Software Engineering1 (Java)

CSY1019

RECAP

Sample Program

```
// Display a greeting in the console window

public class HelloPrinter
   {
       public static void main(String[] args)
      {

          System.out.println("Hello, World!");
       }
    }
```

//Recap: How to use Eclipse and set up workspace [Guides on NILE]

Variables

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

data type **variable name**

```
int total;

int count, temp, result;
```

Multiple variables can be created in one declaration

Variable Initialization

- A variable can be given an initial value in the declaration

```
int sum = 0;
int base = 32, max = 149;
```

- When a variable is referenced in a program, its current value is used
- See example: Variable.java

Identifiers

- Identifiers are programmer-defined names for:
  - classes
  - variables
  - methods
- Identifiers may not be any of the Java reserved keywords.

Identifiers

- Identifiers must follow certain rules:
  - An identifier may only contain:
    - letters a–z or A–Z,
    - the digits 0–9,
    - underscores (_), or
    - the dollar sign ($)
  - The first character may not be a digit.
  - Identifiers are case sensitive.
    - `itemsOrdered` is not the same as `itemsordered`.
  - Identifiers cannot include spaces.

Variable Names

- Variable names should be descriptive.
- Descriptive names allow the code to be more readable; therefore, the code is more maintainable.
- Which of the following is more descriptive?

```
double tr = 0.0725;
double salesTaxRate = 0.0725;
```

- Java programs should be *self-documenting*.

Java Naming Conventions

- Variable names should begin with a lower case letter and then switch to title case thereafter:

  Ex: `int caTaxRate`

- Class names should be all title case.

  Ex: `public class BigLittle`

- More Java naming conventions can be found at:

  http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html

A general rule of thumb about naming variables and classes are that, with some exceptions, their names tend to be nouns or noun phrases.

Primitive Data Types

- There are eight primitive data types in Java
- Four of them represent integers
  - `byte`, `short`, `int`, `long`
- Two of them represent floating point numbers
  - `float (e.g. float num = 23.5F;)`
  - `double (e.g. double num = 14520.904;)`
- One of them represents characters
  - `char (e.g. char letter = 'a';)`
- And one of them represents boolean values
  - `Boolean (e.g. boolean flag = true;)`

Numeric Primitive Data

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

Type	Storage	Min Value	Max Value
`byte`	8 bits	-128	127
`short`	16 bits	-32,768	32,767
`int`	32 bits	-2,147,483,648	2,147,483,647
`long`	64 bits	$< -9 \times 10^{18}$	$> 9 \times 10^{18}$
`float`	32 bits	$+/- 3.4 \times 10^{38}$ with 7 significant digits	
`double`	64 bits	$+/- 1.7 \times 10^{308}$ with 15 significant digits	
`char`	16 bits		
`boolean`	1 bit		

Integer Data Types

- `byte`, `short`, `int`, and `long` are all integer data types.
- They can hold whole numbers such as 5, 10, 23, 89, etc.
- Integer data types cannot hold numbers that have a decimal point in them.
- Integers embedded into Java source code are called *integer literals*.
- See Example: IntegerVariables.java

Floating Point Data Types

- Data types that allow fractional values are called *floating-point* numbers.
  - 1.7 and -45.316 are floating-point numbers.
- In Java there are two data types that can represent floating-point numbers.
  - `float` - also called *single precision* (7 decimal points).
  - `double` - also called *double precision* (15 decimal points).

Floating Point Literals

- When floating point numbers are embedded into Java source code they are called *floating point literals*.
- The default type for floating point literals is `double`.
  - 29.75, 1.76, and 31.51 are `double` data types.
- See example: Sale.java

Floating Point Literals

- A `double` value is not compatible with a `float` variable because of its size and precision.
  - `float number;`
  - `number = 23.5; // Error!`
- A `double` can be forced into a `float` by appending the letter F or f to the literal.
  - `float number;`
  - `number = 23.5F; // This will work.`

The `boolean` Data Type

- The Java `boolean` data type can have two possible values.
  - `true`
  - `false`
- The value of a `boolean` variable may only be copied into a `boolean` variable.

See example: TrueFalse.java

The `char` Data Type

- The Java `char` data type provides access to single characters.
- `char` literals are enclosed in single quote marks.
  - ‘a’, ‘Z’, ‘\n’, ‘1’
- Don’t confuse `char` literals with string literals.
  - `char` literals are enclosed in single quotes.
  - String literals are enclosed in double quotes.

See example: Letters.java

Arithmetic Operators

- Java has five (5) binary arithmetic operators.

