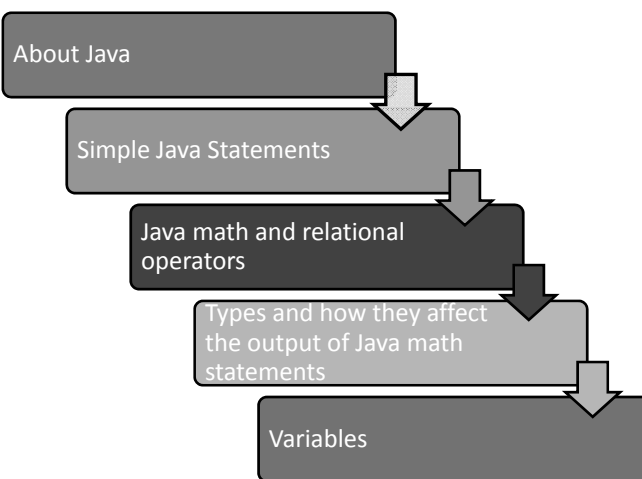




Introduction to Java

Objectives



- 
- This module will be your first foray into using Java to communicate with your computer. Some of the key concepts covered in this section include syntax errors, data types, math operators vs. relational operators, and Strings.
 - Some of the material we will explore includes the concept of an object or class, executing simple math types and variables.

Problem Solving

- 
- The purpose of writing a program is to solve a problem
 - The general steps in problem solving are:
 - Understand the problem
 - Dissect the problem into manageable pieces
 - Design a solution
 - Consider alternatives to the solution and refine it
 - Implement the solution
 - Test the solution and fix any problems that exist

Problem Solving

- Many software projects fail because the developer didn't really understand the problem to be solved
- We must avoid assumptions and clarify ambiguities
- As problems and their solutions become larger, we must organize our development into manageable pieces
- This technique is fundamental to software development
- We will dissect our solutions into pieces called classes and objects, taking an object-oriented approach

Introduction to Java

5

Programming Language Levels

- There are 4 programming language levels:
 - machine language
 - assembly language
 - high-level language
 - fourth-generation language
- The other levels were created to make it easier for a human being to write programs

Introduction to Java

6

Programming Languages

- A program must be translated into machine language before it can be executed on a particular type of CPU
- A compiler is a software tool which translates source code into a specific target language
- Often, that target language is the machine language for a particular CPU type
- The Java approach is somewhat different

The Java Programming Language

- A *programming language* specifies the words and symbols that we can use to write a program
- A programming language employs a set of rules that dictate how the words and symbols can be put together to form valid *program statements*
- Java was created by Sun Microsystems, Inc.
- It was introduced in 1995 and has become quite popular
- It is an object-oriented language

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
- There are two distinct types of Java programs:
- applications
 - Are stand alone programs. Java applications always execute main method. Java applications are executed by the Java interpreter.
- applets
 - are invoked as a part of a Web page and executed by a browser (such as Netscape's, IE, Firefox).

Introduction to Java

9

Java's Basic structure

- Comments:
 - comments are remarks that the compiler ignores
- Class Definition:
 - Everything in Java is an object, and a class is the template used to create an object.
- "main" Method:
 - The starting point to all programs.
- Braces:
 - Special characters used to mark the beginning and the end of blocks of code.
- Statements:
 - A statement is a line of Java codes. Statements ends with a semicolon.

Introduction to Java

10

Java Application Structure

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

class header

class body

Comments can be added almost anywhere

Introduction to Java

11

Java Application Structure

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

method header

method body

Introduction to Java

12

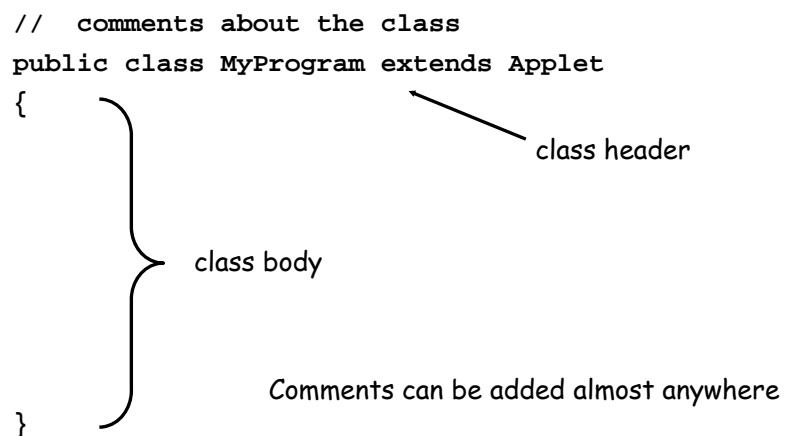
Applets

- ... are intended to be transported over the web and executed using a web browser.
- ... can be executed using the appletviewer tool of the Java SDK
- ... doesn't have a main method. I
- ... has several special methods that serve specific purpose.

