Hello World

CS160

Basic Computation (Savitch, Chapter 2)

TOPICS

- Variables and Data Types
- Expressions and Operators
- Integers and Real Numbers
- Characters and Strings
- Input and Output

Hello World

Variables in a programming language

- Variables store information
  - You can think of them like boxes
  - They "hold" values
  - The value of a variable is its current contents
- Note that this differs from variable in math
  - In math, a variable is an "unknown"
    - Solving an equation reveals its value
  - In programming, variables are "place holders" for values
    - Their current value is always known
    - The program changes their values to achieve a goal

Hello World

Data Types

- Variables are like boxes: they hold values
  - But you can't put an elephant in a shoe box
  - Different boxes hold different types of things
- Therefore, variables have *data types*
  - The data type describes the set of values a variable might contain
  - The value of a variable is a member of the set defined by its data type
  - Examples: int, char, double, boolean, String
  - Java is a strongly typed language – only values of the correct type can be put into a variable

Hello World

Creating Variables

- You create new variables through *declarations.*
  - Examples:

```
int daysPerYear;
char vowel;
```

- You assign values to variables using the *assignment operator '='.*
  - Examples:

```
daysPerYear = 365;
vowel = 'a';
```

- Declaration and assignment can be combined.
  - Examples:

```
int price = 25;
char c = '&';
```

More about Variables

- An uninitialized variable is useless
  - So it's good practice to initialize when declaring variables
  - can be done with one statement:

```
int daysPerYear = 365;
```

- Variables can be re-used:

```
int daysPerYear = 365;
// random code here
daysPerYear = 110;
```


Literals

- *Literals* are values that are directly recognized by Java:
  - numbers

    `237, 10, 9, 1.5, 5.8, 99.999`
  - characters

    `'a', 'Z', '0', '$'`
  - Strings

    `"hello", "there"`

Java Identifiers

- An *identifier* is a name, such as the name of a variable.
- Identifiers may contain only
  - Letters
  - Digits (0 through 9)
  - The underscore character (_)
  - And the dollar sign symbol ($) which has a special meaning
- The first character cannot be a digit.

Java Identifiers

- Identifiers may not contain any spaces, dots (.), asterisks (*), or other characters:

  `7-11 netscape.com util.*` // not allowed!
- Identifiers can be arbitrarily long.
- Since Java is case sensitive, `stuff`, `Stuff`, and `STUFF` are different identifiers.

Keywords or Reserved Words

- Words such as if are called *keywords* or reserved words and have special, predefined meanings.
  - Cannot be used as identifiers.
  - See Appendix 1 for a complete list of Java keywords.
- Examples: `int`, `public`, `class`

Naming Conventions

- Class types begin with an uppercase letter (e.g. String).
- Primitive types begin with a lowercase letter (e.g. int).
- Variables of both class and primitive types begin with a lowercase letters (e.g. myName, myBalance).
- Multiword names are "punctuated" using uppercase letters.

Where to Declare Variables

- Declare a variable
  - Just before it is used or
  - At the beginning of the section of your program that is enclosed in {}:

```
public static void main(String[] args)
{
   /* declare variables here */
   . . .
   /* code starts here */
   . . .
}
```


Java Types

- In Java, there are two kinds of data types:
  - Primitive data types
    - Hold a single, indivisible piece of data
    - Pre-defined by the language
    - Examples: int, char, double, boolean
  - Reference data types (Classes)
    - Hold complex combinations of data
    - Programs may define new classes
    - Examples: String, Scanner, System

Let's start with these

Primitive Types

- Integer types: byte, short, int, and long
  - int is most common
- Floating-point types: float and double
  - double is more common
- Character type: char
- Boolean type: boolean

Primitive Data Types

- The 4 most common primitive data types

Data Type	Description	Example
int	- integer values	5
double	- floating-point values	3.14
char	- characters	'J'
boolean	- either true or false	true

Notice – there are no quotes around true!

