Chapter 2:
Java Fundamentals

Starting Out with Java:
From Control Structures through Objects

Fifth Edition

by Tony Gaddis

PEARSON ALWAYS LEARNING

Troy University Resources

Chapter Topics

Chapter 2 discusses the following main topics:

- The Parts of a Java Program
- The `print` and `println` Methods, and the Java API
- Variables and Literals
- Primitive Data Types
- Arithmetic Operators
- Combined Assignment Operators

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

Chapter Topics (2)

- Creating named constants with final
- The `String` class
- Scope
- Comments
- Programming style
- Using the `Scanner` class for input
- Dialog boxes

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

Troy University Resources

Parts of a Java Program

- A Java source code file contains one or more Java classes.
- If more than one class is in a source code file, only one of them may be public.
- The public class and the filename of the source code file must match.

  ex: A class named *Simple* must be in a file named *Simple.java*
- Each Java class can be separated into parts.

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

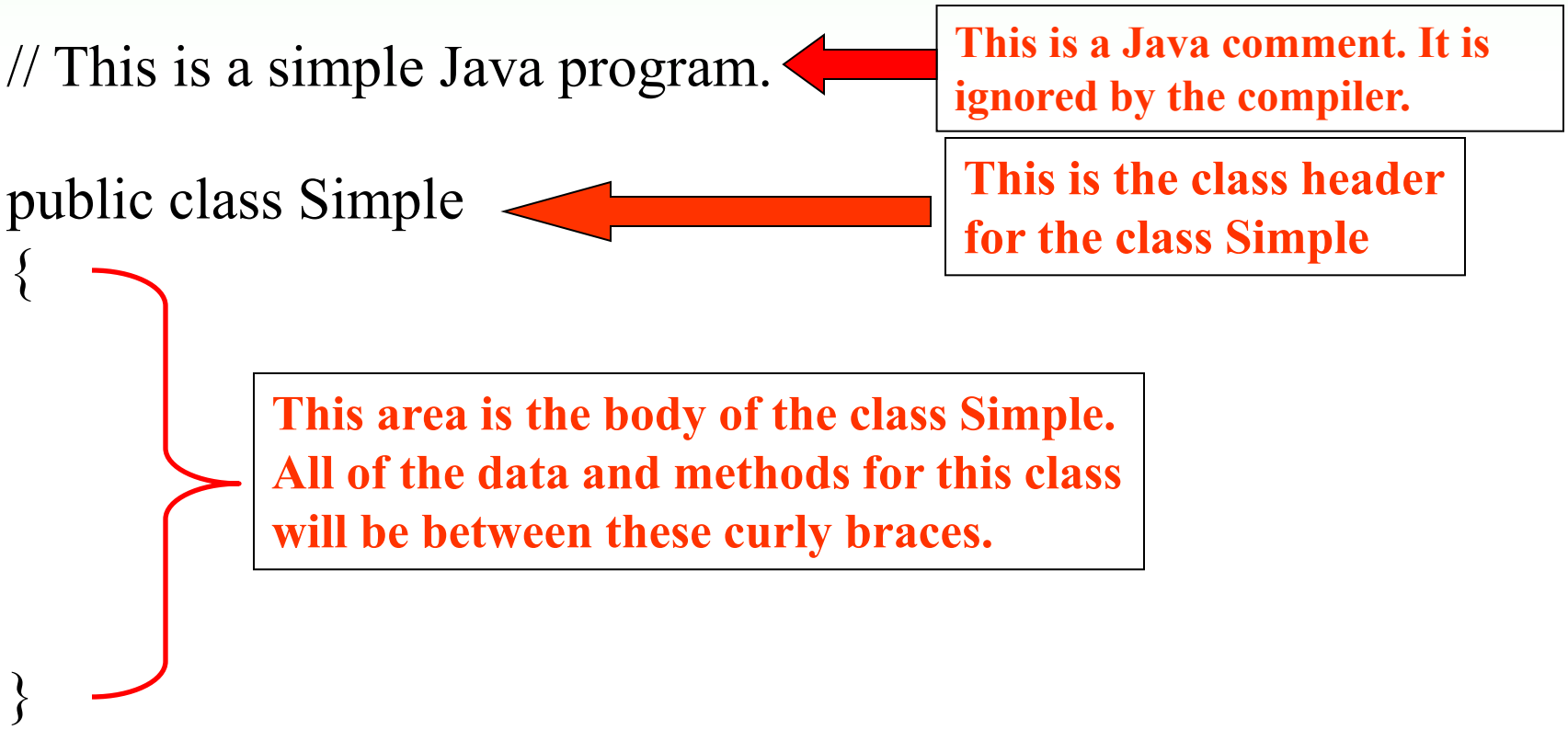
Parts of a Java Program

- See example: Simple.java
- To compile the example:
  - **`javac Simple.java`**
    - Notice the `.java` file extension is needed.
    - This will result in a file named *Simple.class* being created.
- To run the example:
  - **`java Simple`**
    - Notice there is no file extension here.
    - The *java* command assumes the extension is `.class`.

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

Troy University Resources

Analyzing The Example



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

Analyzing The Example

```
// This is a simple Java program.

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

      }
}
```

This is the method header for the main method. The main method is where a Java application begins.

This area is the body of the main method. All of the actions to be completed during the main method will be between these curly braces.

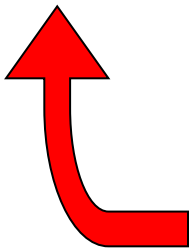
©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

Analyzing The Example

```
// This is a simple Java program.

public class Simple
{
      public static void main(String [] args)
      {
            System.out.println("Programming is great fun!");
      }
}
```



This is the Java Statement that is executed when the program runs.

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

Parts of a Java Program

- Comments
  - The line is ignored by the compiler.
  - The comment in the example is a single-line comment.
- Class Header
  - The class header tells the compiler things about the class such as what other classes can use it (public) and that it is a Java class (class), and the name of that class (Simple).
- Curly Braces
  - When associated with the class header, they define the scope of the class.
  - When associated with a method, they define the scope of the method.

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

Troy University Resources

Parts of a Java Program

- The `main` Method
  - This line must be exactly as shown in the example (except the *args* variable name can be programmer defined).
  - This is the line of code that the *java* command will run first.
  - This method starts the Java program.
  - Every Java *application* must have a `main` method.
- Java Statements
  - When the program runs, the statements within the `main` method will be executed.
  - Can you see what the line in the example will do?

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

Java Statements

- If we look back at the previous example, we can see that there is only one line that ends with a semi-colon.

```
System.out.println("Programming is great fun!");
```

- This is because it is the only Java statement in the program.
- The rest of the code is either a comment or other Java framework code.

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

Troy University Resources

Java Statements

- Comments are ignored by the Java compiler so they need no semi-colons.
- Other Java code elements that do not need semi colons include:
  - class headers
    - Terminated by the code within its curly braces.
  - method headers
    - Terminated by the code within its curly braces.
  - curly braces
    - Part of framework code that needs no semi-colon termination.

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

Short Review

- Java is a case-sensitive language.
- All Java programs must be stored in a file with a .java file extension.
- Comments are ignored by the compiler.
- A .java file may contain many classes but may only have one public class.
- If a .java file has a public class, the class must have the same name as the file.

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

Troy University Resources

Short Review

- Java applications must have a `main` method.
- For every left brace, or opening brace, there must be a corresponding right brace, or closing brace.
- Statements are terminated with semicolons.
  - Comments, class headers, method headers, and braces are not considered Java statements.