Java Applet Structure

```
// comments about the class
public class MyProgram extends Applet
{
    class body
}

Comments can be added almost anywhere
```



Java Applet Structure

```
// comments about the class
public class MyProgram extends Applet
{
    // comments about the method
    public void paint(Graphics g)
    {
    }
}
```

method header

method body

Introduction to Java

15

Comments

- Comments in a program are also 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 two forms:

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

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

Introduction to Java

16

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
- They cannot begin with a digit
- Java is *case sensitive*, therefore `Total` and `total` are different identifiers

Introduction to Java

17

Identifiers

- Sometimes we choose identifiers ourselves when writing a program (such as `Turtle`)
- Sometimes we are using another programmer's code, so we use the identifiers that they 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

Introduction to Java

18

Reserved Words

- The Java reserved words:

<code>abstract</code>	<code>default</code>	<code>goto</code>	<code>operator</code>	<code>synchronized</code>
<code>boolean</code>	<code>do</code>	<code>if</code>	<code>outer</code>	<code>this</code>
<code>break</code>	<code>double</code>	<code>implements</code>	<code>package</code>	<code>throw</code>
<code>byte</code>	<code>else</code>	<code>import</code>	<code>private</code>	<code>throws</code>
<code>byvalue</code>	<code>extends</code>	<code>inner</code>	<code>protected</code>	<code>transient</code>
<code>case</code>	<code>false</code>	<code>instanceof</code>	<code>public</code>	<code>true</code>
<code>cast</code>	<code>final</code>	<code>int</code>	<code>rest</code>	<code>try</code>
<code>catch</code>	<code>finally</code>	<code>interface</code>	<code>return</code>	<code>var</code>
<code>char</code>	<code>float</code>	<code>long</code>	<code>short</code>	<code>void</code>
<code>class</code>	<code>for</code>	<code>native</code>	<code>static</code>	<code>volatile</code>
<code>const</code>	<code>future</code>	<code>new</code>	<code>super</code>	<code>while</code>
<code>continue</code>	<code>generic</code>	<code>null</code>	<code>switch</code>	

Introduction to Java

19

White Space

- Spaces, blank lines, and tabs are collectively called *white space*
- White space is used to separate words and symbols in a program
- Extra white space is ignored
- A valid Java program can be formatted many different ways
- Programs should be formatted to enhance readability, using consistent indentation

Introduction to Java

20

Programming Language Levels

- There are four programming language levels:
 - machine language
 - assembly language
 - high-level language
 - fourth-generation language
- Each type of CPU has its own specific *machine language*
- The other levels were created to make it easier for a human being to write programs

Introduction to Java

21

Programming Languages

- A program must be translated into machine language before it can be executed on a particular type of CPU
- This can be accomplished in several ways
- A *compiler* is a software tool which translates *source code* into a specific target language
- Often, that target language is the machine language for a particular CPU type
- The Java approach is somewhat different

Introduction to Java

22

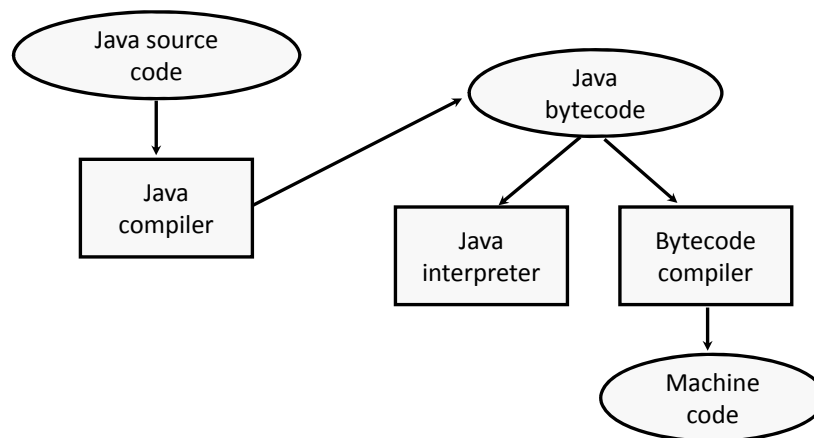
Java Translation and Execution

- The Java compiler translates Java source code into a special representation called *bytecode*
- Java bytecode is not the machine language for any traditional CPU
- Another software tool, called an *interpreter*, translates bytecode into machine language and executes it
- Therefore the Java compiler is not tied to any particular machine
- Java is considered to be *architecture-neutral*

Introduction to Java

23

Java Translation and Execution



Introduction to Java

24

Development Environments

- There are many development environments which develop Java software:
 - Sun Java Software Development Kit (SDK)
 - Borland JBuilder
 - MetroWork CodeWarrior
 - Microsoft Visual J++
 - Netbeans
- Though the details of these environments differ, the basic compilation and execution process is essentially the same

Introduction to Java

25

Setting up Netbeans w/Java

- I know I said we would do this in an assignment but I decided against it.
- Make sure you have Netbeans with Java installed. Follow the download link here: <http://netbeans.org/downloads/>
You only need Java SE.
- Follow this tutorial to learn how to use Netbeans:
 - <http://netbeans.org/kb/docs/java/quickstart.html>

