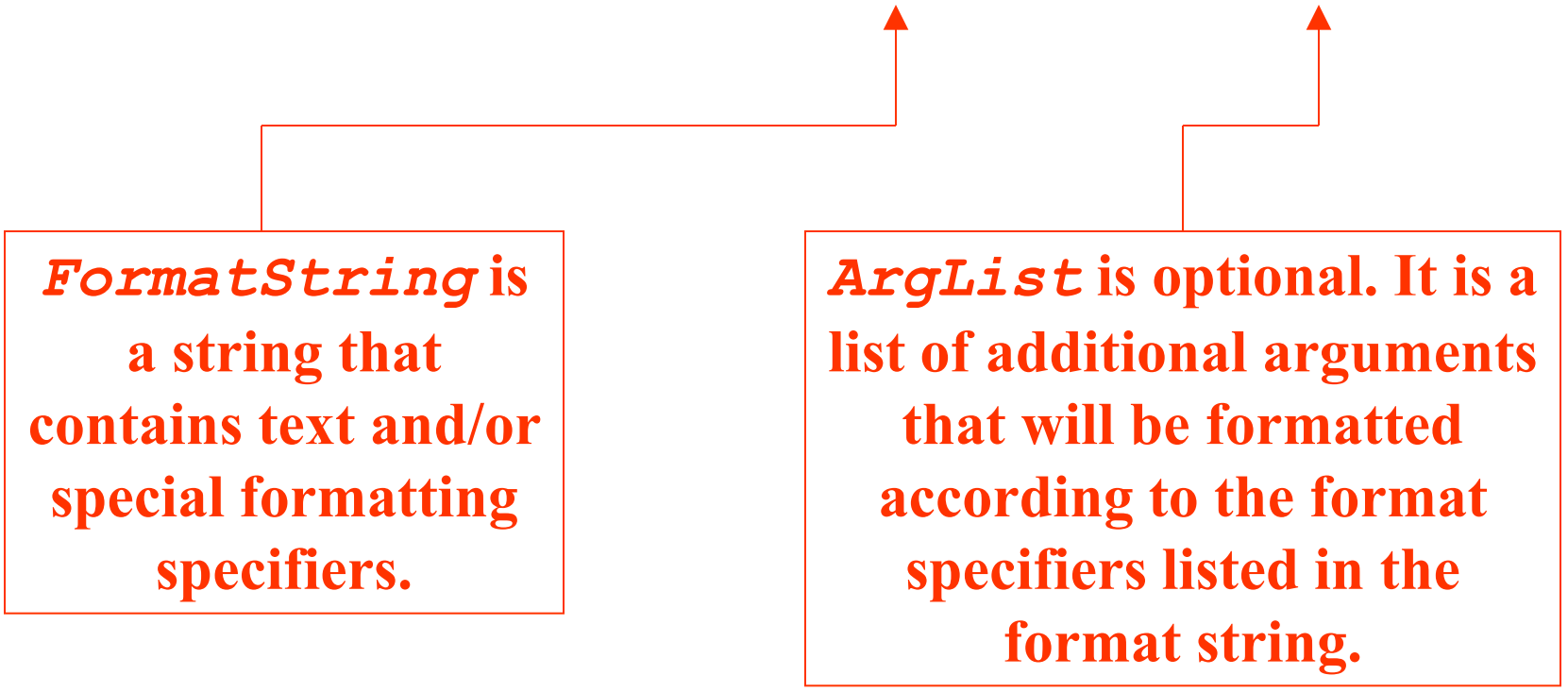
- You can use the `System.out.printf` method to perform formatted console output.
- The general format of the method is:

```
System.out.printf(FormatString, ArgList);
```