Primitive Types

Type Name	Kind of Value	Memory Used	Range of Values
`byte`	Integer	1 byte	–128 to 127
`short`	Integer	2 bytes	–32,768 to 32,767
`int`	Integer	4 bytes	–2,147,483,648 to 2,147,483,647
`long`	Integer	8 bytes	–9,223,372,036,8547,75,808 to 9,223,372,036,854,775,807
`float`	Floating-point	4 bytes	$\pm 3.40282347 \times 10^{+38}$ to $\pm 1.40239846 \times 10^{-45}$
`double`	Floating-point	8 bytes	$\pm 1.79769313486231570 \times 10^{+308}$ to $\pm 4.94065645841246544 \times 10^{-324}$
`char`	Single character (Unicode)	2 bytes	All Unicode values from 0 to 65,535
`boolean`		1 bit	True or false

Assignment Statements

- An assignment statement is used to assign a value to a variable.

  `answer = 42;`
- The "equal sign" is called the *assignment operator.*
- We say, "The variable named **`answer`** is assigned a value of 42," or more simply, "**`answer`** is assigned 42."
- General rule: ***`Var = expression;`***

Building expressions: operators

- *Operators* can act on numbers and primitive data types
  - '+' adds two numbers
  - '*' multiplies two numbers
  - '-' subtracts two numbers
  - '/' divides two numbers
  - '==' tests for equality and returns a Boolean
- Operators can also act on other expressions – build arbitrarily complicated expressions (just like math)

Expressions

- A program is a sequence of statements
  - Well, it also needs a header
  - But the program body lists statements
- Simplest statement is an assignment
- A simple expression looks like:

var1 op var2;

  - Where 'var1' and 'var2' are variables
  - 'op' is any operator (e.g. +, -, *)

Variations on Expressions

- Variables can be re-used across and even within expressions:

```
int x = 7;
x = x + 1;
x = x + x;
```

- Right hand side is evaluated
- … and the result is assigned to the lhs variable

More variations on expressions

- The *right hand side* of an assignment can be any mathematical expression:

```
int y = x + (2 * z);
```

- When more than one operator appears
  - Parenthesis disambiguate
    - See above
  - Without parentheses, operator precedence rules apply
    - e.g., multiply before add, left before right
    - Better to rely on parentheses

Integers

- Numbers without fractional parts
  3, 47, -12
- Variables store integers with an assignment statement

```
int size = 7;
```

- Integer variables may be used like integer literals (i.e., number), e.g.,

```
size = size + 1;
```


Integer Arithmetic Operations

Symbol	Operation	Example	Evaluates to
+	Addition	45 + 5	50
-	Subtraction	657 – 57	600
*	Multiplication	7000 * 3	21000
/	Division	13 / 7	1
%	Remainder	13 % 7	6

Remainder reminder: % and /

- For any pair of integers, x and y, there always exist two unique integers, q and r, such that $x = qy + r$ and $0 \le r < y$
- q is called the quotient, r is the remainder when x is divided by y
- The operators, `%` and `/` compute them: `r = x % y` and `q = x / y`
- Java isn't mathematically correct for negative numbers

Integer Math

```
int i = 10/3;
```

- What's i = ? 3

```
int j = 10 % 3;
```

- What's j = ? 1

```
int k = 13 % 5 / 2;
```

- What's k = ? 1

Additional Integer Operators

- Self-assignment

```
int temperature = 32;
temperature = temperature + 10;
```

What is temperature? 42

- Increment

`cent++;` equivalent to `cent = cent + 1;`

- Decrement

`cent--;` equivalent to `cent = cent - 1;`

Specialized Assignment Operators

- Assignment operators can be combined with arithmetic operators including `-, *, /, %`.

  `amount = amount + 5;`

  can be written as

  `amount += 5;`

  yielding the same results.

Parentheses and Precedence

- Parentheses can communicate the order in which arithmetic operations are performed
- examples:

  `(cost + tax) * discount`

  `cost + (tax * discount)`

- Without parentheses, an expression is evaluated according to the *rules of precedence.*

Precedence Rules

- Figure 2.2 Precedence Rules

Highest Precedence

First: the unary operators +, -, !, ++, and --

Second: the binary arithmetic operators *, /, and %

Third: the binary arithmetic operators + and –

Lowest Precedence

Precedence Rules

- The *binary* arithmetic operators `*`, `/`, and `%`, have *lower precedence* than the *unary* operators `+`, `-`, `++`, `--`, and `!`, but have *higher precedence* than the binary arithmetic operators `+` and `-`.
- When binary operators have equal precedence, the operator on the left acts before the operator(s) on the right.