Operator	Meaning	Type	Example
+	Addition	Binary	`total = cost + tax;`
-	Subtraction	Binary	`cost = total - tax;`
*	Multiplication	Binary	`tax = cost * rate;`
/	Division	Binary	`salePrice = original / 2;`
%	Modulus	Binary	`remainder = value % 5;`

Integer Division

- Division can be tricky.
  - In a Java program, what is the value of 1/2?
- You might think the answer is 0.5…
- But, that's wrong.
- The answer is simply 0.
- Integer division will truncate any decimal remainder.

A Closer Look at the / Operator

- / (division) operator performs integer division if both operands are integers

```
X = 13 / 5;     // X = 2
Y = 91 / 7;     // Y = 13
```

- If either operand is floating point, the result is floating point

```
X = 13 / 5.0;   // X = 2.6
Y = 91.0 / 7;   // Y = 13.0
```

A Closer Look at the % Operator

- % (modulus) operator computes the remainder resulting from integer division

```
a = 13 % 5;    // a = 3
```

Precedence

- If an expression contains more than one operator, Java uses **precedence rules** to determine the order of evaluation. The arithmetic operators have the following relative precedence:

Parentheses ( )
unary –
* / %
+ –

highest

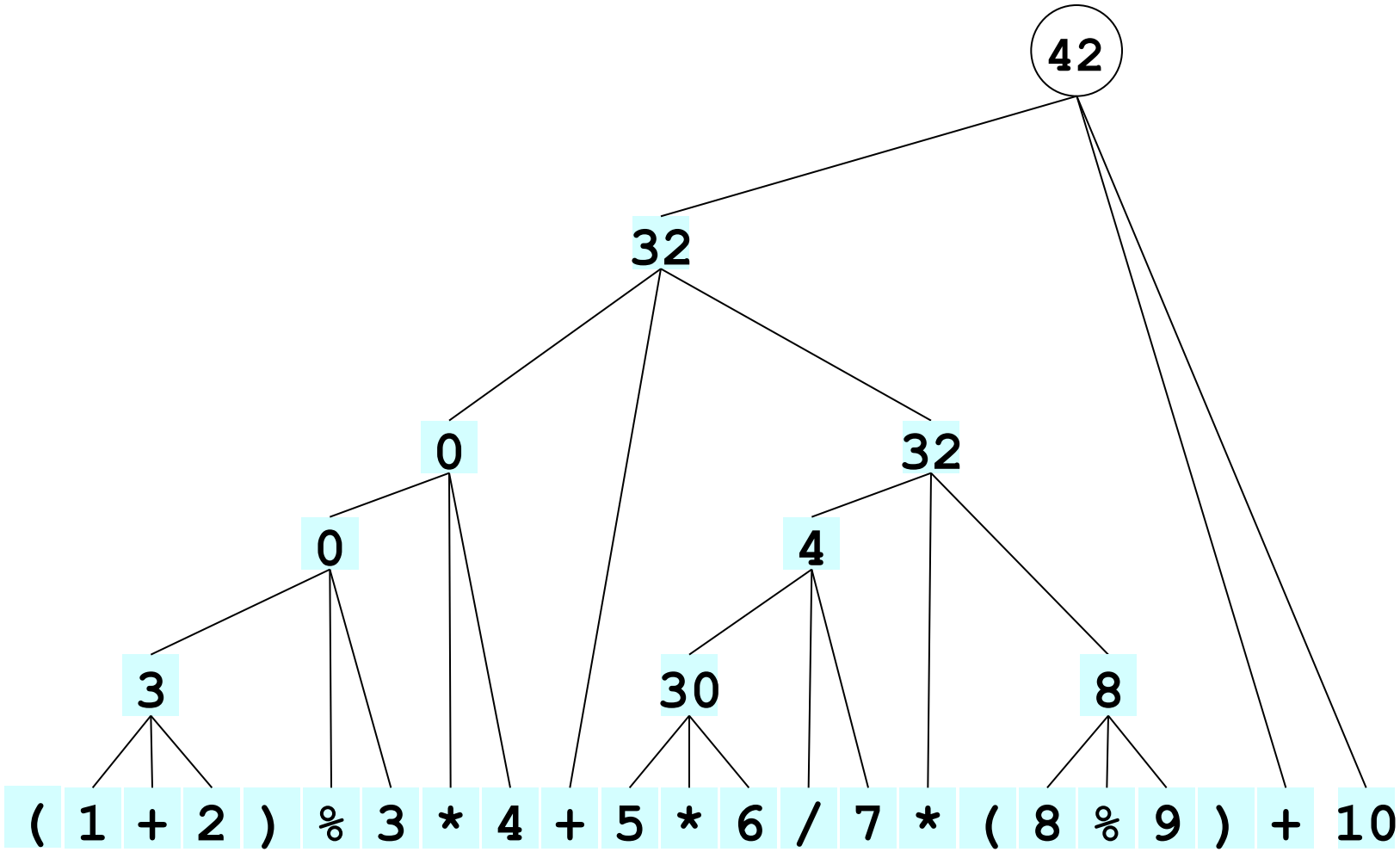
↕

lowest

Operator	Associativity	Example	Result
– (unary negation)	right to left	`x = -4 + 3;`	-1
`* / %`	left to right	`x = -4 + 4 % 3 * 13 + 2;`	11
`+ -`	left to right	`x = 6 + 3 - 4 + 6 * 3;`	23

Exercise: Precedence Evaluation

What is the value of the expression at the bottom of the screen?



The Class `String`

- We've used constants of type `String` already.

  `"Enter a whole number from 1 to 99."`
- A value of type `String` is a
  - Sequence of characters
  - Treated as a single item.

String Constants and Variables

- Declaring

```
String greeting;
 greeting = "Hello!";
```

or

```
String greeting = "Hello!";
```

or

```
String greeting = new String("Hello!");
```

- Printing

```
System.out.println(greeting);
```

Example: StringDemo.java

The `Scanner` Class

- The `Scanner` class is defined in `java.util`, so we will use the following statement at the top of our programs:

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

Reading Input

- The following line creates a Scanner object that reads from the keyboard

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

- The `new` operator creates the `Scanner` object
- Once created, the `Scanner` object can be used to invoke various input methods, such as

```
String answer = scan.nextLine();
```

Reading Input

```
String line = scan.nextLine(); // for a line of text
String word = scan.next(); // for a word
char character = scan.next().charAt(0)// for character
int integer = scan.nextInt();//for integer
double number = scan.nextDouble();//for double
……................
and so on …
```

The `Scanner` class is part of the `java.util` class library, and must be imported into a program to be used. (use import java.util.Scanner)