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

The `printf` Method

```
System.out.printf(FormatString, ArgList);
```



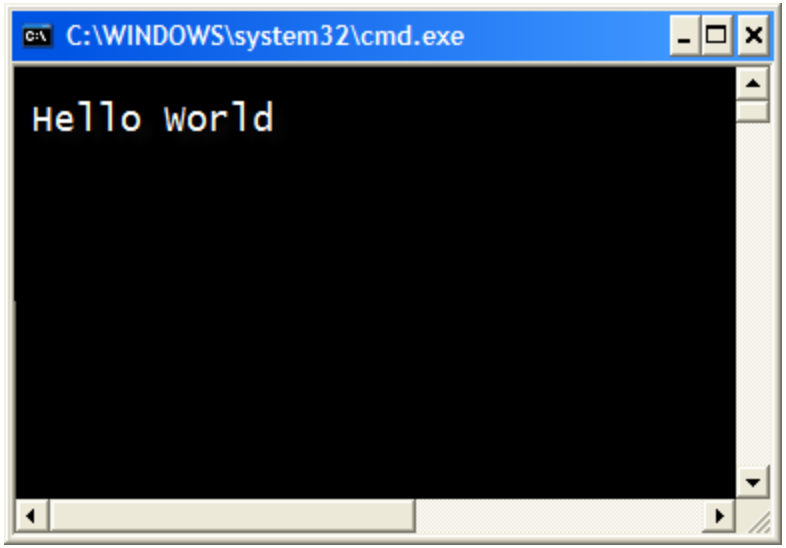
©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

The `printf` Method

- A simple example:

```
System.out.printf("Hello World\n");
```

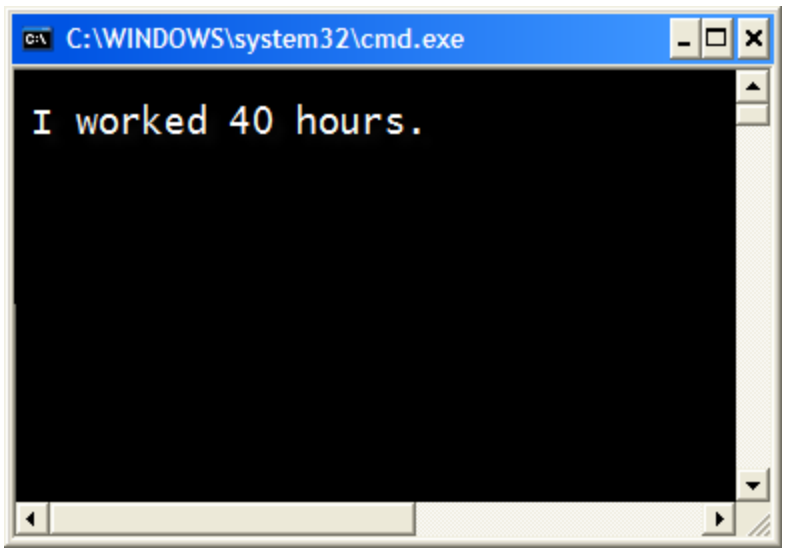


©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

The `printf` Method

- Another example:

```
int hours = 40;
System.out.printf("I worked %d hours.\n", hours);
```