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

Special Characters

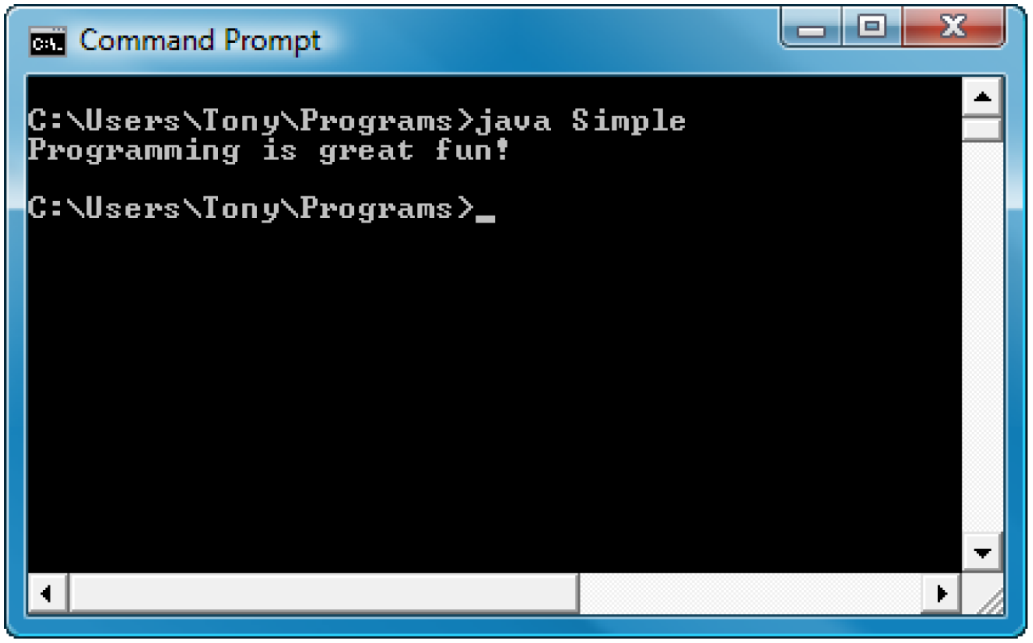
//	**double slash**	**Marks the beginning of a single line comment.**
( )	**open and close parenthesis**	**Used in a method header to mark the *parameter list*.**
{ }	**open and close curly braces**	**Encloses a group of statements, such as the contents of a class or a method.**
“ ”	**quotation marks**	**Encloses a string of characters, such as a message that is to be printed on the screen**
;	**semi-colon**	**Marks the end of a complete programming statement**

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

Troy University Resources

Console Output

- Many of the programs that you will write will run in a console window.



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

Console Output

- The console window that starts a Java application is typically known as the *standard output* device.
- The *standard input* device is typically the keyboard.
- Java sends information to the standard output device by using a Java class stored in the standard Java library.

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

Troy University Resources

Console Output

- Java classes in the standard Java library are accessed using the Java Applications Programming Interface (API).
- The standard Java library is commonly referred to as the *Java API*.

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

Console Output

- The previous example uses the line:

```
System.out.println("Programming is great fun!");
```

- This line uses the `System` class from the standard Java library.
- The `System` class contains methods and objects that perform system level tasks.
- The `out` object, a member of the `System` class, contains the methods `print` and `println`.

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

Troy University Resources

Console Output

- The `print` and `println` methods actually perform the task of sending characters to the output device.
- The line:

```
System.out.println("Programming is great fun!");
```

  is pronounced: System dot out dot println …
- The value inside the parenthesis will be sent to the output device (in this case, a string).

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

Console Output

- The `println` method places a newline character at the end of whatever is being printed out.
- The following lines:

```
System.out.println("This is being printed out");
System.out.println("on two separate lines.");
```

Would be printed out on separate lines since the first statement sends a newline command to the screen.

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

Troy University Resources

Console Output

- The `print` statement works very similarly to the `println` statement.
- However, the `print` statement does not put a newline character at the end of the output.
- The lines:

```
System.out.print("These lines will be");
System.out.print("printed on");
System.out.println("the same line.");
```

Will output:

```
These lines will beprinted onthe same line.
```

Notice the odd spacing? Why are some words run together?

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

Console Output

- For all of the previous examples, we have been printing out strings of characters.
- Later, we will see that much more can be printed.
- There are some special characters that can be put into the output.

```
System.out.print("This line will have a newline at the end.\n");
```

- The `\n` in the string is an escape sequence that represents the newline character.
- Escape sequences allow the programmer to print characters that otherwise would be unprintable.

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

Troy University Resources

Displaying Text with `printf`

- System.out.printf method
  - `f` means "formatted"
  - displays formatted data
- Multiple method arguments are placed in a comma-separated list.
- Java allows large statements to be split over many lines.
  - Cannot split a statement in the middle of an identifier or string.
- Method `printf`'s first argument is a format string
  - May consist of fixed text and format specifiers.
  - Fixed text is output as it would be by `print` or `println`.
  - Each format specifier is a placeholder for a value and specifies the type of data to output.
- Format specifiers begin with a percent sign (%) and are followed by a character that represents the data type.
- Format specifier `%s` is a placeholder for a string.

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

Format specifier

- d : integers ➔ %d
- s: string ➔ %s
- c: character ➔ %c
- f: real number ➔ %s

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

```
1  // Fig. 2.6: Welcome4.java
2  // Displaying multiple lines with method System.out.printf.
3
4  public class Welcome4
5  {
6     // main method begins execution of Java application
7     public static void main( String[] args )
8     {
9        System.out.printf( "%s\n%s\n",
10          "Welcome to", "Java Programming!" );
11    } // end method main
12 } // end class Welcome4
```

Each %s is a placeholder for a String that comes later in the argument list

Statements can be split over multiple lines.

```
Welcome to
Java Programming!
```

Fig. 2.6 | Displaying multiple lines with method `System.out.printf`.

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

Java Escape Sequences

`\n`	newline	Advances the cursor to the next line for subsequent printing
`\t`	tab	Causes the cursor to skip over to the next tab stop
`\b`	backspace	Causes the cursor to back up, or move left, one position
`\r`	carriage return	Causes the cursor to go to the beginning of the current line, not the next line
`\\`	backslash	Causes a backslash to be printed
`\'`	single quote	Causes a single quotation mark to be printed
`\"`	double quote	Causes a double quotation mark to be printed

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

Troy University Resources

Java Escape Sequences

- Even though the escape sequences are comprised of two characters, they are treated by the compiler as a single character.

```
System.out.print("These are our top sellers:\n");
System.out.print("\tComputer games\n\tCoffee\n ");
System.out.println("\tAspirin");
```

Would result in the following output:

```
These are our top seller:
        Computer games
        Coffee
        Asprin
```

- With these escape sequences, complex text output can be achieved.