Introduction to Java

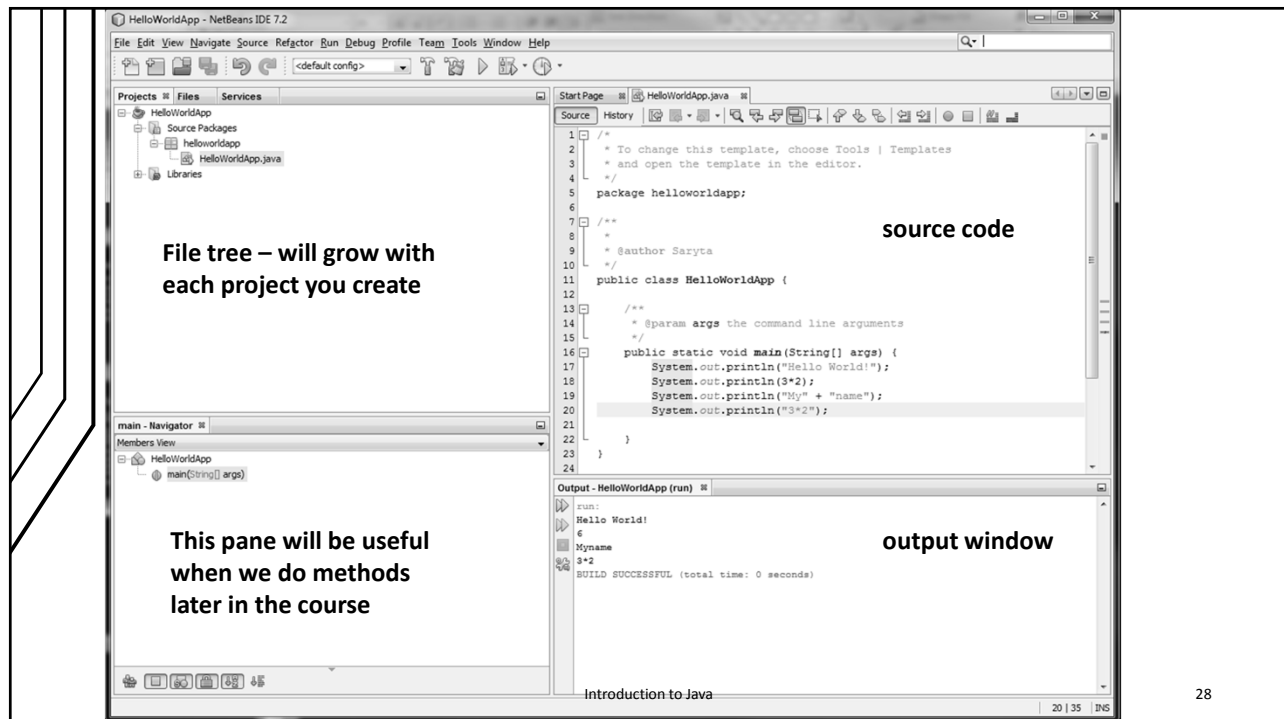
26

Operators, Types, and Syntax Errors

- At this point, you should have Netbeans up and running. If not, please do so before you go any further.
- Make sure you have the tutorial example up and running. We are going to add some code to the main where you entered `System.out.println("Hello World!");`
- Enter the following lines of code below the above line:
 - `System.out.println(3*2);`
 - `System.out.println("My" + "name");`
 - `System.out.println("3*2");`
- As in the tutorial, run the program. (see next slide for what you should see)

Introduction to Java

27



28

Tip...

- A common error in java is to forget some of the punctuation. It is very important to remember to include the correct number of (,) and " characters, for example. A simple rule to remember is to always close what you open. So every (must be followed by a) and every " must be followed by a ".

syntax error

- (definition) A syntax error is an error in the syntax of your code that makes it impossible for the computer to understand. Formally speaking, syntax is "a system of rules and principles that govern the sentence structure" of a language ([Wikipedia](#)).
- When a fussy computer is involved, these rules can sometimes include even hard to see things like where you put a newline or a space, and whether you remember that final) or ". You will need to learn the syntax rules for Java to write programs that work.
- You can compare programming Java to learning a new language – it has a grammar too.

note

- One of the *most common* syntax errors is to forget to end a statement in Java with a semicolon (;).
- In your programs (anything you compile), you have to end your statements with semicolons.
- For example,
Correct:
`System.out.println("hello world");`
not correct
`System.out.println("hello world")`

Syntax and Semantics

- The *syntax rules* of a language define how we can put symbols, reserved words, and identifiers together to make a valid program
- The *semantics* of a program statement define what that statement means (its purpose or role in a program)
- A program that is syntactically correct is not necessarily logically (semantically) correct
- A program will always do what we tell it to do, not what we meant to tell it to do

Errors

- A program can have three types of errors
- The compiler will find problems with syntax and other basic issues (*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 (*logical errors*)

Packages

- The classes of the Java standard class library are organized into packages
- Some of the packages in the standard class library are:

Package

`java.lang`
`java.applet`
`java.awt`
`javax.swing`
`java.net`
`java.util`

Purpose

General support
Creating applets for the web
Graphics and graphical user interfaces
Additional graphics capabilities and components
Network communication
Utilities

The import Declaration

- When you want to use a class from a package, you could use its fully qualified name
 - `java.util.Random`
- Or you can import the class, then just use the class name
 - `import java.util.Random;`
- To import all classes in a particular package, you can use the * wildcard character
 - `import java.util.*;`

Introduction to Java

35

Libraries

- A library is collection of already written useful pieces of code that implements many common programming tasks.
- When we write new Java programs, we usually combine our new code with code extracted from these libraries. This process is called linking.
- import statements: inform Java compiler that we will be using classes from the awt and applet packages.
- applet package: an applet class is derived from the class Applet.
- awt package: abstract windowing toolkit. Classes in this package are used to create GUI objects.