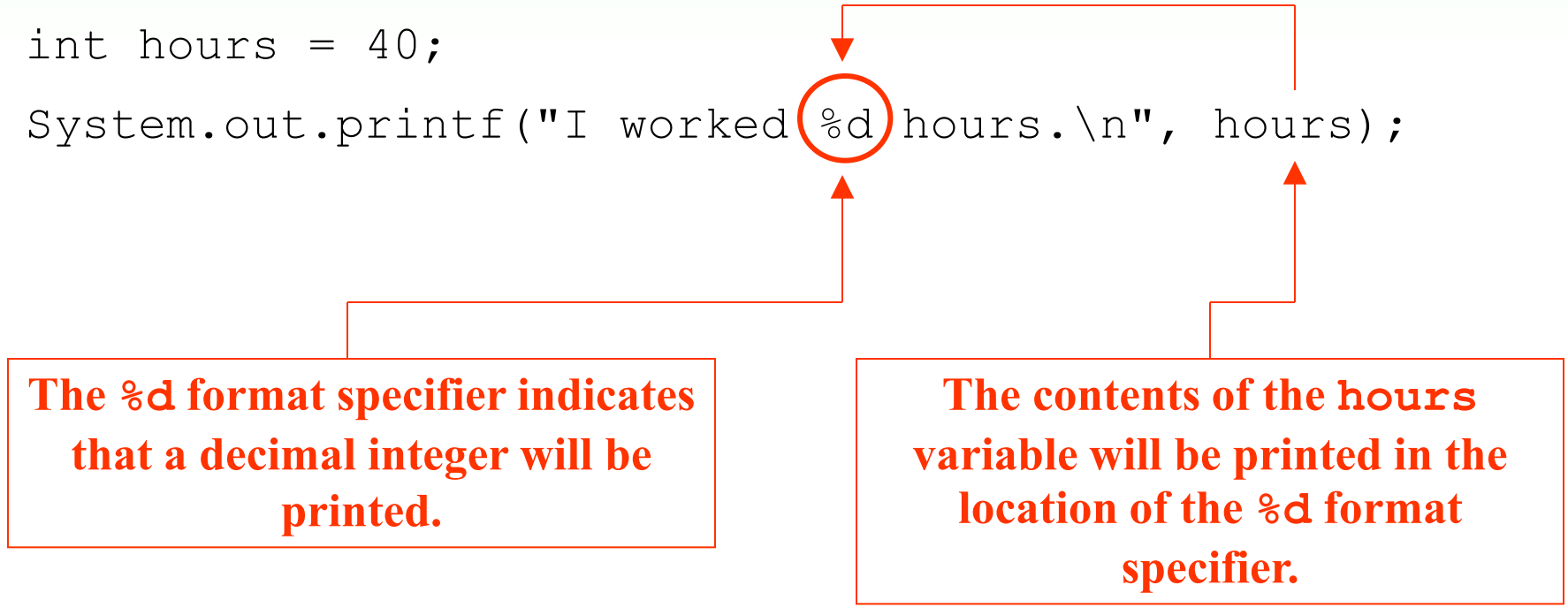


©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

The `printf` Method

```
int hours = 40;
System.out.printf("I worked %d hours.\n", hours);
```

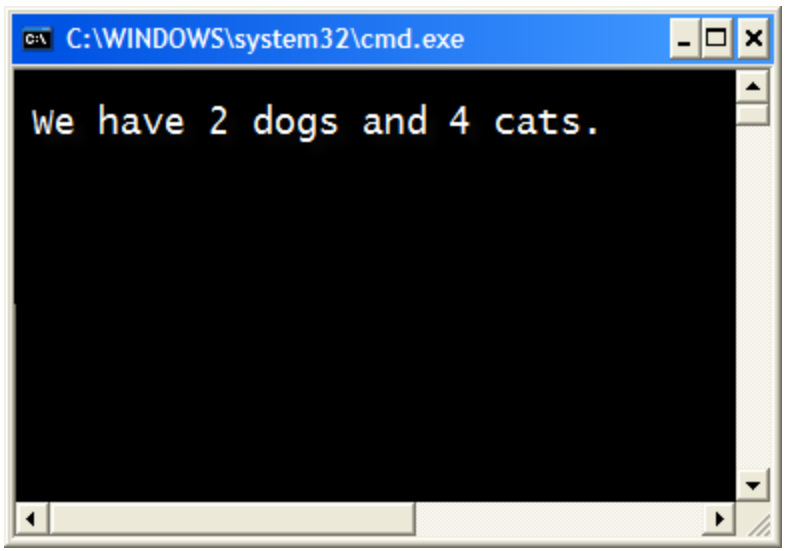


©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

The `printf` Method

- Another example:

```
int dogs = 2, cats = 4;
System.out.printf("We have %d dogs and %d cats.\n",
                  dogs, cats);
```