Sample Expressions

Ordinary Math	Java (Preferred Form)	Java (Parenthesized)
$rate^2 + delta$	`rate * rate + delta`	`(rate * rate) + delta`
$2(salary + bonus)$	`2 * (salary + bonus)`	`2 * (salary + bonus)`
$\frac{1}{time + 3mass}$	`1 / (time + 3 * mass)`	`1 / (time + (3 * mass))`
$\frac{a-7}{t+9v}$	`(a - 7) / (t + 9 * v)`	`(a - 7) / (t + (9 * v))`

Real Numbers

- Also called floating-point numbers
- Numbers with fractional parts

3.14159, 7.12, 9.0, 0.5e001, -16.3e+002

- Declared using the data type `double`

```
double pricePerPound = 3.99,
       taxRate       = 0.05,
       shippingCost  = 5.55;
double pctProfit = 12.997;
```


`double` Arithmetic Operations

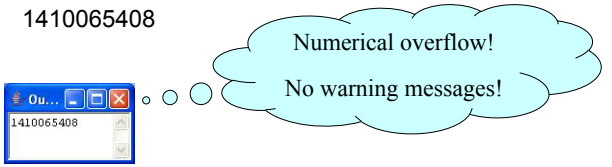
Symbol	Operation	Example
+	Addition	45.0 + 5.30
-	Subtraction	657.0 – 5.7
*	Multiplication	70.0 * 3.0
/	Division	96.0 / 2.0

Numbers in Java

- `int` is of fixed size;
  a value that is too large to be stored in an `int` variable will not match the mathematical value.
  - Example:

```
int x = 100000 * 100000;
out.println( x );
```

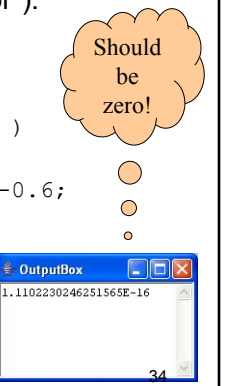
Will print: 1410065408



Numbers in Java

- It is not always possible to test `double` expressions for equality and obtain a correct result because of rounding errors (called "floating point error").

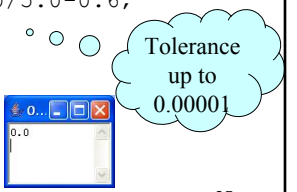
```
public class ProblemDoublePrecision
{
   public static void main( String[ ] args )
   {
      double val = 1.0/5.0+1.0/5.0+1.0/5.0-0.6;
      System.out.println( val );
   }
}
```



Numbers in Java

- How should you handle "floating point error"?
  - Test to see if the value is within a margin of error

```
public class CheckDoubleEquality
{
   public static void main( String[ ] args )
   {
      double val = 1.0/5.0+1.0/5.0+1.0/5.0-0.6;
      if ( Math.abs(val) < 0.00001 )
         val = 0;
      System.out.println( val );
   }
}
```



Assignment Compatibilities

- Java is strongly typed.
  - You can't, for example, assign a floating point value to a variable declared to store an integer.
- But for convenience, some conversions between numbers are possible …

  `doubleVariable = 7;`
- … is possible even if doubleVariable is of type double, for example.

Assignment Compatibilities

- A value of each following type can be assigned to a variable of type to the right:

  byte → short → int → long → float → double
  - but not to a variable of any type to the left.
- You can assign a value of type char to a variable of type int.
- … except through type casting

Type Casting

- A type cast changes the value of a variable from the declared type to some other type.
- For example,

```
double distance;
distance = 9.0;
int points;
points = (int)distance;
```

- Illegal without (int)

Type Casting

- The value of (int)distance is 9,
- The value of distance, both before and after the cast, is 9.0.
- The fractional part (to the right of the decimal point) is truncated rather than rounded.