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

Variables and Literals

- A variable is a named storage location in the computer's memory.
- A literal is a value that is written into the code of a program.
- Programmers determine the number and type of variables a program will need.
- See example: Variable.java

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

Troy University Resources

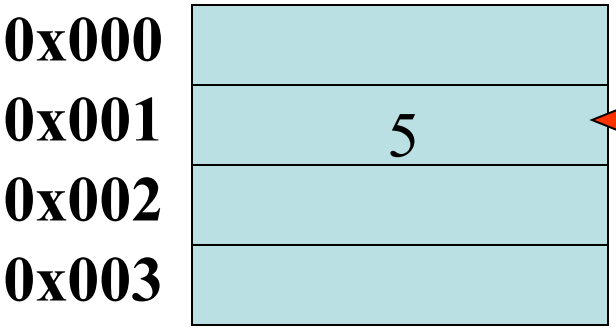
Variables and Literals

This line is called a *variable declaration*.

```
int value;
```

The following line is known as an assignment statement.

```
value = 5;
```



The value 5 is stored in memory.

This is a string *literal*. It will be printed as is.

```
System.out.print("The value is ");
System.out.println(value);
```

The integer 5 will be printed out here. Notice no quote marks?

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

The + Operator

- The + operator can be used in two ways.
  - as a concatenation operator
  - as an addition operator
- If either side of the + operator is a string, the result will be a string.

```
System.out.println("Hello " + "World");
System.out.println("The value is: " + 5);
System.out.println("The value is: " + value);
System.out.println("The value is: " + '/n' + 5);
```

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

Troy University Resources

String Concatenation

- Java commands that have string literals must be treated with care.
- A string literal value cannot span lines in a Java source code file.

```
System.out.println("This line is too long and now it
has spanned more than one line, which will cause a
syntax error to be generated by the compiler. ");
```

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

String Concatenation

- The String concatenation operator can be used to fix this problem.

```
System.out.println("These lines are " +
                   "are now ok and will not " +
                   "cause the error as before.");
```

- String concatenation can join various data types.

```
System.out.println("We can join a string to " +
                   "a number like this: " + 5);
```

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

Troy University Resources

String Concatenation

- The Concatenation operator can be used to format complex String objects.

```
System.out.println("The following will be printed " +
                   "in a tabbed format: " +
                   \n\tFirst = " + 5 * 6 + ", " +
                   "\n\tSecond = " (6 + 4) + "," +
                   "\n\tThird = " + 16.7 + ".");
```

- Notice that if an addition operation is also needed, it must be put in parenthesis.

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

Identifiers

- Identifiers are programmer-defined names for:
  - classes
  - variables
  - methods
- Identifiers may not be any of the Java reserved keywords.

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

Troy University Resources

Identifiers

- Identifiers must follow certain rules:
  - An identifier may only contain:
    - letters a–z or A–Z,
    - the digits 0–9,
    - underscores (_), or
    - the dollar sign ($)
  - The first character may not be a digit.
  - Identifiers are case sensitive.
    - `itemsOrdered` is not the same as `itemsordered`.
  - Identifiers cannot include spaces.

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

Java Reserved Keywords

abstract	double	instanceof	static
assert	else	int	strictfp
boolean	enum	interface	super
break	extends	long	switch
byte	false	native	synchronized
case	for	new	this
catch	final	null	throw
char	finally	package	throws
class	float	private	transient
const	goto	protected	true
continue	if	public	try
default	implements	return	void
do	import	short	volatile
			while

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

Troy University Resources

Variable Names

- Variable names should be descriptive.
- Descriptive names allow the code to be more readable; therefore, the code is more maintainable.
- Which of the following is more descriptive?

```
double tr = 0.0725;
double salesTaxRate = 0.0725;
```

- Java programs should be *self-documenting*.

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

Java Naming Conventions

- Variable names should begin with a lower case letter and then switch to title case thereafter:

  Ex: `int caTaxRate`

- Class names should be all title case.

  Ex: `public class BigLittle`

- More Java naming conventions can be found at:

  http://java.sun.com/docs/codeconv/html/CodeConventions.doc8.html

- A general rule of thumb about naming variables and classes are that, with some exceptions, their names tend to be nouns or noun phrases.

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

Primitive Data Types

- Primitive data types are built into the Java language and are not derived from classes.
- There are 8 Java primitive data types.
  - `byte`
  - `short`
  - `int`
  - `long`
  - `float`
  - `double`
  - `boolean`
  - `char`

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

Numeric Data Types

byte	1 byte	Integers in the range -128 to +127
short	2 bytes	Integers in the range of -32,768 to +32,767
int	4 bytes	Integers in the range of -2,147,483,648 to +2,147,483,647
long	8 bytes	Integers in the range of -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
float	4 bytes	Floating-point numbers in the range of ±3.410-38 to ±3.41038, with 7 digits of accuracy
double	8 bytes	Floating-point numbers in the range of ±1.710-308 to ±1.710308, with 15 digits of accuracy

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

Troy University Resources

Variable Declarations

- Variable Declarations take the following form:
  - *DataType VariableName;*
    - `byte inches;`
    - `short month;`
    - `int speed;`
    - `long timeStamp;`
    - `float salesCommission;`
    - `double distance;`

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

Integer Data Types

- `byte`, `short`, `int`, and `long` are all integer data types.
- They can hold whole numbers such as 5, 10, 23, 89, etc.
- Integer data types cannot hold numbers that have a decimal point in them.
- Integers embedded into Java source code are called *integer literals*.
- See Example: IntegerVariables.java

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

Troy University Resources

Floating Point Data Types

- Data types that allow fractional values are called *floating-point* numbers.
  - 1.7 and -45.316 are floating-point numbers.
- In Java there are two data types that can represent floating-point numbers.
  - `float` - also called *single precision* (7 decimal points).
  - `double` - also called *double precision* (15 decimal points).

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

Floating Point Literals

- When floating point numbers are embedded into Java source code they are called *floating point literals*.
- The default type for floating point literals is `double`.
  - 29.75, 1.76, and 31.51 are `double` data types.
- Java is a *strongly-typed* language.
- See example: Sale.java

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

Troy University Resources

Floating Point Literals

- A `double` value is not compatible with a `float` variable because of its size and precision.
  - `float number;`
  - `number = 23.5; // Error!`
- A `double` can be forced into a `float` by appending the letter F or f to the literal.
  - `float number;`
  - `number = 23.5F; // This will work.`

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

Floating Point Literals

- Literals cannot contain embedded currency symbols or commas.
  - `grossPay = $1,257.00; // ERROR!`
  - `grossPay = 1257.00;  // Correct.`
- Floating-point literals can be represented in *scientific notation*.
  - 47,281.97 == 4.728197 x 10^4.
- Java uses *E notation* to represent values in scientific notation.
  - 4.728197X10^4 == 4.728197E4.

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

Troy University Resources

Scientific and E Notation

Decimal Notation	Scientific Notation	E Notation
247.91	2.4791×10^{2}	2.4791E2
0.00072	7.2×10^{-4}	7.2E-4
2,900,000	2.9×10^{6}	2.9E6

See example: SunFacts.java

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

The `boolean` Data Type

- The Java `boolean` data type can have two possible values.
  - `true`
  - `false`
- The value of a `boolean` variable may only be copied into a `boolean` variable.

See example: TrueFalse.java

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

Troy University Resources

The `char` Data Type

- The Java `char` data type provides access to single characters.
- `char` literals are enclosed in single quote marks.
  - ‘a’, ‘Z’, ‘\n’, ‘1’
- Don’t confuse `char` literals with string literals.
  - `char` literals are enclosed in single quotes.
  - String literals are enclosed in double quotes.