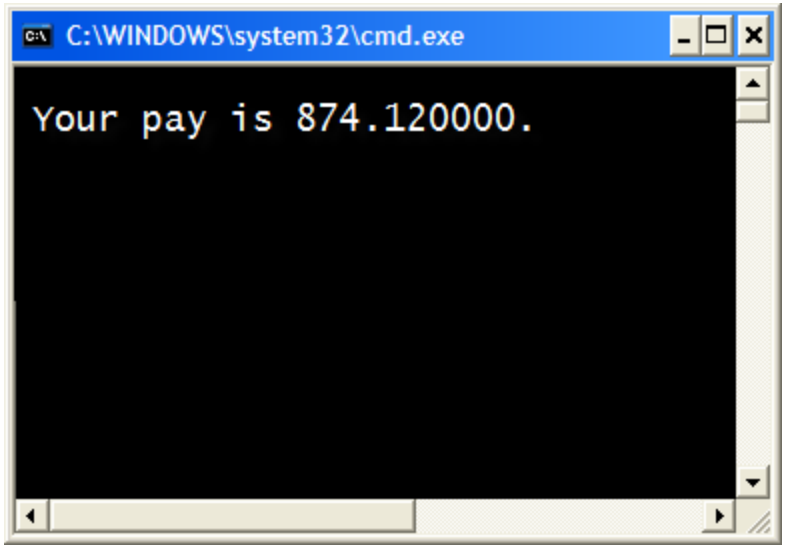
©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

The `printf` Method

- Another example:

```
double grossPay = 874.12;

System.out.printf("Your pay is %f.\n", grossPay);
```



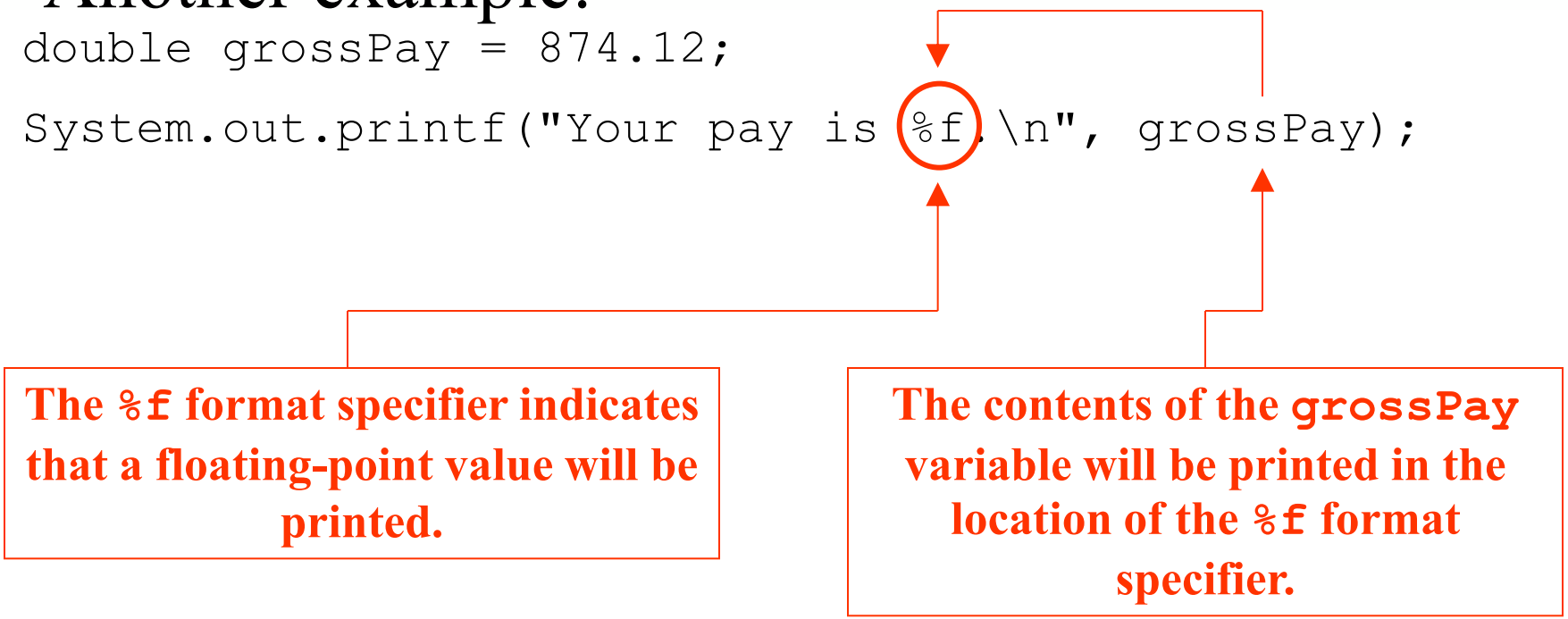
©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