Introduction to Java

36

Libraries

- a collection of already written useful pieces of code that implements many common tasks.
- We usually combine our code with code extracted from these libraries. This process is called linking.
- import statements inform the Java compiler that we will be using classes from the library.
 - applet package: Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
 - awt package: (abstract windowing toolkit) Contains all of the classes for creating UI and for painting graphics and images.

Introduction to Java

37

The import Declaration

- When you want to use a class from a package, you could use its fully qualified name
 - `java.awt.Graphics;`
- Or you can import the class, then just use the class name
 - `import java.awt.Graphics;`
- To import all classes in a particular package, you can use the wildcard character (`*`)
 - `import java.awt.*;`

Introduction to Java

38

Operators

- Since computers represent everything using numbers, one of the easiest ways to start talking to them is by asking them to do math.
- In your test run above you entered:
 - `System.out.println(3*2);`
 - This statement takes `3*2` and multiplies the two numbers together and displays them to the screen.
 - `System.out.println("3*2");`
 - This statement displays `3*2` but does not output the results because it is surrounded by quotations (`"`). This is called a String. More later on this.

- The non-numeric symbols you just used in your numeric statements are called *operators*. Java has many different kinds of operators.
- Operator:
 - (definition) An operator is a primitive (very very simple) java command used to combine data in specific ways. Examples include `+`, `*`, `-`, `/`, and `%` (these are called math operators) or `>=`, `<` and others (called relational operators).
- Relational operator
 - (definition) A relational operator is a primitive (very very simple) java command used to decide whether something is true or false. Examples include `>`, `<`, `==` (the same), `!=` (not the same), `<=`, `>=`. Each of these operators compares two things in some way. `3 > 4` will compare 3 to 4 and return false since 3 is not greater than 4.

- Most of those operators should be very familiar to you if you have ever used a calculator. Here is a summary of how they work:

Greater Than >	Result	Less Than <	Result
4 > 3	true	2 < 3	false
3 > 3	false	3 < 2	false
Equal ==	Result	Not Equal !=	Result
3 == 3	true	3 != 4	true
3 == 4	false	3 != 3	false

Greater than or equal to >=	Result	Less than or equal to <=	Result
3 >= 4	false	2 <= 4	true
3 >= 3	true	2 <= 2	true
2 >= 3	false	4 <= 3	false

Introduction to Java

41

- Just like in math, Java uses parentheses to specify the order that arithmetic operations are executed.
- By default (if no parentheses are specified), Java uses the standard order of operations: multiplication and division, then addition and subtraction.
- For more on order of operations, see this [Wikipedia article](#).
- Parentheses can be used to group things so you know what an operator applies to.
 - For example, $3 + 4 / 2 * 5$ (which equals 13) is very different from $((3 + 4) / 2) * 5$ (which equals 17.5) which is very different from $(3 + (4 / 2)) * 5$ (which equals 25), which is different again from $(3 + (4 / 2) * 5)$ (which equals 13).

Introduction to Java

42

Learn by doing

- What will $5.0 + 4.0 / 4.0 + 3.0$ be evaluated as?
- What will $(5.0 + 4.0) / (4.0 + 3.0)$ be evaluated as (round to the nearest tenth)?
- What will $(5.0 + 4.0) / 4.0 + 3.0$ be evaluated as (rounded to the nearest tenth)?

Our answer

- What will $5.0 + 4.0 / 4.0 + 3.0$ be evaluated as?
 - $= 5.0 + 1 + 3.0 = 9$
- What will $(5.0 + 4.0) / (4.0 + 3.0)$ be evaluated as (round to the nearest tenth)?
 - $= 9.0 / 7.0 = 1.3$
- What will $(5.0 + 4.0) / 4.0 + 3.0$ be evaluated as (rounded to the nearest tenth)?
 - $= 9.0 / 4.0 + 3.0 = 2.25 + 3.0 = 5.3$

Tip...

- You must always "balance your parentheses" when programming.
- For example, $(3 + (4/2)*5)$ will cause a syntax error if you use it in a program, as will `System.out.println("hello";`
- You also have to match the parentheses: for example, if you have used (as a left parentheses, then you must use) as a right parentheses, not } or any other closing element.

- Unlike a calculator, Java's behavior is not defined solely by what operator is being used.
- The *type* of the thing each operator is applied to has a big effect on how the operator works.
- For example, you can actually add (or, more precisely *concatenate*) two pieces of text together using + (go ahead and try it in Netbeans -- "hello"+"world" will give you "helloworld", for example.)

- **Concatenate**

- (definition) concatenate is a verb that describes the joining of two things together.
- In java, we concatenate using the + operator.