Mixing Numeric Data Types

- *Widening conversion* Java will automatically convert `int` expressions to `double` values without loss of information
  - `int i = 5;`

    `double x = i + 10.5;` — Arithmetic promotion of i to largest data type double
  - `double y = i;` — assignment promotion of i
- *Narrowing conversion* To convert `double` expressions to `int` requires a *typecasting* operation and truncation will occur

  `i = (int) (10.3 * 2);` — i = **20** : the .6 is truncated
- To *round-up* instead of truncating add 0.5

  `i = (int) (10.3 * x + 0.5);`

Characters

- Any key you type on the keyboard generates a character which may or may not be displayed on the screen (e.g., *nonprinting* characters)
- Characters are a primitive type in Java and are not equivalent to Strings
- Examples

```
char vitamin = 'A',
     chromosome = 'y',
     middleInitial = 'N';
```


Important Literal Characters

'A', … ,'Z'	Uppercase letters
'a', … ,'z'	Lowercase letters
'0', … , '9'	Digits
'.', ',', '!','"',etc.	Punctuation Marks
' '	Blank
'\n'	New line
'\t'	Tab
'\\'	Backslash
'\''	Single Right Quote

The other meta-type: Classes

- A *primitive* data type is indivisible
  - They have no meaningful subparts
  - The primitives are defined by the language
    - int, char, double, etc.
- A *class* is a data type that contains many bits of information
  - For example, Strings (many primitive chars)
  - Many classes defined by the language
    - You can also define new ones…

Classes

- Classes have data and methods
  - The data may be primitives or collections of primitives, other even classes.
  - Methods are used instead of operators
- The period ('.') accesses methods of a class:

```
String greeting = "hello";
char c = greeting.charAt(0);
// c now equals 'h'
```

More about Strings

- `String` is defined in the `java.lang` package
  - *The java.lang package is automatically included in all programs, so you do not need to import it.*
- String literals are defined in double-quotes;
  - *Examples:*

```
String t1 = "To be ";
String t2 = "or not to be";
System.out.println(t1.concat(t2));
// prints To be or not to be
```


String Methods

Name	Description
`int length()`	Returns the length of this string
`int indexOf(String s)`	Returns the index within the string of the first occurrence of the string s.
`String substring (int beginx, int endx)`	Returns the substring beginning at index beginx and ending at index endx-1
`String toUpperCase()`	Converts all characters of the string to uppercase
`String concat(String s)`	Concatenates the string s to the end of the original string
`char charAt(int index)`	Returns the character at the index, which must be between 0 and length of string - 1

Syntax: primitives vs classes

- Operators act on primitive variables
  - Examples: +. -, *, %
  - Standard math in-fix notation
    - x + y;
    - y / 7;
- Methods act on class variables
  - Example: length()
  - Notation: class.method(arguments)
    - String s1 = "foo";
    - int x = s1.length();

String Method Examples

```
import java.util.Scanner;

// Simple string program in Java

public class SimpleString {

    public static void main(String[] args) {

        // Keyboard input example
        String string1;
        Scanner keyboard = new Scanner(System.in);
        System.out.print("String: ");
        string1 = keyboard.nextLine();
        System.out.println(string1);

        // Using String methods
        String string2 = "Here is a test string";
        System.out.println(string2.charAt(2));       // prints "r"
        System.out.println(string2.indexOf("s"));    // prints 6
        System.out.println(string2.indexOf("x"));    // prints -1
        System.out.println(string2.length());        // prints 21
        System.out.println(string2.substring(8,14)); // prints 'a test'
    }
}
```

Object Examples

- Scanner instance is an object (not primitive)
- Methods for the Scanner class include
  - nextInt ← returns next sequence as integer
  - nextDouble ← returns next sequence as double
  - next ← returns next sequence of chars
    - read until the next whitespace (spaces, tabs, end of line)
  - nextLine ← returns next line up until enter key
    - reads

Input/Output (I/O)

Q: Without a graphical user interface (GUI), how does a program communicate with user?

A: Must use the console for output, keyboard for input.