The `printf` Method

- Another example:

```
double grossPay = 874.12;

System.out.printf("Your pay is %f.\n", grossPay);
```



©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

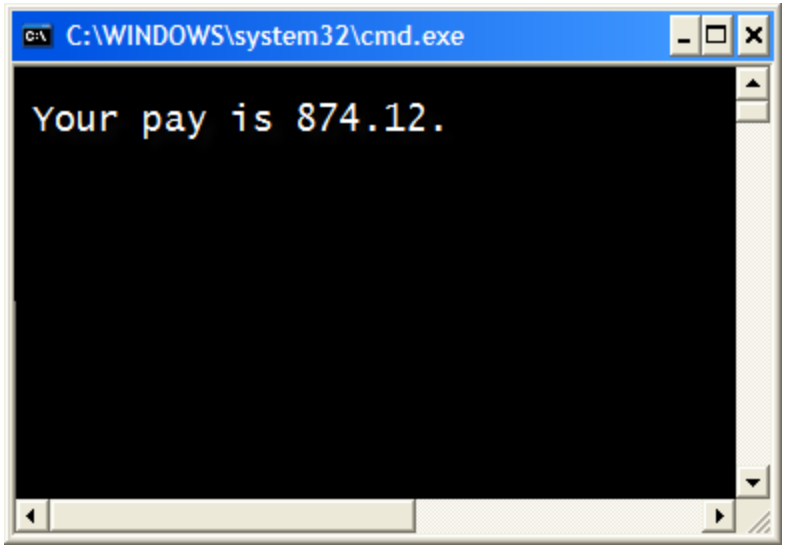
Troy University Resources

The `printf` Method

- Another example:

```
double grossPay = 874.12;

System.out.printf("Your pay is %.2f.\n", grossPay);
```



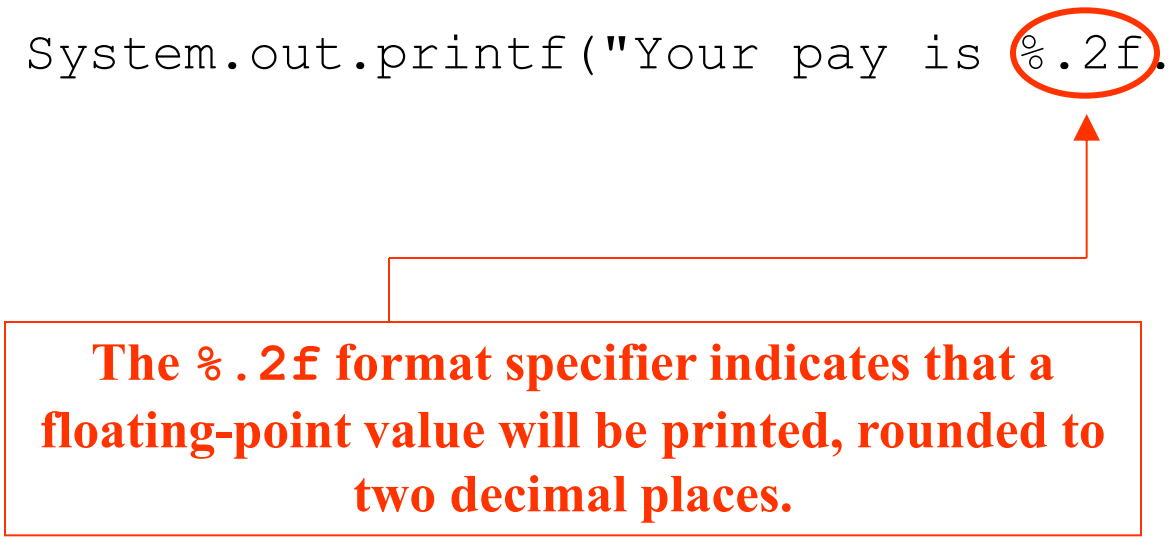
©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

