Lecture 2

Java Fundamentals

References:

Tony Gaddis, Chapter 2: Java Fundamentals, Starting out with Java: From Control Structures through Objects, 7 edition

1

Chapter Topics (1 of 2)

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

2

Chapter Topics (2 of 2)

- Creating named constants with `final`
- Comments
- Using the `Scanner` class for input
- Dialog boxes

A Java Program (1 of 2)

- A Java source code file containing one or more Java classes
  - Only one of them may be public
  - The public class and the filename of the source code file must match

    e.g., A public class named *Simple* must be in a file named *Simple.java*

A Java Program (2 of 2)

- See example: Simple.java
- To compile the example:
  - **`javac Simple.java`**
    - Notice the `.java` file extension
    - *Simple.class* created
- To run the example:
  - **`java Simple`**
    - No file extension here

Analyzing The Example (1 of 3)

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

public class Simple
{

}
```

This is a Java comment. It is ignored by the compiler.

This is the class header for the class Simple

This area is the body of the class Simple. All of the data and methods for this class will be between these curly braces.

Analyzing The Example (2 of 3)

```
// 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.

Analyzing The Example (3 of 3)

```
// 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.

Parts of a Java Program (1 of 3)

- Comments
  - Ignored by the compiler
- Class Header
  - Other classes can use it (public), a Java class (class), and class name (Simple).
- Curly Braces
  - Define the scope of the class or the scope of the method.

Parts of a Java Program (2 of 3)

- The `main` Method
  - The line of code that the *java* command will run first
- Java Statements
  - The statements executed within the `main` method

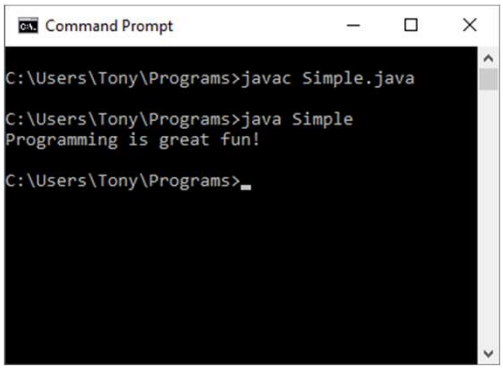
```
System.out.println("Programming is great
fun!");
```

Parts of a Java Program (3 of 3)

- The `println` method
  - Places a newline character at the end of the output
- However, the `print` method
  - Does not put a newline character at the end of the output.

Console Output

- Programs running in a console window



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**

Java Escape Sequences (1 of 2)

`\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

Java Escape Sequences (2 of 2)

- 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
```

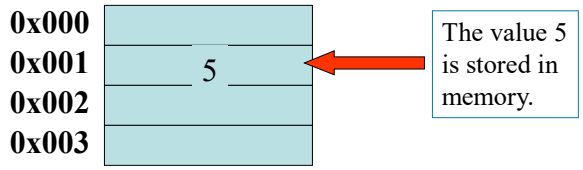

Variables and Literals (1 of 2)

- A variable
  - A named storage location in the computer's memory
- A literal
  - A value written into the code of a program

Variables and Literals (2 of 2)

This line is called a *variable declaration*.
`int value;`

The following line is known as an assignment statement.
`value = 5;`



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?

The + Operator

- The + operator used in two ways:
  - as a concatenation operator
  - as an addition operator

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

Identifiers (1 of 2)

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

Identifiers (2 of 2)

- 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.

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

Variable Names

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

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

- Programs should be *self-documenting*

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

Primitive Data Types

- Primitive data types
  - Built into the Java language and not derived from classes
- 8 Java primitive data types

```
byte
short
int
long
float
double
boolean
char
```

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

Variable Declarations

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

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.
- Integers embedded into Java source code are called ***integer literals***

Floating Point Data Types

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

Floating Point Literals (1 of 3)

- *floating point literals*
  - When floating point numbers are embedded into Java source code
- The default type for floating point literals is `double`.
  - 29.75, 1.76, and 31.51 are `double` data types.

Floating Point Literals (2 of 3)

- 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.`

Floating Point Literals (3 of 3)

- 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.

