Introduction to Java

CSC 2014 – Java Bootcamp

Dr. Mary-Angela Papalaskari
Department of Computing Sciences
Villanova University

Some slides in this presentation are adapted from the slides accompanying Java Software Solutions by Lewis & Loftus

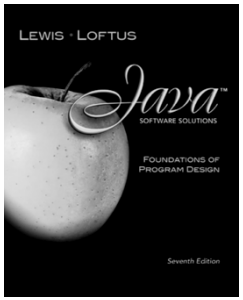
Course website

www.csc.villanova.edu/~map/2014/f14/

Links to:

- **Schedule** – topics, slides, projects, labs, code, etc.
- **Syllabus** – course information
- **Piazza** – class discussions, announcements
- **Blackboard** – submit projects, check grades

Our textbook



Java Software Solutions
Foundations of Program Design
Seventh Edition

John Lewis
William Loftus

Addison-Wesley
is an imprint of
PEARSON

An old quote

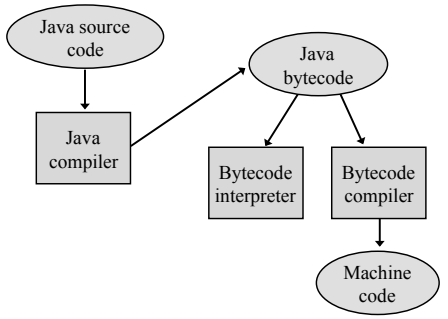
A priest asked: “What is Fate, Master?”
And he answered:
“It is that which gives a beast of burden its reason for existence. It is that which men in former times had to bear upon their backs. It is that which has caused nations to build byways from City to City upon which carts and coaches pass, and alongside which inns have come to be built to stave off Hunger, Thirst and Weariness.”
“And that is Fate?” said the priest.
“Fate... I thought you said Freight,” responded the Master.
“That's all right,” said the priest. “I wanted to know what Freight was too.”
- Kehlog Albran

Source unknown. This quote appeared as one of the “fortunes” displayed by the fortune cookie program on old unix systems. (“fortune” was a program that ran automatically every time you logged out of a unix session and displayed a random, pithy saying.)

High-level programming languages

- Programmer writes **Source code**
- Translation produces the binary equivalent – **Object code**
- Translation is performed by an assembler, compiler, or interpreter (stay tuned)

Java Translation



Development Environments

- There are many programs that support the development of Java software, including:
  - Sun Java Development Kit (JDK)
  - Sun NetBeans
  - IBM Eclipse
  - IntelliJ IDEA
  - Oracle JDeveloper
  - BlueJ
  - jGRASP ⬅
- Though the details of these environments differ, the basic compilation and execution process is essentially the same

Java Program Structure

- In the Java programming language:
  - A program is made up of one or more *classes*
  - A class contains one or more *methods*
  - A method contains program *statements*
- These terms will be explored in detail throughout the course
- A Java application always contains a method called `main`
- See `Lincoln.java`

Java Program Example

```
//********************************************************************
//  Lincoln.java       Author: Lewis/Loftus
//
//  Demonstrates the basic structure of a Java application.
//********************************************************************

public class Lincoln
{
   //-----------------------------------------------------------------
   //  Prints a presidential quote.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      System.out.println ("A quote by Abraham Lincoln:");

      System.out.println ("Whatever you are, be a good one.");
   }
}
```


Java Program Structure

```
//  comments about the class
public class MyProgram
{

}
```

class header

class body

Comments can be placed almost anywhere

Java Program Structure

```
//  comments about the class
public class MyProgram
{
   //  comments about the method
   public static void main (String[] args)
   {

   }

}
```

method header

method body

Comments

- Comments in a program are called *inline documentation*
- They should be included to explain the purpose of the program and describe processing steps
- They do not affect how a program works
- Java comments can take three forms:

```
// Basic this comment runs to the end of the line

/*  Basic this comment runs to the terminating
    symbol, even across line breaks        */

/** this is a javadoc comment   */
```

Identifiers

- *Identifiers* are the words a programmer uses in a program
- An identifier can be made up of letters, digits, the underscore character (_), and the dollar sign
- Identifiers cannot begin with a digit
- Java is *case sensitive* - `Total, total,` and `TOTAL` are different identifiers
- By convention, programmers use different case styles for different types of identifiers, such as
  - *title case* for class names - `Lincoln`
  - *upper case* for constants - `MAXIMUM`

Identifiers

- Sometimes we choose identifiers ourselves when writing a program (such as `Lincoln`)
- Sometimes we are using another programmer's code, so we use the identifiers that he or she chose (such as `println`)
- Often we use special identifiers called *reserved words* that already have a predefined meaning in the language
- A reserved word cannot be used in any other way