The `printf` Method

- Another example:

```
double grossPay = 874.12;

System.out.printf("Your pay is %.2f.\n", grossPay);
```



©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

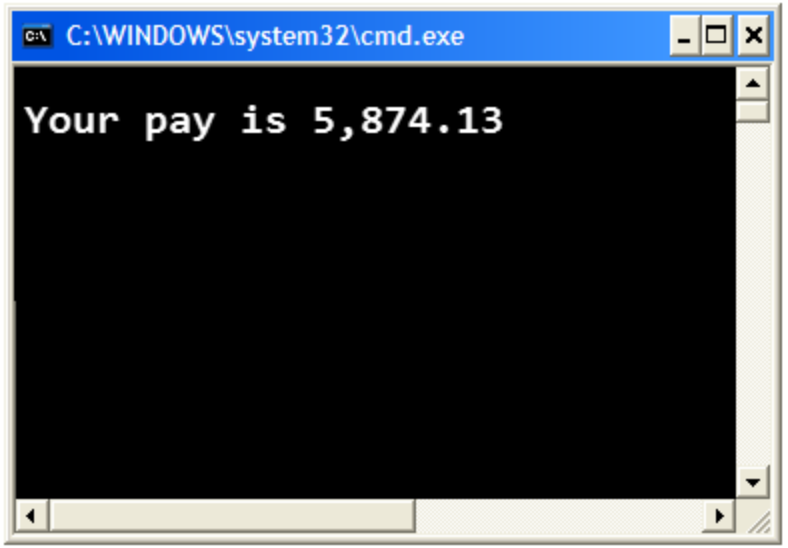
The `printf` Method

- Another example:

```
double grossPay = 5874.127;

System.out.printf("Your pay is %,.2f\n", grossPay);
```

The `%,.2f` format specifier indicates that a floating-point value will be printed with comma separators, rounded to two decimal places.



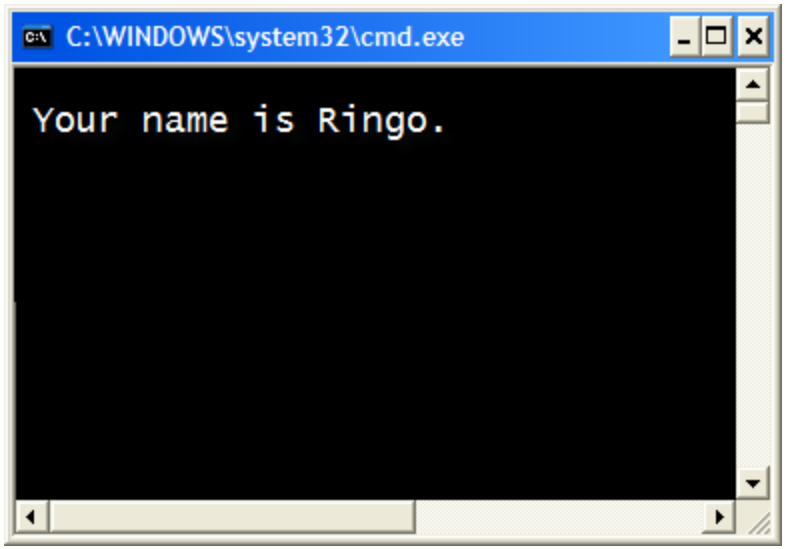
©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

The `printf` Method

- Another example:

```
String name = "Ringo";

System.out.printf("Your name is %s.\n", name);
```



The `%s` format specifier indicates that a string will be printed.