See example: Letters.java

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

Unicode

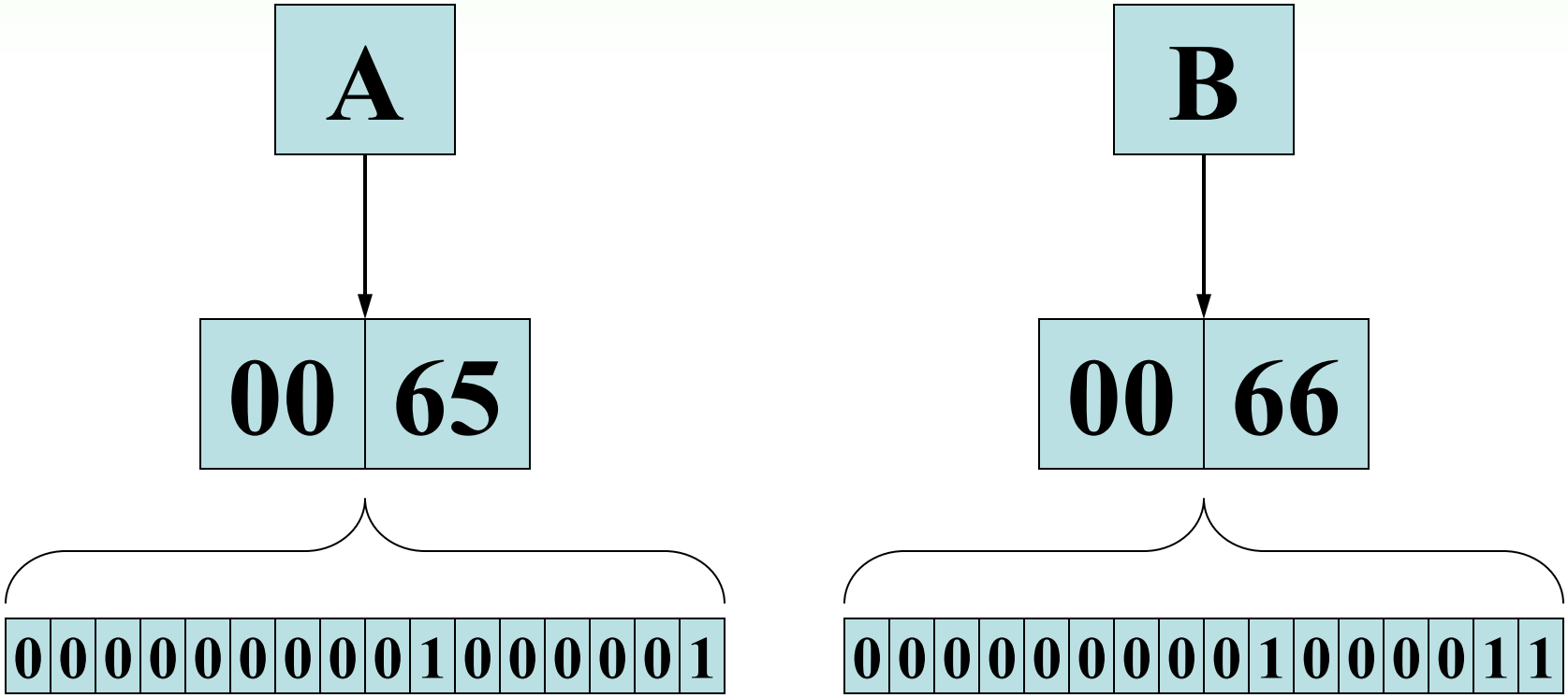
- Internally, characters are stored as numbers.
- Character data in Java is stored as Unicode characters.
- The Unicode character set can consist of 65536 (2^{16}) individual characters.
- This means that each character takes up 2 bytes in memory.
- The first 256 characters in the Unicode character set are compatible with the ASCII* character set.

See example: Letters2.java

*American Standard Code for Information Interchange

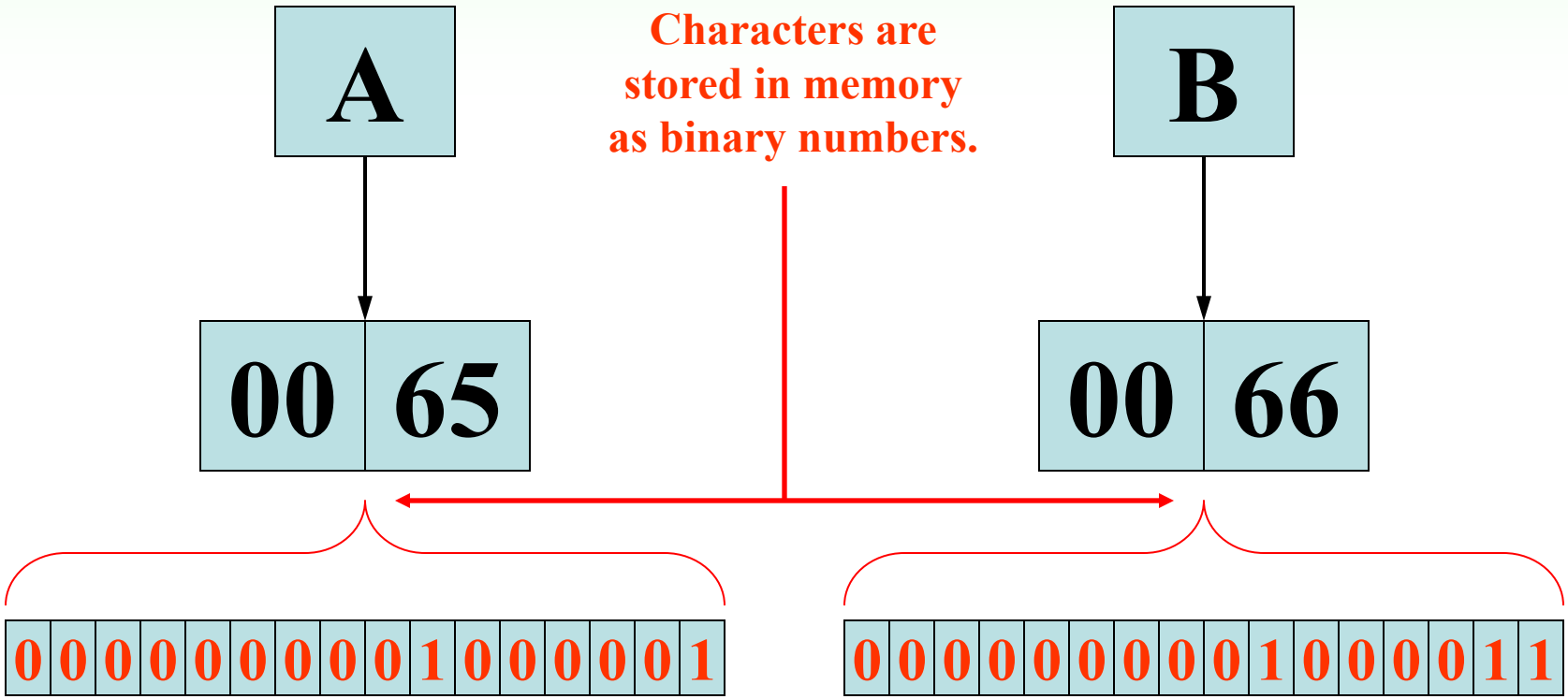
©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Unicode