Learn by doing

- As described above, we can join two strings together using "string1" + "string2". It's also possible to join something like a string and a number.
- Create the code in Netbeans to find the results of:
 - "the result is " + 2 + 3
 - "the result is " + (2 + 3)

Our answer

- `System.out.println("the result is " + 2 + 3);`
- `System.out.println("the result is " + (2 + 3));`
- the result is 23
- the result is 5

- Java is a strongly typed language. This means that everything that has a value in Java also has a type. The type of something is used by the computer to interpret it. For literal values such as 3, 3.0, and "hello", the compiler determines the type.
- For example, 3 is an integer, while 3.0 is floating point (usually represented as a double) because it has a fractional part.
- The result of an operation is always the same as the operands.

- Because Java thinks of 3 as an integer, it will always return an integer when you do an operation with 3. This leads to non-intuitive results.
- For example:
 - if you type $1/3$ into Netbeans, the result is 0 instead of 0.3333 because both 1 and 3 are integers, and therefore the calculated result should be an integer.
 - Java always discards any fractional part in this situation (always rounds down, in other words), so the answer is 0.
- When only doubles or at least one double is used for the operation, the operation returns double even if the result could be accurately stored in an int.
- For example:
 - when $1.0/3$ is calculated, the operation returns 0.33333 instead of 0, and when $4.0/2$ is calculated, the operation returns 2.0 instead of 2.

Introduction to Java

51

Learn by doing

- What will $(5/4)$ be evaluated as (what is the value of $5/4$)?
- What is the type of the value of $(5/4)$?
- Why isn't $3/4$ equal to .75?

Introduction to Java

52

Our answer

- What will $(5/4)$ be evaluated as (what is the value of $5/4$)?
 - 5 and 4 are both integers, so $5/4$ should return an integer result. Java truncates any non integer values when doing integer arithmetic.
- What is the type of the value of $(5/4)$?
 - Look for multiple literals or variable names linked by an operator to find the expressions in this code. Here, both 5 and 4 are integers, so the answer should also be an integer.
- Why isn't $3/4$ equal to .75?
 - 3 and 4 are both ints, so Java assumes that you want the result as an int. In this case, the answer can be thought of as the integer 1 plus .75. Since Java thinks you want an integer, it returns the integer part of that only. It doesn't round for you, because if you wanted that, you should say so!

- Numbers, such as ints and doubles, are only one of the core types that Java understands. Numbers alone are fine for computers to interpret, but not a very easy way for people to think about anything but math.
- Java can also handle other types of information, such as Strings (groups of characters that represent text) and boolean expressions (statements that test whether something is true or false).

Variables

- Variables are an extremely useful part of your code for several reasons:
 - Holding information:
 - Variables can store information temporarily for you so you don't have to write a long and complex piece of code all on one line!
 - Making your code more legible
 - Use. Names. That. Make. Sense. ... PLEASE! ... actually, this is a requirement of the course.
 - Specifying the type of something
 - Creating a variable forces you to think about what the type of something is. Also, when you declare a variable to be an int, or double, or String, it will keep its type. Java is strongly typed, meaning that it really cares that you don't mix and match types without telling it, and will warn you if you make a mistake.

Introduction to Java

55

Some Variables

- Suppose you are in a restaurant and have just finished eating. Here are a series of variables you might need:

```
// the number of people eating at the table with you
int numPeople = 5;
// the cost of the meal
double bill = 220.50;
// the tip
double tip = bill*.2;
// the cost per person
double perPerson = (bill + tip)/numPeople;
// the name of the restaurant
String place = "Casbah";
// the answer to, "Is the cost per person < 10?"
boolean lowCost = perPerson < 10;
```

56

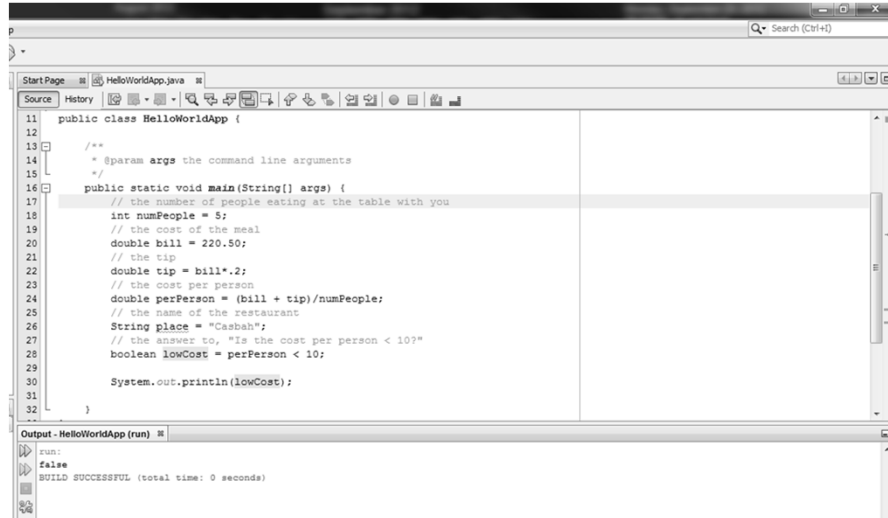
Tip...

- When you see source code like the program you just read, you may want to try it out in Netbeans.

