CIS2571 – Intro to Java

Chapter 2 → Elementary Programming

Information consolidated from a variety of textbook and online resources

Topic Objectives

- Know the steps for writing simple programs
- Understand the identifier naming requirements
- Know how to use data types
  - Numeric primitive
  - char and String
- Understand the difference between variables and constants
- Identify the various operators used in expressions and their order of evaluation
- Know how to convert between different data types
- Get input from
  - The console
  - GUI dialog box

Writing Simple Programs

- Involves designing algorithms and translation into code
- Java programs
  - Begin with class declaration
  - Must have a **main** method
  - Use **identifiers** to name program items
    - Variables, constants, methods, etc.
  - Statements execute actions and follow defined syntax

Identifiers

- Names of items contained within a program
- Must obey naming rules:
  - Sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($).
  - Must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
  - Cannot be a reserved word. (See Appendix A, "Java Keywords," for a list of reserved words).
    - An identifier cannot be true, false, or null.
  - Can be of any length.

Naming Conventions

- Choose meaningful and descriptive names
- Use case sensitivity and abbreviations for multiple words
  - Start variables and methods with lowercase

    `radius` and `showInputDialog`
  - Start class names with uppercase

    `ComputeArea` and `JOptionPane`
  - Capitalize all letters in constants; use underscore to separate words

    `PI` and `MAX_VALUE`

Data Types

- Data type has range of values and possible operations
  - Compiler allocates appropriate memory
- Primitive numeric data types

Name	Range	Storage Size
byte	-2^7 (-128) to 2^7-1 (127)	8-bit signed
short	-2^{15} (-32768) to $2^{15}-1$ (32767)	16-bit signed
int	-2^{31} (-2147483648) to $2^{31}-1$ (2147483647)	32-bit signed
long	-2^{63} to $2^{63}-1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
float	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
double	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754

Variables

- Used to hold values in memory
  - On left hand side of assignment statements
  - Name to access memory location
  - Data type to determine size, value types, and range of operations
  - Variable initialization allowed
  - Multiple declarations allowed
  - Declaration format:

```
datatype variableName;
datatype variableName = value;
datatype variableName1 = value1,
         variableName2 = value2;
```

Character Data Type

- Character data type used to represent a single character
- Character literal enclosed in single quotation marks

```
char letter = 'A';
```

- Encoding maps a character to its binary representation
  - 16-bit Unicode takes two bytes and is preceded by \u

```
char letter = '\u0041';
```

- Escape sequences represent special characters:

Character	Name	Unicode
\b	Backspace	\u0008
\t	Tab	\u0009
\n	Linefeed	\u000A
\f	Formfeed	\u000C
\r	Carriage Return	\u000D
\\	Backslash	\u005C
\'	Single Quote	\u0027
\"	Double Quote	\u0022

String Data Type

- String data type used to represent multiple characters
- Is pre-defined class in Java library; not a primitive data type
- Uses reference type for variable
- String literal enclosed in double quotation marks

```
String message = "Welcome to Java";
```

- Plus sign used for concatenation (also +=)

```
String message = "Welcome " + "to" + " Java";
String message = "Chapter" + 2;
String message = "Supplement" + 'B';
```

Constants

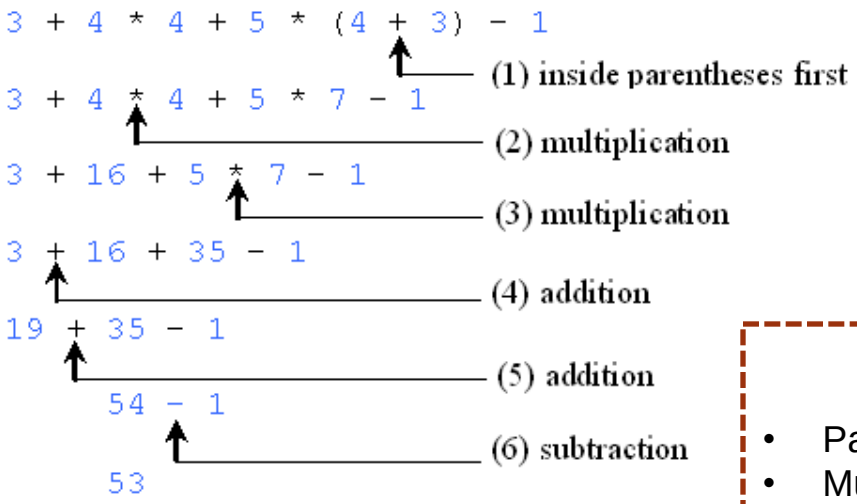
- Constants represent data that doesn't change
  - Named constants use **final** keyword in declaration

```
final datatype CONSTANTNAME = value;
```