The `boolean` Data Type

- The Java `boolean` data type can have two possible values.
  - `true`
  - `false`

The `char` Data Type

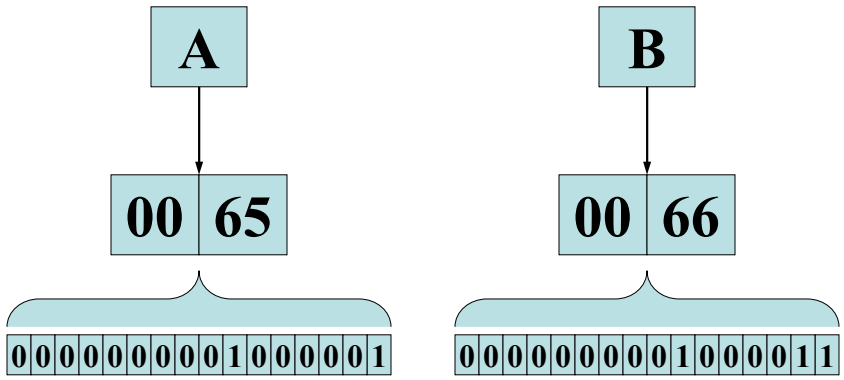
- The Java `char` data type provides access to single characters
- `char` literals are enclosed in single quote marks.
  - ‘a’, ‘Z’, ‘\n’, ‘1’
- Don’t confuse `char` literals with string literals
  - `char` literals are enclosed in single quotes.
  - String literals are enclosed in double quotes

Unicode (1 of 2)

- 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.

*American Standard Code for Information Interchange

Unicode (2 of 2)



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 compatible with the type of the variable

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;`

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

Combined Assignment Operators (2 of 2)

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.

Creating Constants with `final`

- Many programs have data that does not need to be changed
- By convention, constants are all upper case and words are separated by the underscore character

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

Commenting Code (1 of 3)

- Java provides three methods for commenting code.

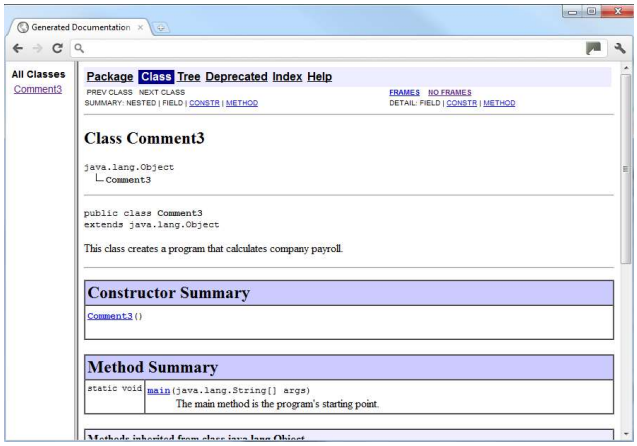
Comment Style	Description
`//`	Single line comment. Anything after the // on the line will be ignored by the compiler.
`/* … */`	Block comment. Everything beginning with /* and ending with the first */ will be ignored by the compiler. This comment type cannot be nested.
`/** … */`	Javadoc comment. This 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.

Commenting Code (2 of 3)

- 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.

Commenting Code (3 of 3)

- Example index.html:



The `Scanner` Class (1 of 2)

- `Scanner` class
  - To read input from the keyboard
  - Defined in `java.util`,
  - Use the following statement at the top of our programs:

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

The `Scanner` Class (2 of 2)

- Scanner objects work with `System.in`
- To create a `Scanner` object:
  `Scanner keyboard = new Scanner (System.in);`
- `Scanner` class has various methods
- See example: Payroll.java

Dialog Boxes

- A *dialog box*
  - A small graphical window displaying a message to the user or requests input
  - Using the `JOptionPane` class
- Two of the dialog boxes are:
  - Message Dialog: displays a message
  - Input Dialog: prompts the user for input

The JOptionPane Class (1 of 2)

- The following statement must be before the program's class header:

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

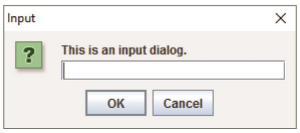
- This statement tells the compiler where to find the `JOptionPane` class.

The JOptionPane Class (2 of 2)

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

Message dialog

Input dialog



Message Dialogs

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

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

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`.

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*
  - A process running in the computer, when a `JOptionPane` is created
- If the `System.exit` method is not called,
  - this thread continues to execute

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


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.

The Parse Methods (1 of 2)

- Each of the numeric wrapper classes
  - 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`
- These methods known as *parse methods*

The Parse Methods (2 of 2)

```
// 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");
```

Reading an Integer with an Input Dialog

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


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