Reserved Words

- The Java reserved words:

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


White Space (Spaces, blank lines, and tabs)

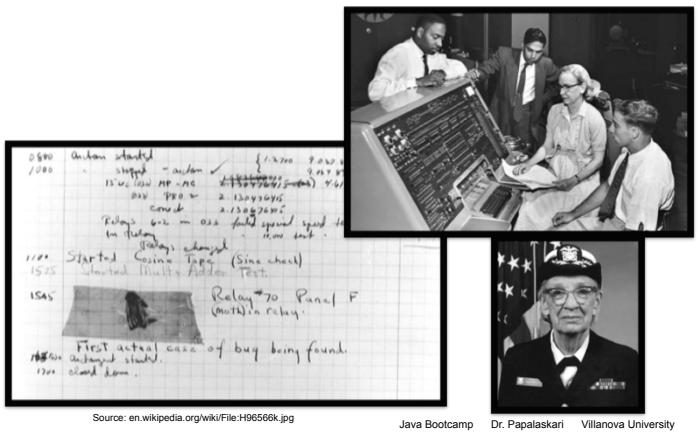
- Extra white space is ignored
- Programs should be formatted to enhance readability, using consistent indentation
- See `Lincoln2.java`, `Lincoln3.java`

Errors

Errors

- A program can have three types of errors
- The compiler will find syntax errors and other basic problems (*compile-time errors*)
  - If compile-time errors exist, an executable version of the program is not created
- A problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (*run-time errors*)
- A program may run, but produce incorrect results, perhaps using an incorrect formula (*logical errors*)

The original ”bug” found in the relays of Harvard’s Mark II computer by Admiral Grace Murray Hopper’s team.



Source: en.wikipedia.org/wiki/File:H96566k.jpg

Lab 1:

- Learn about jGrasp - the programming environment that we will be using in this class
  - Compile  …. and run  a java program
- Understand the relationship between a Java class name and the name of the .java file where the class is defined
- Practice using basic Java output statements and adding comments
- Learn about variables, string literals, concatenation. *E.g.,*

```
System.out.println ("Howdy " + name);
System.out.println ("The answer is " + x);
System.out.print ("Counting... up: " + (count + 1));
System.out.println ("  ... and\n ... down: " + (count - 1));
```

- Explore Java syntax
- Experience some errors!

Character Strings

- A *string literal* is represented by putting double quotes around the text
- Examples:

```
"This is a string literal."
"123 Main Street"
"X"
```


Character Strings

- A *string literal* is represented by putting double quotes around the text
- Examples:

```
"This is a string literal."
"123 Main Street"
"X"
```

spaces matter in here!

The println Method

- In the `Lincoln` program we invoked the `println` method to print a character string
- The `System.out` object represents a destination (the monitor screen) to which we can send output

```
System.out.println ("Whatever you are, be a good one.");
```

object — method name — information provided to the method (parameters)

The print Method

- In the `Lincoln` program we invoked the `println` method to print a character string
- The `System.out` object represents a destination (the monitor screen) to which we can send output
- `print` is similar to the `println` except that it does not advance to the next line

```
System.out.print ("Whatever you are, be a good one.");
```

object — method name — information provided to the method (parameters)

String Concatenation

- The *string concatenation operator* (+) is used to append one string to the end of another

```
"And one more " + "thing"
```


Hands on:

- Use MyQuote.java as a starting point (program from Lab 1), focus on this part of the code:

```
System.out.println ("Howdy " + name);
System.out.println ("The answer is " + x);
System.out.print ("Counting... up: " + (count + 1));
System.out.println ("  ... and\n ... down: " + (count - 1));
```

- Try the following:
  1) What if you remove the parentheses around (count + 1)?
  2) What happens if we try this way of breaking a line:

```
System.out.print ("Counting...
                  up: " + (count + 1));
```

  3) How can we get all this output to print all in one line?

- Other examples (textbook): Countdown.java Facts.java

Escape Sequences

- What if we wanted to print the quote character?
- Let's try something like this…

```
System.out.println ("I said "Hello" to you.");
```

- An *escape sequence* is a series of characters that represents a special character
- An escape sequence begins with a backslash character (\)

```
System.out.println ("I said \"Hello\" to you.");
```


Escape Sequences

- Some Java escape sequences:

Escape Sequence	Meaning
\b	backspace
\t	tab
\n	newline
\r	carriage return
\"	double quote
\'	single quote
\\	backslash