Learn by doing

- In the code above, lines starting with `//` are comments. Every variable starts with a description of its type, then its name, and after the equals sign comes its value. So `numPeople` is an `int` (integer) whose value is 5. See if you can figure out the value of `lowCost`.
- hint:
 - you can type each line of code into Netbeans and then type `System.out.println(lowCost)` to see the result.

Our answer



The screenshot shows an IDE window titled 'HelloWorldApp.java'. The code defines a public class HelloWorldApp with a main method. The main method calculates the cost per person at a restaurant named 'Casbah' and prints the result. The output window shows the result 'false'.

```
11 public class HelloWorldApp {
12
13     /**
14      * @param args the command line arguments
15      */
16     public static void main(String[] args) {
17         // the number of people eating at the table with you
18         int numPeople = 5;
19         // the cost of the meal
20         double bill = 220.50;
21         // the tip
22         double tip = bill*.2;
23         // the cost per person
24         double perPerson = (bill + tip)/numPeople;
25         // the name of the restaurant
26         String place = "Casbah";
27         // the answer to, "Is the cost per person < 10?"
28         boolean lowCost = perPerson < 10;
29
30         System.out.println(lowCost);
31     }
32 }
```

Output - HelloWorldApp (run)

```
run:
false
BUILD SUCCESSFUL (total time: 0 seconds)
```

Introduction to Java

59

Variable Rules

- Must start with a letter (A-Z, a-z)
- Can contain any number of letters or digits
- Can contain the underscore (_) or \$
- Can be any length
- Case sensitive
- Recommendation: use a lower case letter (a-z) when defining a variable and capital case for class names.


Introduction to Java

60

Assignment

- An assignment statement changes the value of a variable
- The assignment operator is the = sign

```
total = 55;
```



- The expression on the right is evaluated and the result is stored in the variable on the left
- The value that was in total is overwritten
- You can only assign a value to a variable that is consistent with the variable's declared type

Introduction to Java

61

Constants

- A constant is an identifier that is similar to a variable except that it holds one value for its entire existence
- The compiler will issue an error if you try to change a constant
- In Java, we use the final modifier to declare a constant
 - `final int MIN_HEIGHT = 69;`
- Constants:
 - facilitate changes to the code
 - prevent inadvertent errors

Introduction to Java

62

Primitive Data

- There are exactly eight primitive data types in Java
- Four of them represent integers:
 - byte, short, int, long
- Two of them represent floating point numbers:
 - float, double
- One of them represents characters:
 - char
- And one of them represents boolean values:
 - boolean

Introduction to Java

63

Numeric Primitive Data

- The difference between the various numeric primitive types is their size, and therefore the values they can store:

Type	Storage	Max/Min Value
byte	8 bits	+/- 2^7
short	16 bits	+/- 2^{15}
int	32 bits	+/- 2^{31}
long	64 bits	+/- 9×10^{18}
float	32 bits	+/- 3.4×10^{38} (8)
double	64 bits	+/- 1.7×10^{308} (16)

Significant digits (# of digits displayed at any time. Anything greater we see exponent references)

Introduction to Java

64

Numeric Data: Limitations

- Numeric types are representations of number systems, so there are tradeoffs:
 - A finite number of integers can be represented.
 - A finite number of values between 1.11 and 1.12..
- Debugging Tip: A round-off error is the inability to represent certain numeric values exactly.
 - A float can have at most 8 significant digits. So 12345.6789 would be rounded off to 12345.679.
- Debugging Tip: In using numeric data the data type you choose must have enough precision to represent the values your program needs.

Characters

- A char variable stores a single character from the Unicode character set
- A character set is an ordered list of characters, and each character corresponds to a unique number
- The Unicode character set uses sixteen bits per character, allowing for 65,536 unique characters
- It is an international character set, containing symbols and characters from many world languages
- Character literals are delimited by single quotes:
 - 'a' 'X' '7' '\$' ',' '\n'

Characters

- The ASCII character set is older and smaller than Unicode, but is still quite popular
- The ASCII characters are a subset of the Unicode character set, including:

uppercase letters	A, B, C, ...
lowercase letters	a, b, c, ...
Punctuation	period, semi-colon, ...
digits	0, 1, 2, ...
special symbols	&, , \, ...
control characters	carriage return, tab,...

Boolean

- A boolean value represents a true or false condition
- A boolean can also be used to represent any two states, such as a light bulb being on or off
- The reserved words true and false are the only valid values for a boolean type
- `boolean done = false;`

Literals

- A literal is an explicit data value used in a program.
- Integer literals:
25 69 -4288
- Floating point literals:
3.14159 42.075 -0.5
- String literals:
"The result is: " "Hello"

Introduction to Java

69

Learn by doing

- Are the following variables valid or invalid?

int
anint
i
i1
23
thing2
2thing
ONE-HUNDRED
ONE_HUNDRED
something2do

Introduction to Java

70

Our answers

int	no	reserved word
anint	yes	should be anInt
i	yes	
i1	yes	
23	No	literal number
thing2	yes	
2thing	no	starts with a digit
ONE-HUNDRED	no	contains a hyphen -
ONE_HUNDRED	yes	upper case -> constant
something2do	yes	

