Chapter 2: Data and Expressions

CS 121

Department of Computer Science
College of Engineering
Boise State University

November 2, 2015

Chapter 2

- Part 1: Data Types

  Go to part 1

- Part 2: Expressions and `Scanner`

  Go to part 2

Part 1: Data Types

- What is a data type?
- Character Strings
  - Concatenation
  - Escape Sequences
- Java Primitive Data Types
- Declaring and Using Variables

Go to index.

What is a data type?

- Programs represent all kinds of data.
- What types of data might the following programs need to represent?
  - A calculator program.
  - A word processor.
  - An address book.
- A data type is a classification identifying various types of data, such as real, integer, Boolean, words, etc.

Character Strings

- A sequence of characters can be represented as a string literal by putting double quotes around it.
- `"This is a string literal." "So is this."`
- What about the string literal? `""`
- A character string is an object in Java, defined by the `String` class.
- Every string literal represents a `String` object.
- See javadoc for String here: `http://docs.oracle.com/javase/8/docs/api/java/lang/String.html`.

Printing Strings

- The `System.out` object represents a destination (the monitor) to which we can send output.
- We can invoke the `println` and `print` methods of the `System.out` object to print a character string.
  - `println` – prints a new line character ('\n') after the string.
  - `print` – does NOT print a new line character ('\n') after the string.
- Example: `Countdown.java`

String Concatenation (1)

- The string concatenation operator (+) appends one string to the end of another.

```
"Peanut butter " + "and jelly"
```

- Allows strings to be broken across multiple lines.

```
"If this was a long string, we may want it on " +
"two lines so we can see it more easily"
```

- Also used to append numbers to a string.

```
"We will have " + 8 + " quizzes this semester."
```

String Concatenation (2)

- The + operator is also used for addition.
- The function it performs depends on the context.
  - String concatenation
    - Both operands are strings.
    - One operand is a string and one is a number.
  - Addition
    - Both operands are numeric.
  - Example: `Addition.java`
- Precedence: evaluated left to right, but can use parenthesis to force order (more about this later).

Escape Sequences

- What if we wanted to actually print the " character??
- Let's try it.

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

- Our compiler is confused! Do you know why?
- We can fix it with an escape sequence – a series of characters that represents a special character.
- Begins with a backslash character (\).

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

Some Java Escape Sequences

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

Using Java Escape Sequences

- Example: `BlankOrDark.java`
- Example: `CarriageReturnDemo.java` (must run from command-line)

Primitive Data Types

- There are 8 primitive data types in Java (varies in other languages)
- Integers
  - `byte`, `short`, `int`, `long`
- Floating point types
  - `float`, `double`
- Characters
  - `char`
- Boolean values (true/false)
  - `boolean`

Numeric Types

Type	Space (#bits)	Minimum value	Maximum Value
`byte`	8	-128	127
`short`	16	-32768	32767
`int`	32	-2147483648	2147483647
`long`	64	-9223372036854775808	9223372036854775807
`float`	32	1.4E-45	3.4028235E38
`double`	64	4.9E-324	1.79769313486231 57E308

- `float` has 6-9 significant digits
- `double` has 15-17 significant digits

Initializing Numeric variable

- A decimal literal value is an `int` by default. To write a long literal value, we have to use the `L` suffix.

```
int answer = 42;
long neuronsInBrain = 100000000000L;
```

- A floating point literal value is `double` by default. To write a `float` literal value, we have to use the `F` suffix.

```
double delta = 453.234343443;
float ratio = 0.2363F;
```

Characters

- A `char` stores a *single* character delimited by single quotes.

```
char topGrade = 'A';
char comma = ',';
char tab = '\t';
```

- A `char` variable in Java can store any character from the Unicode character set.
  - Each character corresponds to a unique 16-bit number.
  - The `Character` class supports the full Unicode Standard.
- English speakers typically use characters from the ASCII character set.
  - Older and smaller subset of Unicode (only 7-bits per character).
- See Apendix C on page 951 of your textbook.
- http://en.wikipedia.org/wiki/Unicode
- http://en.wikipedia.org/wiki/ASCII

Booleans

- Only two valid values for the `boolean` type: `true` or `false`.
- Reserved words `true` and `false`.

```
boolean done = false;
```

- Commonly used to represent two states (e.g. on/off)

Review

What are the 8 primitive data types available in Java?