- Console Output
- Keyboard Input

Console

- We've seen several examples of console output already.
- `System.out` is the output object in Java.
- Question: Is there a `System.in` for input?
- `println()` is one of the methods available to the `System.out` object.
- `print()` is another method, which is identical to println() without line termination.

Screen Output

- The concatenation operator (+) is useful when everything does not fit on one line.

```
System.out.println(
   "Lucky number = " + 13 +
   "Secret number = " + number);
```

- Beware of dual use of + for addition and concatenation:

```
System.out.println("5+3=" + 5 + 3);
```

Keyboard Input

- Java has reasonable facilities for handling keyboard input.
- These facilities are provided by the Scanner class in the java.util package.
  - A package is a library of classes.

Using the Scanner Class

- Near the beginning of your program, insert

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

- Create an object of the `Scanner` class

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

- Read data (an `int` or a `double`, for example)

```
int n1 = keyboard.nextInt();
double d1 = keyboard.nextDouble();
```


More Scanner Class

- From util package

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

System.in refers to the keyboard

- Create a new instance:

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

- Input *(depends on data type reading in)*
  - `String input = in.next( );`
  - `String line = in.nextLine( );`
  - `int intVal = in.nextInt( );`
  - `double dblVal = in.nextDouble( );`

Some Scanner Class Methods

Scannner_Object_Name`.next()`
Returns the `String` value consisting of the next keyboard characters up to, but not including, the first delimiter character. The default delimiters are whitespace characters.

Scannner_Object_Name`.nextLine()`
Reads the rest of the current keyboard input line and returns the characters read as a value of type `String`. Note that the line terminator `'\n'` is read and discarded; it is not included in the string returned.

Scannner_Object_Name`.nextInt()`
Returns the next keyboard input as a value of type `int`.

Scannner_Object_Name`.nextDouble()`
Returns the next keyboard input as a value of type `double`.

Scannner_Object_Name`.nextFloat()`
Returns the next keyboard input as a value of type `float`.

Reading Integers

```
import java.util.*;

public class getInput
{
    public static void main( String[ ] args )
    {
        Scanner in;
        int intVal;

        in = new Scanner( System.in );
        System.out.println("Enter an integer: ");
        intVal = in.nextInt( );

        System.out.println( intVal );
    }
}
```


Reading Strings

```
import java.util.*;

public class getStringInput
{
    public static void main( String[ ] args )
    {
        Scanner in;
        String name;

        in = new Scanner( System.in );
        System.out.println("Enter your name: ");
        name = in.next( );

        System.out.println( name );
    }
}
```


`nextLine()` Method Caution

- The `nextLine()` method reads
  - The remainder of the current line,
  - Even if it is empty.

nextLine Method Caution

- Example – given following declaration.

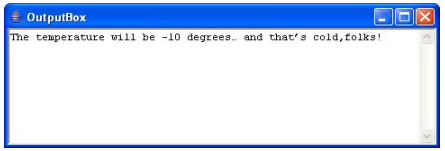
```
int n;
String s1, s2;
n = scan();
s1 = scan.nextLine();
s2 = scan.nextLine();
```

- Assume input shown

```
42
next line
another line
```

  - n is set to 42
  - but s1 is set to the empty string
  - and s2 is set to “next line”

Printing Integers




```
public class Forecast
{
    public static void main (String args[])
    {
     System.out.print("The temperature will be ");
     System.out.print(-10);
     System.out.print(" degrees…");
     System.out.println(" and that's cold,folks!");
    }
}
```

What happens if you use println each time?

Formatting Output with printf

- Java has a method named `printf()` that applies a specific format to output.
- The `printf()` method similar to the `print()/println()` methods.
- System.out.printf can have any number of arguments!
  - The first argument contains one or more format specifiers for the remaining arguments
  - All the arguments except the first are values to be output to the screen

`printf` Format Specifier

- The following code

```
double price = 19.8;
System.out.print("$");
System.out.printf("%6.2f", price);
System.out.println(" each");
```

will output the line

```
$ 19.80 each
```

- The format specifier `"%6.2f"` is interpreted as follows:
  - Display up to 6 right-justified characters (*field width* is `6`)
  - Display exactly 2 digits after the decimal (*precision* is `2`)
  - Display a floating point number (*conversion character* is `f`)