Introduction to Java

71

Arithmetic Expressions review

- An expression is a combination of operators and operands
- Arithmetic expressions compute numeric results and make use of the arithmetic operators:
 - Addition +
 - Subtraction -
 - Multiplication *
 - Division /
 - Remainder %
- If either or both operands to an arithmetic operator are floating point, the result is a floating point

Introduction to Java

72

Division and Remainder

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

14 / 3 equals? 4

8 / 12 equals? 0

- The remainder operator (%) returns the remainder after dividing the second operand into the first

14 % 3 equals? 2

8 % 12 equals? 8

Introduction to Java

73

Operators

- An operator performs a function on one, two, or three operands.
- An operator that requires one operand is called a unary operator. (i.e.: +/- pos/neg)
- An operator that requires two operands is a binary operator. (i.e.: = is a binary operator that assigns the value from its right-hand operand to its left-hand operand)
- A ternary operator is one that requires three operands. The Java programming language has one ternary operator, ?:, which is a shorthand if-else statement.

Introduction to Java

74

Operator Precedence

- Operators can be combined into complex expressions
 - `result = total + count / max - offset;`
- Multiplication, division, and remainder are evaluated prior to addition, subtraction, and string concatenation
- Arithmetic operators with the same precedence are evaluated from left to right
- Parentheses can always be used to force the evaluation order

Introduction to Java

75

Assignment statements

- The assignment operator has a lower precedence than the arithmetic operators

First the expression on the right hand side of the = operator is evaluated

`answer = sum / 4 + MAX * lowest;`

The diagram illustrates the evaluation order of the expression `sum / 4 + MAX * lowest`. A bracket is placed under `4 + MAX * lowest`, and another bracket is placed under `MAX * lowest`. An arrow points from the first bracket to the assignment operator `=`, and another arrow points from the second bracket to the multiplication operator `*`.

- Then the result is stored in the variable on the left hand side

Introduction to Java

76

Reusing variables

- As an example, suppose that you decided to go to another restaurant with the same five people as above. You might update your variables as follows:

```
// the cost of the meal
double bill = 53.35;
// the tip
double tip = bill*.2;
// the cost per person
double perPerson = (bill + tip)/numPeople;
// the name of the restaurant
String place = "McDonalds";
// the answer to, "Is the cost per person < 10?"
boolean lowCost = perPerson < 10;
```

Introduction to Java

77

Learn by doing

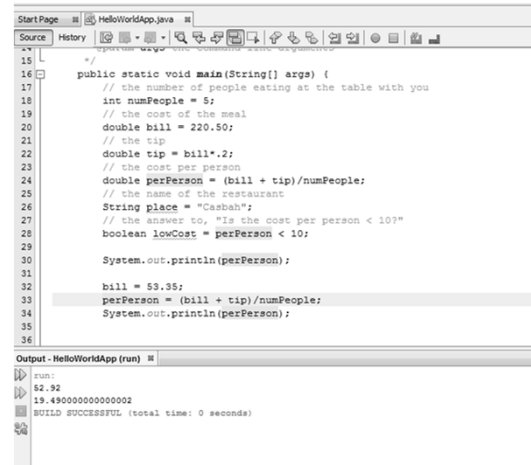
- Try this in Netbeans.
 - Assuming you already typed in the original code, it will still remember what a bill, tip, and so on is.
 - Notice that this time around, we didn't have to *declare* the variables, meaning we didn't have to specify their type.
 - This is because all we're doing is changing the contents of their memory box. Java already knows what type they are.
- What is the original and new value of perPerson?
- Can you figure out how to tell the total cost perPerson for the evening?

Introduction to Java

78

Our solution

- We can reassign the value of bill and perPerson to find the new value.
- This is destructive – the original values of bill and perPerson are lost.



```
15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
public static void main(String[] args) {
    // the number of people eating at the table with you
    int numPeople = 5;
    // the cost of the meal
    double bill = 220.50;
    // the tip
    double tip = bill*.2;
    // the cost per person
    double perPerson = (bill + tip)/numPeople;
    // the name of the restaurant
    String place = "Casbah";
    // the answer to, "Is the cost per person < 10?"
    boolean lowCost = perPerson < 10;

    System.out.println(perPerson);

    bill = 53.92;
    perPerson = (bill + tip)/numPeople;
    System.out.println(perPerson);
}
```

Output - HelloWorldApp (run)

run: 53.92
19.490000000000002
BUILD SUCCESSFUL (total time: 0 seconds)

Introduction to Java

79

Learn by doing

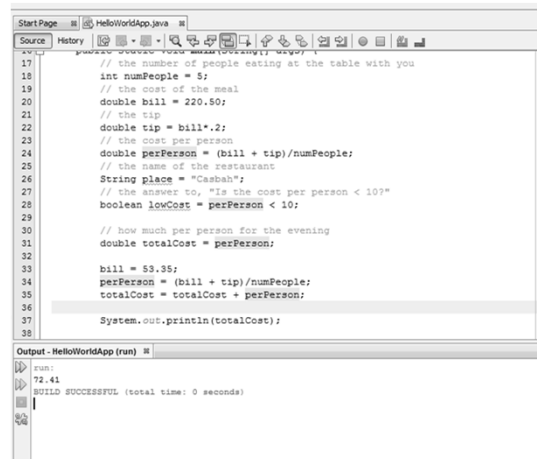
- How would you find the total value per person for the evening?