- Integers (`byte`, `short`, `int`, `long`)
- Floating Point (`float`, `double`)
- Character (`char`)
- Boolean (`boolean`)

How do we represent a sequence of characters?

- Strings (the `String` object).

Declaring and Using Variables

- We know we can represent different types of data in our programs using the data types we just discussed, but we need a way to keep track of all of this data.
- Variables allow us to define and reference the data we use in our programs.

Java Identifiers

- Identifiers are words a programmer uses in a program.
  - Consists of a combination of A-Z, a-z, 0-9, _, and $
  - Can't begin with digit.
  - Case sensitive.
    - `Total`, `total`, and `TOTAL` are *different*
- Good practice to use different case style for different types of identifiers.
  - title case for class names – `Lincoln`, `HelloClass`
  - camel case for variables – `count`, `nextCount`
  - upper case for constants – `MAXIMUM`, `MINIMUM`

Reserved Words

- Reserved words are special identifiers that have pre-defined meaning. They can't be used in any other way.
- Some examples – `public`, `static`, `void`, `class`
- See page 7 (Chapter 1) in textbook for full list of reserved words.

Variables

- A variable is just a name for a location in memory.
- Variable names are identifiers. They must be unique.
- Variables must be declared by specifying a name and the type of information it will hold.

```
String name;
int radius, area, circumference;
```

- When a variable is used in a program, the current value is used.

Assignment

- An assignment statement changes the value of a variable.
- The assignment operator is the equals sign (=).

```
int radius;
radius = 10;
```

- The value on the right-hand side is stored in the variable on the left.
- The previous value in `radius` is overwritten.
- Variables can also be initialized when they are declared.

```
int count = 0;
```

- The type of the right-hand side must be compatible with the type of the variable.

Assignment

- The right-hand side can be an expression.
- The expression will be evaluated *first* and *then* stored in the variable.

```
radius = 10;
radius = radius * 2; // double the radius
```

- What is the new value of `radius`? 20

Constants

- A constant is an identifier (similar to a variable) that holds the *same* value during its entire existence.
- It is constant, not variable.
- The compiler will issue an error if you try to change the value of a constant.
- In Java, we use the `final` modifier to declare a constant.
- We typically use all caps to name constants.

```
final int MAX_RADIUS = 1000;
```

Why do we need constants?

- Readability – give meaning to arbitrary literals.
- Program maintenance – only need to change value once.
- Program protection – establishes that a value should not change; less chance for error.

In-class Exercises

- EX 2.2. What output is produced by the following code fragment? Explain.

```
System.out.print("Here we go!");
System.out.println("12345");
System.out.print("Another.");
System.out.println("");
System.out.println("All done.");
```

- EX 2.4. What output is produced by the following statement? Explain.

```
System.out.println("50 plus 25 is " + 50 + 25);
```

- PP 2.1. Create a revised version of the Lincoln application from Chapter 1 such that quotes appear around the quotation.

Part 2: Expressions and `Scanner`

- Expressions
- Data conversions
- The `Scanner` class for interactive programs

Go to index.

In-class Exercises

- Which data type would you use to represent each of the following items?
  - The name of a restaurant.
  - The maximum number of occupants a restaurant can hold.
  - The current number of occupants.
  - The price of a meal.
  - Whether or not the restaurant is open.
- Write a variable declaration for each of the above items. Make sure to give your variables descriptive names.

Expressions

- An expression is a combination of one or more operators and operands.
- We focus on arithmetic expressions that produce numeric results.

Arithmetic Expressions

- Arithmetic expressions use the arithmetic operators.

Addition	+
Subtraction	-
Multiplication	*
Division	/
Remainder (modulo)	%

Arithmetic Expressions and Data Types

- If *any one* of the operands used by an arithmetic operator is *floating point* (`float` or `double`), then the result will be a floating point.
- For example:

```
int radius = 10;
final double PI = 3.14159265358979323;
double area = PI * radius * radius;
```

- If *both* operands used by an arithmetic operator are *floating point*, then the result will be a floating point
- If *both* operands used by an arithmetic operator are *integer*, then the result will be an integer. Be careful!!