Multiple arguments with `printf`

- The following code contains a `printf` statement having three arguments

```
double price = 19.8;
String name = "apple";
System.out.printf("$%6.2f per %s.", price, name);
System.out.println();
System.out.println("Wow");
```

will output

```
$ 19.80 per apple.
Wow
```

  - Note that the first argument is a format string containing two format specifiers (`%6.2f` and `%s`)
  - These format specifiers match up with the two arguments that follow (`price` and `name`)

Line Breaks with `printf`

- Line breaks can be included in a format string using `%n` or `\n`
- The code

```
System.out.printf("Hello%nWorld\n!");
System.out.println("Wow");
```

will output

```
Hello
World
!Wow
```


Format Specifiers for `System.out.printf`

Display 2.1 Format Specifiers for `System.out.printf`

CONVERSION CHARACTER	TYPE OF OUTPUT	EXAMPLES
d	Decimal (ordinary) integer	%5d %d
f	Fixed-point (everyday notation) floating point	%6.2f %f
e	E-notation floating point	%8.3e %e
g	General floating point (Java decides whether to use E-notation or not)	%8.3g %g
s	String	%12s %s
c	Character	%2c %c

The `printf` Method (Part 1 of 3)

Display 2.2 The `printf` Method

```
public class PrintfDemo
{
    public static void main(String[] args)
    {
        String aString = "abc";

        System.out.println("String output:");
        System.out.println("START1234567890");
        System.out.printf("START%sEND %n", aString);
        System.out.printf("START%4sEND %n", aString);
        System.out.printf("START%2sEND %n", aString);
        System.out.println();
```

(continued)

The `printf` Method (Part 2 of 3)

Display 2.2 The `printf` Method

```
        char oneChracter = 'Z';

        System.out.println("Character output:");
        System.out.println("START1234567890");
        System.out.printf("START%cEND %n", oneCharacter);
        System.out.printf("START%4cEND %n", oneCharacter);
        System.out.println();

        double d = 12345.123456789;

        System.out.println("Floating-point output:");
        System.out.println("START1234567890");
        System.out.printf("START%fEND %n", d);
        System.out.printf("START%.4fEND %n", d);
        System.out.printf("START%.2fEND %n", d);
        System.out.printf("START%12.4fEND %n", d);
        System.out.printf("START%eEND %n", d);
        System.out.printf("START%12.5eEND %n", d);
    }
}
```

(continued)

The `printf` Method (Part 3 of 3)

Display 2.2 **The printf Method**

Sample Dialogue

```
String output:
START1234567890
STARTabcEnd
START abcEnd
STARTabcEnd

Character output:
START1234567890
STARTZEND
START   ZEND

Floating-point output:
START1234567890
START12345.123457END
START12345.1235END
START12345.12END
START  12345.1235END
START1.234512e+04END
START 1.23451e+04END
```

The value is always output. If the specified field width is too small, extra space is taken.

Note that the output is rounded, not truncated, when digits are discarded.

Legacy Code

- Code that is "old fashioned" but too expensive to replace is called *legacy code*
- Sometimes legacy code is translated into a more modern language
- The Java method `printf` is just like a C language function of the same name
- This was done intentionally to make it easier to translate C code into Java

What could go wrong?

- If you mis-type a variable name or a data type…
  - When you try to compile & run it in Eclipse
    1. Eclipse will tell you there was an error
    2. The editor will put a red ‘x’ at the left of the line with the error
  - This is an example of a compile-time error

What else could go wrong?

- You can specify an illegal operation
  - E.g. try to divide a string by a string
  - Again, a compile-time error with a red ‘x’
- You can forget a ; or a }
  - Same as above

More Errors

- Capitalization errors
  - Java is case sensitive, identifier names must use the same capitalization rules each time
- Logic Errors
  - Program appears to run correctly, but on closer inspection the wrong output is displayed

Debugging Hints

- Let Eclipse help you!
  - Gives suggestions on methods to use
  - Provides warning and error messages as you type… even provides suggestions of how to fix the problem.
- Add debugging statements to check the computation `System.out.println(…);`