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

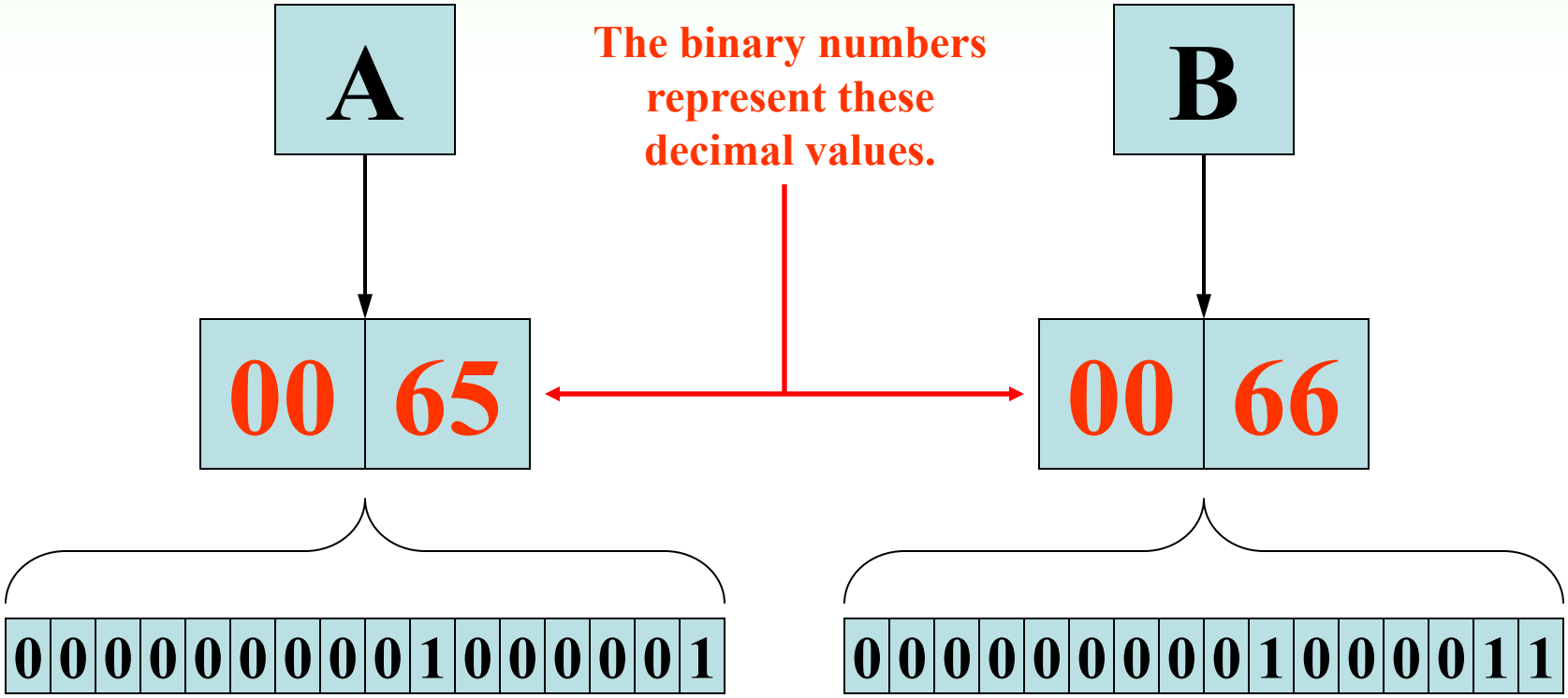
Unicode



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

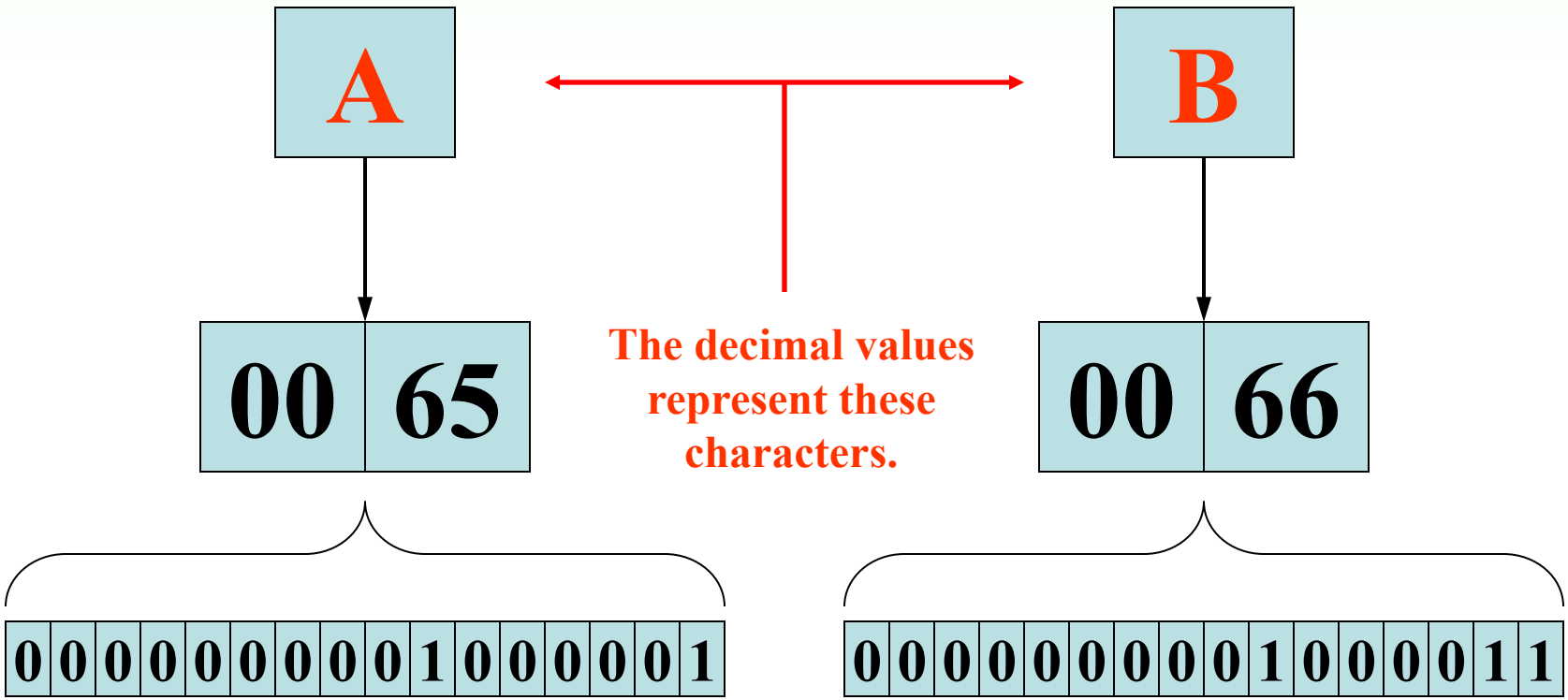
Troy University Resources

Unicode



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

Unicode



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

Troy University Resources

Variable Assignment and Initialization

- In order to store a value in a variable, an *assignment statement* must be used.
- The *assignment operator* is the equal (=) sign.
- The operand on the left side of the assignment operator must be a variable name.
- The operand on the right side must be either a literal or expression that evaluates to a type that is compatible with the type of the variable.

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

Variable Assignment and Initialization

```
// This program shows variable assignment.

public class Initialize
{
  public static void main(String[] args)
  {
    int month, days;

    month = 2;
    days = 28;
    System.out.println("Month " + month + " has " +
                       days + " Days.");
  }
}
```

The variables must be declared before they can be used.