Division and Data Types

- If both operands of the division operator are integers, then the result will be an integer.
- This means we **lose the fractional part of the result**.
- For example, let's assume we want to divide a wall into equal sections.

```
int length = 15;
int sections = 2;
double newLength = length / sections;
```

- Let's try this.
- How can we fix it?
- Data conversion – we'll get to this soon.

Remainder Operator (modulo)

- Given two positive numbers, *a* (the dividend) and *b* (the divisor), *a* % *n* (*a* mod *n*) is the remainder of the Euclidean division of *a* by *n*.

```
14 / 3  ==  4          14 % 3  ==  2
 8 / 12 ==  0           8 % 12 ==  8
10 / 2  ==  5          10 % 2  ==  0
 7 / 6  ==  1           7 % 6  ==  1
 9 / 0  ==  error       9 % 0  ==  error
```

Remainder Operator (modulo)

- Typically used to determine if a number is odd or even.
- How?

Operator Precedence (Order of Operations)

- Just like in Mathematics, operators can be combined into complex expressions.

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

- Operators have well-defined precedence to determine order of evaluation.

```
result = total + count / max - offset;
         4         2         1       3
```

- Expressions are evaluated from left to right in order of operator precedence.

Operator Precedence

Precedence	Operator	Operation	Association
0	()	parenthesis	L to R
1	+	unary plus	R to L
	-	unary minus	
2	*	multiplication	L to R
	/	division	
	%	modulo (remainder)	
3	+	addition	L to R
	-	subtraction	
	+	string concatenation	
4	=	assignment	R to L

See the full precedence table in Figure D.1 on page 956 of your textbook.

In-Class Exercise

- Determine the order of evaluation in the following expressions.

```
1)  a + b + c + d + e

2)  a + b * c - d / e

3)  a / (b + c) - d % e

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

In-Class Exercise

- Determine the order of evaluation in the following expressions.

```
1)  a  +  b  +  c  +  d  +  e
       1     2     3     4

2)  a  +  b  *  c  -  d  /  e
       3     1     4     2

3)  a  /  (b  +  c)  -  d  %  e
       2     1     4     3

4)  a  /  (b  *  (c  +  (d  -  e)))
       4     3     2     1
```

Order of Evaluations and Integer Division

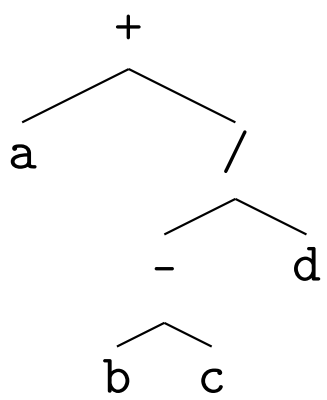
- Expressions are evaluated from left to right in order of operator precedence.
- This order can change the results of an expression, especially where possible integer division is involved, which can easily lead to bugs in code.

```
final double PI = 3.14159;
double radiusCubed = 1.0;
double volume1 = 4 / 3 * PI * radiusCubed;
double volume2 = PI * radiusCubed * 4 / 3;
```

- Does `volume1` equal `volume2`?
- Example: Volume.java

Expression Trees

- Evaluation order of an expression can also be shown using an expression tree.
- The operators *lower* in the tree have *higher* precedence for that expression.
  - `a + (b - c) / d`



Assignment Operator

- The assignment operator has the *lowest* operator precedence.
- The *entire* right-hand side expression is evaluated first, then the result is stored in the original variable.
- It is common for the right hand side and left hand sides of an assignment statement to contain the same variable.
  - `count = count + 1;`

Increment and Decrement Operators

- The increment operator (++) adds one to its operand.
  - The following statements produce the same result.

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

- The decrement operator (--) subtracts one from its operand.
  - The following statements produce the same result.

```
count--;
count = count - 1;
```

- The increment ++ and decrement - operators have the same level of precedence as the unary + and unary - operators.

Postfix vs. Prefix

- The increment and decrement operators can be applied in postfix form

```
count++;   count--;
```

  or prefix form

```
++count;   --count;
```

- When used as part of a larger expression, the two can have different effects. *Use with care!!*

Assignment Operators

- Java provides assignment operators to simplify expressions where we perform an operation on an expression then store the result back into that variable.
- Consider the following expression.