  - Literal constants displayed directly as value
    - Integer literal → default is **int**
      - Use **L** or **l** to denote integer literal of type **long**
        - `21L`
      - Use leading **0** for octal and leading **0x** for hexadecimal
    - Floating point literal → default is **double**
      - Use **F** or **f** to denote floating point literal of type **float**
        - `100.2F`
      - Use **D** or **d** to denote floating point literal of type **double**
        - `100.2D`
      - Use **E** or **e** to denote scientific notation
        - **`1.23E+2`**

Expressions

- Expression represents computation involving values, variables and operators that evaluate to single value
- Result of a Java expression and its corresponding arithmetic expression are the same



Order of evaluation

- Parentheses
- Multiplication/division/remainder (*left to right*)
- Addition/subtraction (*left to right*)

Operators

- Numeric
  - + for addition and string concatenation
  - Integer versus decimal division
  - Remainder useful for determining odd or even numbers
  - Floating point numbers are approximate
- Assignment operator uses =

  `variable = expression;`
- Shortcut Assignment

Name	Meaning	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

Operator	Example	Equivalent
+=	i += 8	i = i + 8
-=	f -= 8.0	f = f - 8.0
*=	i *= 8	i = i * 8
/=	i /= 8	i = i / 8
%=	i %= 8	i = i % 8

Operators

- Increment and Decrement

Operator	Name	Description
++var	preincrement	The expression (++var) increments var by 1 and evaluates to the new value in var after the increment.
var++	postincrement	The expression (var++) evaluates to the original value in var and increments var by 1.
--var	predecrement	The expression (--var) decrements var by 1 and evaluates to the new value in var after the decrement.
var--	postdecrement	The expression (var--) evaluates to the original value in var and decrements var by 1.

```
int i = 10;
int newNum = 10 * i++;
--OR--
int i = 10;
int newNum = 10 * i;
i = i + 1;
```

```
int i = 10;
int newNum = 10 * (++i);
--OR--
int i = 10;
i = i + 1;
int newNum = 10 * i;
```

Numeric Type Conversion

- Can assign value to variable with larger range of values
  - When operands are of different type in a binary operation, expression value automatically promoted to data type of larger range
- **Cannot** assign value to variable with smaller range of values **unless type casting** is used
  - Widening
    - Casting variable to data type with larger range
  - Narrowing
    - Casting variable to data type with smaller range
  - Does not change variable being cast, only value

```
System.out.println((double)1 / 2);
System.out.println((int)1.7);
```

Numeric Type Conversion

- Target type in parentheses followed by variable's name or value

```
double d = 4.5;
int i = (int)d;
```

- **char** data type can be cast into any numeric type, and vice versa
  - For **int**, only lower 16 bits are stored
  - **char** operand is automatically cast to number if other operand is number

```
char ch = (char)0XAB0041;  // lower 16 bits hex code 0041 is assigned
System.out.println(ch);    // ch is character 'A' (65 ASCII decimal)

int j = 2 + 'a';                    // (int)'a' is 97
System.out.println("j is " + j);  // j is 99
```

Input From Console

- **`System.in`** refers to the standard input device (i.e. keyboard)
- Console input not directly supported in Java
  - Use **`Scanner`** class for console input
- Scanner methods to read various types of input:

Method	Description
`nextByte()`	reads integer of byte type
`nextShort()`	reads integer of short type
`nextInt()`	reads integer of int type
`nextLong()`	reads integer of long type
`nextFloat()`	reads number of float type
`nextDouble()`	reads number of double type
`next()`	reads string that ends before whitespace character
`nextLine()`	reads line of text that ends with Enter key