Introduction to Java

80

Our solution

- We have two choices:
 - Create two new variables for bill and perPerson.
- OR
- Create a new variable to hold a running total.



```
17 // the number of people eating at the table with you
18 int numPeople = 5;
19 // the cost of the meal
20 double bill = 220.50;
21 // the tip
22 double tip = bill*.2;
23 // the cost per person
24 double perPerson = (bill + tip)/numPeople;
25 // the name of the restaurant
26 String place = "Casbah";
27 // the answer to, "Is the cost per person < 10?"
28 boolean lowCost = perPerson < 10;
29
30 // how much per person for the evening
31 double totalCost = perPerson;
32
33 bill = 53.35;
34 perPerson = (bill + tip)/numPeople;
35 totalCost = totalCost + perPerson;
36
37 System.out.println(totalCost);
38
```

Output - HelloWorldApp (run)

```
run
72.41
BUILD SUCCESSFUL (total time: 0 seconds)
```

Introduction to Java

81

Data Conversions

- Sometimes it is convenient to convert data from one type to another
- For example, we may want to treat an integer as a floating point value during a computation
- Conversions must be handled carefully to avoid losing information
- Widening conversions are safest because they tend to go from a small data type to a larger one (such as a short to an int)
- Narrowing conversions can lose information because they tend to go from a large data type to a smaller one (such as an float to a int)

Introduction to Java

82

Data Conversions

- In Java, data conversions can occur in three ways:
 - assignment conversion
 - arithmetic promotion
 - casting
- Assignment conversion occurs when a value of one type is assigned to a variable of another
- Only widening conversions can happen via assignment
- Arithmetic promotion happens automatically when operators in expressions convert their operands

Introduction to Java

83

Data Conversions

- Casting is the most powerful, and dangerous, technique for conversion
- Both widening and narrowing conversions can be accomplished by explicitly casting a value
- To cast, the type is put in parentheses in front of the value being converted
- For example, if total and count are integers, but we want a floating point result when dividing them, we can cast total (count will become a float by arithmetic promotion):

```
result = (float) total / count;
```

Introduction to Java

84

Statements and Blocks

- A statement forms a complete unit of execution and is terminated with a semicolon (;).
- There are three kinds of statements:
 - expression statements - assignment
 - declaration statements – declare a variable
 - control flow statements – loops
- You can group zero or more statements together into a block with curly brackets ({ and }).

Introduction to Java

85

Syntax

- Must have an equal number of brackets { } and parenthesis ()
- Statements end with a semicolon - ;
- Good formatting of the source code leads to ease of readability and debugging
- When using a float, you must place a f after your value i.e.: 0.20f
- We will be coming back declaring and using variables again.

Introduction to Java

86

Writing Legible Code

- Code
 - (definition) Code is the name for the words that you write when you program. Another word for code is "a program". Another word for programming is "coding." Code is written in a specific language (a "programming language"). A program, or a piece of code, gives instructions to the computer that it can understand and execute.
- Comment
 - (definition) A comment is some text in your program that explains how the program works to humans but is ignored by the computer. You can write a comment in two ways. Either put `//` right at the start of each line of a comment, or use `/*` to start your comment and `*/` to end your comment, with anything you want over multiple lines in between. Both are demonstrated in the example below.

- From here on out, you should write code not only to be functional, but also to be legibility. Programming languages are designed to bridge the gap between people and computers in both directions. However, neither computers, nor people, will understand clearly what your code is meant to do unless you work to make your code legible.

Rules for writing legible code

- Always comment your code
 - Comments are for humans only, they are ignored completely by the computer. Some people write a program out in comments, with explanations, before they even write a line of code! In your code, each piece of logic should be explained in comments.
- Always use meaningful variable names.
 - If you had three kids, you would not name them "a" "b" and "c". Worse, imagine if your spouse was named "d". No one would know who you were talking about or what kind of thing they were (an adult? a child? a person?). Most programs that get written eventually is read by someone else! Your programs need to be legible to people (including yourself after you've forgotten about it), not just computers.
- Variable names start in lowercase, objects in uppercase
 - Variable names should start in lowercase ("bill", not "Bill"). If you use multiple words, you may capitalize each subsequent word ("billAndTip"). This helps to differentiate variables from other names in your code, such as objects, which start in uppercase.
- Indent your code properly
 - Indenting your code properly greatly increases legibility. A good rule of thumb for indenting code is to add a level of indentation within a set of curly braces (`{}`). An easy way to properly indent your code in Netbeans is to select format under the Source menu.

Introduction to Java

89

Summary

- We have used Netbeans to print something out
- We have asked Java to do simple math for us using Netbeans.
- We wrote some example programs and executed them.
- We used the `main()` method to cause something to be printed to the screen.
- We also learned about writing legible code, including the importance of commenting and naming
- We also covered some of the uses of variables, including:
 - holding information
 - making your code more legible
 - specifying the type of something

Introduction to Java

90