```
num = num + count;
```

- We can simplify this using the addition assignment operator.

```
num += count;
```

- Java provides the following assignment operators.
  - += (string concatenation or addition), -=, *=, /=, %=

Data Conversion

- Sometimes we need to convert from one data type to another (e.g. `double` to `int`).
- These conversions *do not change the type of a variable*, they just convert it temporarily as part of a computation.
- Widening conversions. Safest. Go from small data type to large one.
  - e.g. `short` to `int`, `int` to `double`
- Narrowing conversions. Not so safe. Go from large data type to smaller one. Must be used carefully as we can lose information!
  - e.g. `int` to `short`, `double` to `int`
- By default, Java will not allow narrowing conversions unless we force it (shown later)
  - `int count = 3.14 ; //won't compile!`

Data Conversions

- Assignment conversion.
- Promotion.
- Casting.

Assignment Conversion

- Assignment conversion occurs when one type is assigned to a variable of another.
- Only *widening conversions* can happen via assignment.
- For example:

```
double totalCost;
int dollars;
totalCost = dollars;
```

- The *value* stored in `dollars` is converted to a `double` before it is assigned to the `totalCost` variable.
- The `dollars` variable and the value stored in it are still `int` after the assignment.

Promotion

- Promotion happens automatically when operators in expressions convert their operands.
- For example:

```
double sum;
int count;
double result = sum / count;
```

- The value of `count` is converted to a `double` before the division occurs.
- Note that a widening conversion also occurs when the result is assigned to `result`.

Casting

- Casting is the most powerful and potentially dangerous conversion technique.
- Explicitly perform *narrowing* and *widening* conversions.
- Recall our example from earlier:

```
int length = 15, sections = 2;
double newLength = length / sections;
```

- Recall: *If both operands of the division operator are integers, then the result will be an integer. If either or both operands used by an arithmetic operator are floating point, then the result will be a floating point.*
- By casting one of the operands (`length` in this case), we get the desired result

```
double newLength = ((double) length) / sections;
```

In-Class Exercise

Will the following program produce an accurate conversion (why or why not)?

```
/**
 * Computes the Fahrenheit equivalent of a specific
 * Celsius value using the formula:
 *                F = (9/5) * C + 32.
 */
public class TempConverter
{
    public static void main (String[] args)
    {
        final int BASE = 32;

        double fahrenheitTemp;
        int celsiusTemp = 24;  // value to convert

        fahrenheitTemp = celsiusTemp * 9 / 5 + BASE;

        System.out.println ("Celsius Temperature: " +
   celsiusTemp);
        System.out.println ("Fahrenheit Equivalent: "
   + fahrenheitTemp);
    }
}
```

1. Yes.
2. Sometimes.
3. Nope.
4. I have no idea.

The Scanner class

- Typically, we want our programs to interact with our users.
- The `Scanner` class is part of the `java.util` class library. It must be imported.

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

- It provides methods for reading input values of various types.
- A `Scanner` object can read input from various sources (e.g. keyboard, file)
- See Java 8 API docs: http://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html

Using the Scanner

- Create a new `Scanner` object that reads from the keyboard.

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

- The `new` operator creates a new `Scanner` object.
- The `System.in` object represents keyboard input.
- After the object is created, we can invoke various input methods it provides.

Example

- Example: Convert `TempConverter.java` to interactive program.
- Example: `Echo.java`

Input Tokens

- By default, white space is used to separate input elements (called tokens).
- White space includes
  - Space characters (' ')
  - Tab characters ('\t')
  - New line characters ('\n' and '\r')
- The `next`, `nextInt`, `nextDouble`, etc. methods of the `Scanner` class read and return the next input tokens.
- See `Scanner` documentation for more details.

Example

- Example: `GasMileage.java`

Exercises

- **Recommended Homework**:
  - Exercises: EX 2.5, 2.7, 2.8, 2.9, 2.10 (a, b, c, d), 2.11 (e, f, g, i, j).
  - Projects: PP 2.3, 2.4, 2.8.
- Browse Chapter 3 of textbook.