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

Troy University Resources

Variable Assignment and Initialization

```
// This program shows variable assignment.

public class Initialize
{
  public static void main(String[] args)
  {
    int month, days;

    month = 2;
    days = 28;
    System.out.println("Month " + month + " has " +
                       days + " Days.");
  }
}
```

Once declared, they can then receive a value (initialization); however the value must be compatible with the variable's declared type.

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

Variable Assignment and Initialization

```
// This program shows variable assignment.

public class Initialize
{
  public static void main(String[] args)
  {
    int month, days;

    month = 2;
    days = 28;
    System.out.println("Month " + month + " has " +
                       days + " Days.");
  }
}
```

After receiving a value, the variables can then be used in output statements or in other calculations.

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

Troy University Resources

Variable Assignment and Initialization

```
// This program shows variable initialization.

public class Initialize
{
  public static void main(String[] args)
  {
    int month = 2, days = 28;
    System.out.println("Month " + month + " has " +
                       days + " Days.");
  }
}
```

Local variables can be declared and initialized on the same line.

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

Variable Assignment and Initialization

- Variables can only hold one value at a time.
- Local variables do not receive a default value.
- Local variables must have a valid type in order to be used.

```
public static void main(String [] args)
 {
   int month, days; //No value given…
  System.out.println("Month " + month + " has " +
                        days + " Days.");
 }
```

Trying to use uninitialized variables will generate a Syntax Error when the code is compiled.

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

Arithmetic Operators

- Java has five (5) arithmetic operators.

Operator	Meaning	Type	Example
+	Addition	Binary	`total = cost + tax;`
-	Subtraction	Binary	`cost = total - tax;`
*	Multiplication	Binary	`tax = cost * rate;`
/	Division	Binary	`salePrice = original / 2;`
%	Modulus	Binary	`remainder = value % 5;`

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

Arithmetic Operators

- The operators are called binary operators because they must have two operands.
- Each operator must have a left and right operator.

See example: Wages.java

- The arithmetic operators work as one would expect.
- It is an error to try to divide any number by zero.
- When working with two integer operands, the division operator requires special attention.

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

Troy University Resources

Integer Division

- Division can be tricky.

  In a Java program, what is the value of 1/2?
- You might think the answer is 0.5…
- But, that's wrong.
- The answer is simply 0.
- Integer division will truncate any decimal remainder.

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

Operator Precedence

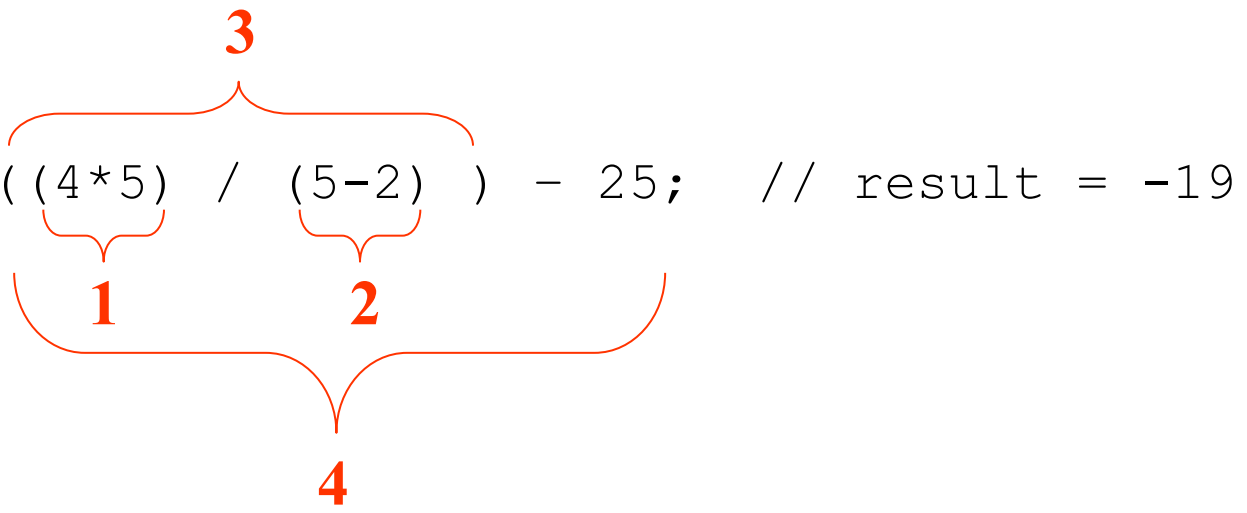
- Mathematical expressions can be very complex.
- There is a set order in which arithmetic operations will be carried out.

	Operator	Associativity	Example	Result
Higher Priority	– (unary negation)	Right to left	`x = -4 + 3;`	-1
	`* / %`	Left to right	`x = -4 + 4 % 3 * 13 + 2;`	11
Lower Priority	`+ –`	Left to right	`x = 6 + 3 – 4 + 6 * 3;`	23

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

Grouping with Parenthesis

- When parenthesis are used in an expression, the inner most parenthesis are processed first.
- If two sets of parenthesis are at the same level, they are processed left to right.



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

Combined Assignment Operators

- Java has some combined assignment operators.
- These operators allow the programmer to perform an arithmetic operation and assignment with a single operator.
- Although not required, these operators are popular since they shorten simple equations.

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

Troy University Resources

Combined Assignment Operators

Operator	Example	Equivalent	Value of variable after operation
+=	`x += 5;`	`x = x + 5;`	The old value of **x** plus 5.
-=	`y -= 2;`	`y = y - 2;`	The old value of **y** minus 2
***=**	`z *= 10;`	`z = z * 10;`	The old value of **z** times 10
/=	`a /= b;`	`a = a / b;`	The old value of **a** divided by **b**.
%=	`c %= 3;`	`c = c % 3;`	The remainder of the division of the old value of **c** divided by 3.

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

Creating Constants

- Many programs have data that does not need to be changed.
- Littering programs with literal values can make the program hard do read and maintain.
- Replacing literal values with constants remedies this problem.
- Constants allow the programmer to use a name rather than a value throughout the program.
- Constants also give a singular point for changing those values when needed.

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

Troy University Resources

Creating Constants

- Constants keep the program organized and easier to maintain.
- Constants are identifiers that can hold only a single value.
- Constants are declared using the keyword `final`.
- Constants need not be initialized when declared; however, they must be initialized before they are used or a compiler error will be generated.

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

Creating Constants

- Once initialized with a value, constants cannot be changed programmatically.
- By convention, constants are all upper case and words are separated by the underscore character.

```
final int CAL_SALES_TAX = 0.725;
```

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

Troy University Resources

The `String` Class

- Java has no primitive data type that holds a series of characters.
- The `String` class from the Java standard library is used for this purpose.
- In order to be useful, the a variable must be created to reference a `String` object.

```
String number;
```

- Notice the `S` in `String` is upper case.
- By convention, class names should always begin with an upper case character.

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

Primitive vs. Reference Variables

- Primitive variables actually contain the value that they have been assigned.

```
number = 25;
```

- The value 25 will be stored in the memory location associated with the variable `number`.
- Objects are not stored in variables, however. Objects are *referenced* by variables.