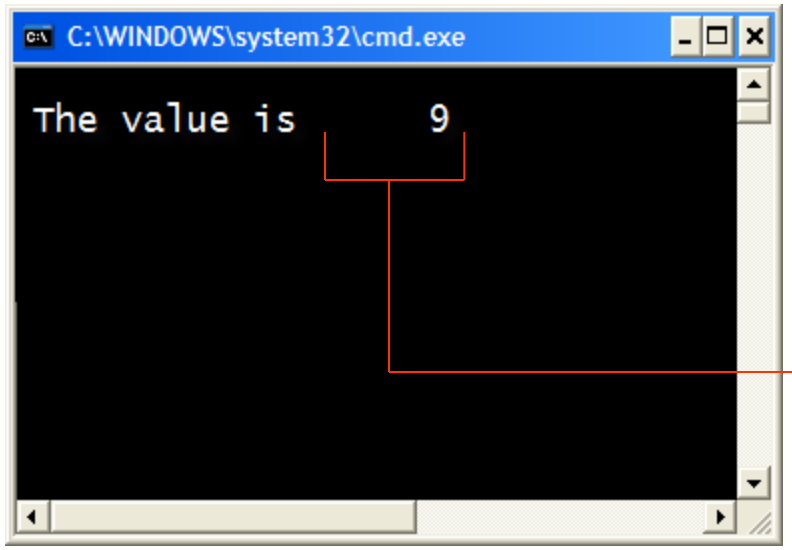
©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

The `printf` Method

- Specifying a field width:

```
int number = 9;
System.out.printf("The value is %6d\n", number);
```



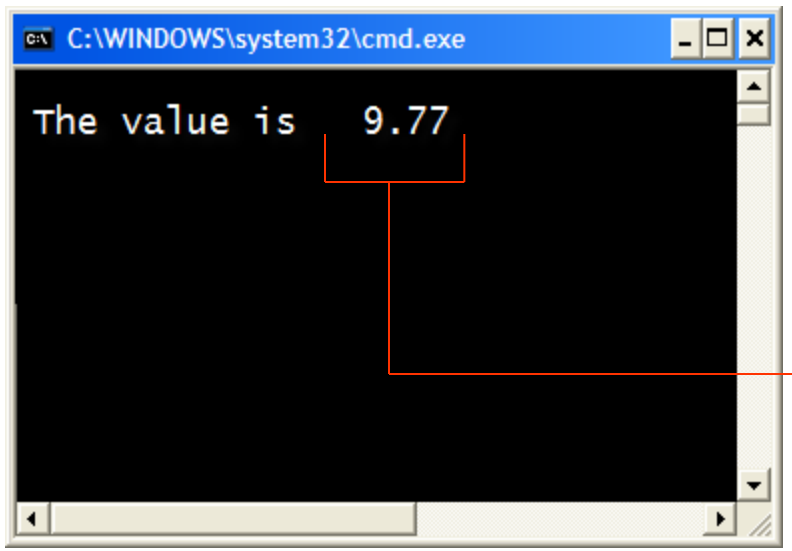
The %6d format specifier indicates the integer will appear in a field that is 6 spaces wide.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

The `printf` Method

- Another example:

```
double number = 9.76891;
System.out.printf("The value is %6.2f\n", number);
```



The `%6.2f` format specifier indicates the number will appear in a field that is 6 spaces wide, and be rounded to 2 decimal places.

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

The `printf` Method

- See examples:
  - Columns.java
  - CurrencyFormat.java

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

The `DecimalFormat` Class

- When printing out `double` and `float` values, the full fractional value will be printed.
- The `DecimalFormat` class can be used to format these values.
- In order to use the `DecimalFormat` class, the following `import` statement must be used at the top of the program:

```
import java.text.DecimalFormat;
```

- See examples:

  Format1.java, Format2.java, Format3.java, Format4.java

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

Exercise for week 3

- Programming challenges
  - 3.7, 3.13, 3.16
  - 4.9, 4.12

©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.