Input From Console

- Example to read floating point number:

```
System.out.print("Enter a double value: ");
Scanner input = new Scanner(System.in);
double d = input.nextDouble();
```

- Example to read 3 strings ending in whitespace:

```
System.out.print("Enter three strings: ");
Scanner input = new Scanner(System.in);
String s1 = input.next();
String s2 = input.next();
String s3 = input.next();
```

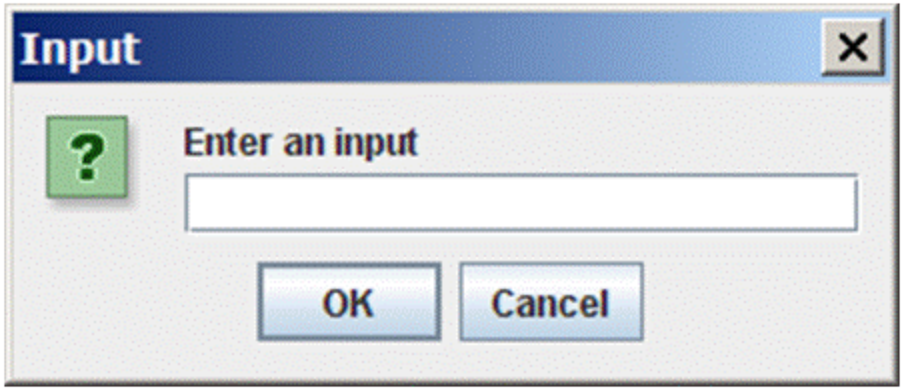
- Example to read string ending with *Enter* key:

```
System.out.print("Enter a string: ");
Scanner input = new Scanner(System.in);
String s = input.nextLine();
```

Input from GUI Input Dialogs

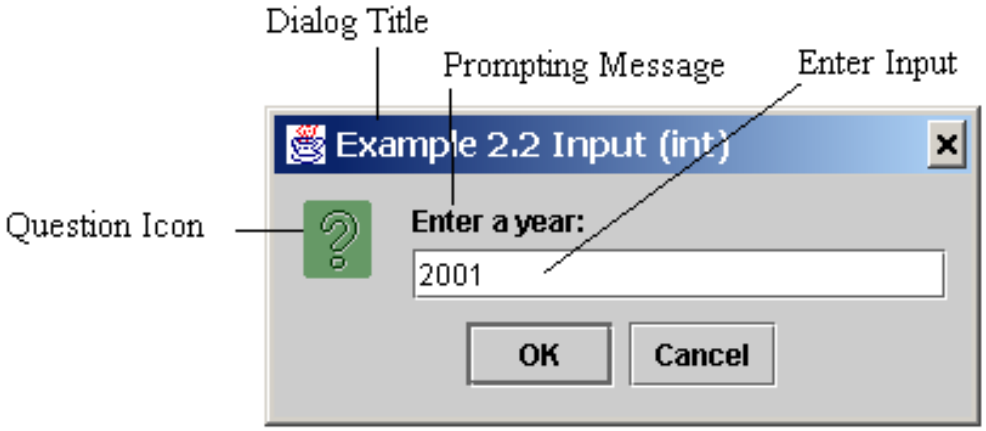
- showInputDialog is pre-defined method in the JOptionPane class used to get input from a dialog box

```
import javax.swing.JOptionPane;
public class WelcomeInMessageDialogBox {
   public static void main(String[] args) {
      String string = JOptionPane.showInputDialog(null, "Enter an Input");
   }
}
```



Input from GUI Input Dialogs

```
String string = JOptionPane.showInputDialog(
   null, "Prompting Message",  "Dialog Title",
   JOptionPane.QUESTION_MESSAGE);
```



Converting Strings to Number

- Input returned from input dialog box is string.
- Convert string to numeric value types using static class methods (in package java.lang):

```
byte byteValue = Byte.parseByte(byteString);
short shortValue = Short.parseShort(shortString);
int intValue = Integer.parseInt(intString);
long longValue = Long.parseLong(longString);
float floatValue = Float.parseFloat(floatString);
double doubleValue =
  Double.parseDouble(doubleString);
```