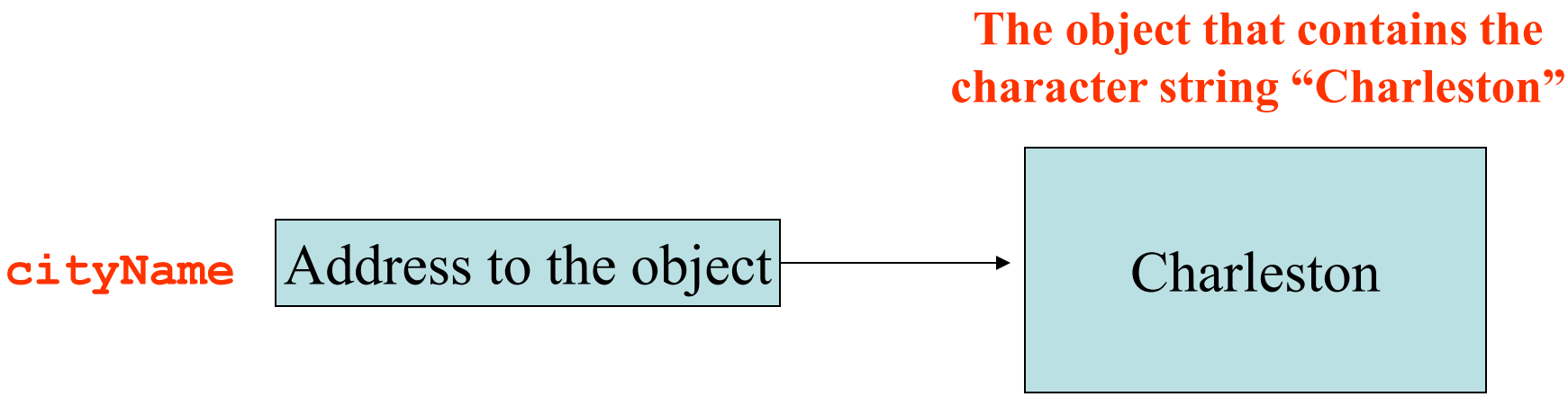
©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.

Troy University Resources

Primitive vs. Reference Variables

- When a variable references an object, it contains the memory address of the object's location.
- Then it is said that the variable *references* the object.

```
String cityName = "Charleston";
```



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

`String` Objects

- A variable can be assigned a String literal.

```
String value = "Hello";
```

- `Strings` are the only objects that can be created in this way.
- A variable can be created using the *new* keyword.

```
String value = new String("Hello");
```

- This is the method that all other objects must use when they are created.

See example: StringDemo.java

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

Troy University Resources

The `String` Methods

- Since `String` is a class, objects that are instances of it have methods.
- One of those methods is the `length` method.

```
stringSize = value.length();
```

- This statement runs the `length` method on the object pointed to by the `value` variable.

See example: StringLength.java

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

`String` Methods

- The `String` class contains many methods that help with the manipulation of `String` objects.
- `String` objects are *immutable*, meaning that they cannot be changed.
- Many of the methods of a `String` object can create new versions of the object.

See example: StringMethods.java

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

Troy University Resources

Scope

- *Scope* refers to the part of a program that has access to a variable's contents.
- Variables declared inside a method (like the main method) are called *local variables*.
- Local variables' scope begins at the declaration of the variable and ends at the end of the method in which it was declared.

See example: Scope.java (This program contains an intentional error.)

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

Commenting Code

- Java provides three methods for commenting code.

Comment Style	Description
`//`	Single line comment. Anything after the // on the line will be ignored by the compiler.
`/* … */`	Block comment. Everything beginning with /* and ending with the first */ will be ignored by the compiler. This comment type cannot be nested.
`/** … */`	Javadoc comment. This is a special version of the previous block comment that allows comments to be documented by the javadoc utility program. Everything beginning with the /** and ending with the first */ will be ignored by the compiler. This comment type cannot be nested.

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

Troy University Resources

Commenting Code

- Javadoc comments can be built into HTML documentation.
- See example: Comment3.java
- To create the documentation:
  - Run the `javadoc` program with the source file as an argument
  - Ex: **`javadoc Comment3.java`**
- The `javadoc` program will create `index.html` and several other documentation files in the same directory as the input file.

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

Commenting Code

- Example index.html:



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

Example

```
/**
* <h1>Hello, World!</h1>
* The HelloWorld program implements an application that
* simply displays "Hello World!" to the standard output.
* <p>
* Giving proper comments in your program makes it more
* user friendly and it is assumed as a high quality code.
*
*
* @author  Zara Ali
* @version 1.0
* @since   2014-03-31
*/
public class HelloWorld {

  public static void main(String[] args) {
    // Prints Hello, World! on standard output.
    System.out.println("Hello World!");
  }
}
```

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

JavaDoc Comments

- For the JavaDoc comments to be recognized as such by the javadoc tool, **they must appear immediately before** the class, interface, constructor, method, or data member declarations.
- If you put the JavaDoc comment for the class **before** the import statements, it will be **ignored**.
- The first sentence is a **"summary sentence"**. This should be a short description of the element described by the comment.

Note:

JavaDoc does not provide a format for commenting elements within methods, i.e. the local variables and the computing going on inside the methods. But you still can use the regular comments marks // or /*..*/, to comment this part of your program.

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

JavaDoc Tags

- There are a number of special tags we **can embed** with the JavaDoc comments. These tags start with the "at" symbol **@**.
- JavaDoc tags must start at the **beginning** of a line.

Example:

```
/**
 * Inner class to listen for discovery events.
 *
 * @author Your Name
 * @since 1.0.0
 */
class Listener implements DiscoveryListener
```

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

JavaDoc Tags

@author

Use this tag to create an author entry. You can have multiple @author tags. This tag is **meaningful** only for the **class/interface** JavaDoc comment.

@version

Use this tag to create a version entry. A JavaDoc comment may contain **at most** one @version tag. Version normally refers to the version of the software (such as the JDK) that contains this feature.

@see

Use this tag to add a hyperlinked "See Also" entry to the class.

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

JavaDoc Tags

@param

Use this tag to add a parameter description for a method. This tag contains two parts: the first is the name of the parameter and the second is the description. The description can be more than one line.

@param size the length of the passed array

@return

Use this tag to add a return type description for a method. This tag is meaningful only if the method's return is non-void.

@return true if the array is empty; otherwise return false

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

JavaDoc Tags

Example:

```
/**
 * This class implements the backend interface. It is activatable.
 * This is the class who is used by RMI to assure service-side execution.
 * The compilation process (see comiple_jta.bat) generates stubs
 * from this class, which are transported to the client.
 * Internally these stubs comunicates with the service JTABackend object.
 *
 * @author Your Name
 * @version 1.0.0
 * @see JTABackendProtocol
 *
 */
public static class JTABackend extends Activatable
```

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

JavaDoc Tags

Generated HTML:

public static class **JINITransportAgent.JTABackend**
extends java.rmi.activation.Activatable
implements JINITransportAgent.JTABackendProtocol, java.lang.Runnable

This class implements the backend interface. It is activatable. This is the class who is used by RMI to assure service-side execution. The compilation process (see comiple_jta.bat) generates stubs from this class, which are transported to the client. Internally these stubs comunicates with the service JTABackend object.