Example from textbook: `Roses.java`

```
//********************************************************************
//  Roses.java       Author: Lewis/Loftus
//
//  Demonstrates the use of escape sequences.
//********************************************************************

public class Roses
{
   //-----------------------------------------------------------------
   //  Prints a poem (of sorts) on multiple lines.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      System.out.println ("Roses are red,\n\tViolets are blue,\n" +
         "Sugar is sweet,\n\tBut I have \"commitment issues\",\n\t" +
         "So I'd rather just be friends\n\tAt this point in our " +
         "relationship.");
   }
}
```

Output

```
Roses are red,
        Violets are blue,
Sugar is sweet,
        But I have "commitment issues",
        So I'd rather just be friends
        At this point in our relationship.
```


Quick Check

Write a single `println` statement that produces the following output:

"Thank you all for coming to my home
tonight," he said mysteriously.

Variables

- A *variable* is a name for a location in memory
- A variable must be *declared* by specifying the variable's name and the type of information that it will hold

data type — variable name

```
int sum;
double milesPerGallon;
String name, petName;
```


Some types of data in Java

type	set of values	literal values	operations
`char`	characters	'A' '@'	compare
`String`	sequences of characters	"Hello World" "jackie123"	concatenate
`int`	integers	17 12345	add, subtract, multiply, divide
`double`	floating-point numbers	3.1415 6.022e23	add, subtract, multiply, divide
`boolean`	truth values	`true` `false`	and, or, not

Assignment Statement

- ***Changes the value of a variable***
- The assignment operator is the **=** sign

```
total = 55 - discount;
```

- The expression on the right is evaluated and the result is stored in the variable on the left

Combined declaration and assignment

A variable can be given an initial value in the declaration

```
int age = 18;
double x = 3.2, y = -0.80;
String name = scan.nextLine();
```

what's this?? (stay tuned)

Combined declaration and assignment

A variable can be given an initial value in the declaration

- a new value can be assigned later:

```
int age = 18;
double x = 3.2, y = -0.80;
String name = scan.nextLine();
age = 19;
x = x + 0.5;
name = scan.nextLine();
```


Combined declaration and assignment – Note: CANNOT declare twice

A variable can be given an initial value in the declaration

- a new value can be assigned later:

```
int age = 18;
double x = 3.2, y = -0.80;
String name = scan.nextLine();
int age = 19;
```

Error: declaring variable age again

Example

Computing the total number of seconds

```
int hours = 1;
int minutes = 25;
int seconds = 31;

int totalMinutes = (hours * 60) + minutes;
int totalSeconds = (totalMinutes * 60) + seconds;
```


Example

Computing the total number of seconds

Another alternative:

```
int hours = 1;
int minutes = 25;
int seconds = 31;

int totalSeconds =
        (hours * 3600) + (minutes * 60) + seconds;
```


Arithmetic Operators

Addition	+
Subtraction	-
Multiplication	*
Division	/
Remainder	%

- If either or both operands used by an arithmetic operator are floating point, then the result is a floating point

Division and Remainder

- If both operands are integers, the division result is an integer (the fractional part is discarded):

```
14 / 3        143 / 60

8 / 12        20 / 16
```

- % gives the remainder of the division:

```
14 % 3        143 % 60

8 % 12        20 % 16
```

Example

Extracting hours, minutes seconds from total number of seconds

An

```
int totalSeconds = 7222;
int hours = totalSeconds/3600;
int remainingSeconds = totalSeconds%3600;
int minutes = remainingSeconds/60;
int seconds = remainingSeconds%60;
```


Operator Precedence

```
result  =  total + count / max - offset;
```

Order of evaluation:

1. Multiplication, division, remainder
2. addition, subtraction, string concatenation

   – Operators with the same precedence: left →right
   – Use parentheses to override default order

Examples

```
a + b + c + d + e

a – b / c + d * e

a / (b + c) - d % e

a / (b * (c + (d - e)))
```


Assignment operator

- Assignment (=) copies the value of the right side into the memory location associated with the left side
- *It does not set up an ongoing equivalence*

```
int davesAge = 21;
int suesAge = davesAge;

davesAge = 22;

System.out.println (davesAge); // prints 22
System.out.println (suesAge);  // prints 21
```

Increment and Decrement

- The *increment operator* (++) adds one to its operand
- The *decrement operator* (--) subtracts one from its operand
- The statement

```
count++;
```

is functionally equivalent to

```
count = count + 1;
```


CONSTANTS: like variables, but value cannot change – declare using `final` modifier:

```
final int INCHES_PER_FOOT = 12;

final double LBS_PER_KG = 2.2;
```

Convention: Use UPPER_CASE identifiers

Summary

- Variable. A name that refers to a value of declared type.
- Literal. Programming language representation of a value.
- Assignment statement. Associates a value with a variable.