Version:
1.0.0
Author:
Your Name
See Also:
`JTABackendProtocol`, Serialized Form

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

Programming Style

- Although Java has a strict syntax, whitespace characters are ignored by the compiler.
- The Java whitespace characters are:
  - space
  - tab
  - newline
  - carriage return
  - form feed

See example: Compact.java

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

Generating JavaDoc Documentation

- After adding the JavaDoc comments to the source files, we use the **javadoc** command to generate the documentation.
- We run the **javadoc** as we run **javac** or **java** tools.
- After the **javadoc** command, we provide the parameters, when the mandatory ones are the either the package's name or the source file (s) name.

Example:

In order to enforce JavaDoc to generate documentation for the complete **GIPSY** package, we write:

javadoc gipsy

In order to enforce JavaDoc to generate documentation for the **JINITransportAgent.java** file, to include the author and version tag and to include all the classes, attributes and methods we write:

javadoc -private -version -author JINITransportAgent.java

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

Indentation

- Programs should use proper indentation.
- Each block of code should be indented a few spaces from its surrounding block.
- Two to four spaces are sufficient.
- Tab characters should be avoided.
  - Tabs can vary in size between applications and devices.
  - Most programming text editors allow the user to replace the tab with spaces.

See example: Readable.java

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

Troy University Resources

The `Scanner` Class

- To read input from the keyboard we can use the `Scanner` class.
- The `Scanner` class is defined in `java.util`, so we will use the following statement at the top of our programs:

```
import java.util.Scanner;
```

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

The `Scanner` Class

- Scanner objects work with `System.in`
- To create a `Scanner` object:

```
Scanner keyboard = new Scanner (System.in);
```

- `Scanner` class methods are listed in Table 2-18 in the text.
- See example: Payroll.java

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

Troy University Resources

Reading Input from the Console

1. Create a Scanner object

```
Scanner input = new Scanner(System.in);
```

2. Use the methods next(), nextByte(), nextShort(), nextInt(), nextLong(), nextFloat(), nextDouble(), or nextBoolean() to obtain to a string, byte, short, int, long, float, double, or boolean value. For example,

```
System.out.print("Enter a double value: ");
Scanner input = new Scanner(System.in);
double d = input.nextDouble();
```



ComputeAreaWithConsoleInput



ComputeAverage

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

Another `Scanner` example

```
import java.util.*;   // so that I can use Scanner

public class ScannerSum {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("Please type three numbers: ");
        int num1 = console.nextInt();
        int num2 = console.nextInt();
        int num3 = console.nextInt();

        int sum = num1 + num2 + num3;
        System.out.println("The sum is " + sum);
    }
}
```

- Output (user input underlined):

```
Please type three numbers: 8 6 13
The sum is 27
```

The `Scanner` can read multiple values from one line.

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

Dialog Boxes

- A *dialog box* is a small graphical window that displays a message to the user or requests input.
- A variety of dialog boxes can be displayed using the `JOptionPane` class.
- Two of the dialog boxes are:
  - Message Dialog - a dialog box that displays a message.
  - Input Dialog - a dialog box that prompts the user for input.

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

The `JOptionPane` Class

- The `JOptionPane` class is not automatically available to your Java programs.
- The following statement must be before the program's class header:

```
import javax.swing.JOptionPane;
```

- This statement tells the compiler where to find the `JOptionPane` class.

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

Troy University Resources

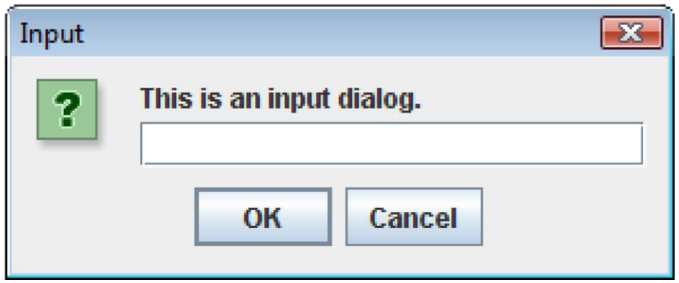
The `JOptionPane` Class

The `JOptionPane` class provides methods to display each type of dialog box.

Message dialog



Input dialog



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

Message Dialogs

- `JOptionPane.showMessageDialog` method is used to display a message dialog.

```
JOptionPane.showMessageDialog(null, "Hello World");
```

- The first argument will be discussed in Chapter 7.
- The second argument is the message that is to be displayed.



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

Troy University Resources

Input Dialogs

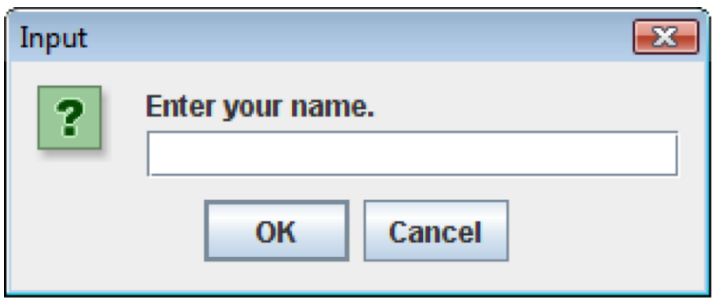
- An input dialog is a quick and simple way to ask the user to enter data.
- The dialog displays a text field, an Ok button and a Cancel button.
- If Ok is pressed, the dialog returns the user's input.
- If Cancel is pressed, the dialog returns null.

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

Input Dialogs

```
String name;
name = JOptionPane.showInputDialog(
          "Enter your name.");
```

- The argument passed to the method is the message to display.
- If the user clicks on the OK button, `name` references the string entered by the user.
- If the user clicks on the Cancel button, `name` references `null`.



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

The `System.exit` Method

- A program that uses `JOptionPane` does not automatically stop executing when the end of the main method is reached.
- Java generates a *thread*, which is a process running in the computer, when a `JOptionPane` is created.
- If the `System.exit` method is not called, this thread continues to execute.

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

The `System.exit` Method

- The `System.exit` method requires an integer argument.

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

- This argument is an *exit code* that is passed back to the operating system.
- This code is usually ignored, however, it can be used outside the program:
  - to indicate whether the program ended successfully or as the result of a failure.
  - The value 0 traditionally indicates that the program ended successfully.

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

Troy University Resources

Converting a String to a Number

- The `JOptionPane`'s `showInputDialog` method always returns the user's input as a `String`
- A `String` containing a number, such as "127.89, can be converted to a numeric data type.

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

The Parse Methods

- Each of the numeric wrapper classes, (covered in Chapter 10) has a method that converts a string to a number.
  - The `Integer` class has a method that converts a string to an `int`,
  - The `Double` class has a method that converts a string to a `double`, and
  - etc.
- These methods are known as *parse methods* because their names begin with the word "parse."

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

Troy University Resources

The Parse Methods

```
// Store 1 in bVar.
byte bVar = Byte.parseByte("1");

// Store 2599 in iVar.
int iVar = Integer.parseInt("2599");

// Store 10 in sVar.
short sVar = Short.parseShort("10");

// Store 15908 in lVar.
long lVar = Long.parseLong("15908");

// Store 12.3 in fVar.
float fVar = Float.parseFloat("12.3");

// Store 7945.6 in dVar.
double dVar = Double.parseDouble("7945.6");
```

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

Reading an Integer with an Input Dialog

```
int number;
String str;
str = JOptionPane.showInputDialog(
   "Enter a number.");
number = Integer.parseInt(str);
```

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

Troy University Resources

Reading a `double` with an Input Dialog

```
double price;
String str;
str = JOptionPane.showInputDialog(
    "Enter the retail price.");
price = Double.parseDouble(str);
```

See example: PayrollDialog.java

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