Example Programs (using Scanner Class)

See:

ScannerDemo.java

Payroll.java

GasMileage.java

Scope

- *Scope* refers to the part of a program that has access to a variable's contents.
- Variables declared inside a method (like the main method) are called *local variables*.
- Local variables' scope begins at the declaration of the variable and ends at the end of the method in which it was declared.

See example: Scope.java (This program contains an intentional error.)

Programming Style

- Although Java has a strict syntax, whitespace characters are ignored by the compiler.
- The Java whitespace characters are:
  - space
  - tab
  - newline
  - carriage return
  - form feed

See example: Compact.java

Indentation

- Programs should use proper indentation.
- Each block of code should be indented a few spaces from its surrounding block.
- Two to four spaces are sufficient.
- Tab characters should be avoided.
  - Tabs can vary in size between applications and devices.
  - Most programming text editors allow the user to replace the tab with spaces.

See example: Readable.java

TODAY

Combined Assignment Operators

- Java has some combined assignment operators.
- These operators allow the programmer to perform an arithmetic operation and assignment with a single operator.
- Although not required, these operators are popular since they shorten simple equations.

Combined Assignment Operators

Operator	Example	Equivalent	Value of variable after operation
+=	`x += 5;`	`x = x + 5;`	The old value of **x** plus 5
-=	`y -= 2;`	`y = y - 2;`	The old value of **y** minus 2
***=**	`z *= 10;`	`z = z * 10;`	The old value of **z** times 10
/=	`a /= b;`	`a = a / b;`	The old value of **a** divided by **b**
%=	`c %= 3;`	`c = c % 3;`	The remainder of the division of the old value of **c** divided by 3

See : CombinedAssignmentOperators.java

Increment and Decrement

- The increment and decrement operators use only one operand
- The *increment operator* (`++`) adds one to its operand
- The *decrement operator* (`--`) subtracts one from its operand
- The statement `count++;`

  is functionally equivalent to

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

- The statement `count--;`

  is functionally equivalent to

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

See: IncrementDecrement.java

Increment and Decrement

- The increment and decrement operators can be applied in ***postfix*** *form* `count++` or ***prefix*** *form* `++count`
- When used as part of a larger expression, the two forms can have different effects
- For example:

```
int count = 1;
int total  = count++;
System.out.println("total="+ total);
System.out.println("count="+ count);
```

OUTPUT:

total = 1

count = 2

```
int count = 1;
int total  = ++count;
System.out.println("total="+ total);
System.out.println("count="+ count);
```

OUTPUT:

total = 2

count = 2

Ranking

Primitive Data Type Ranking:

double	Highest Rank
float	
long	
int	
short	
byte	Lowest Rank

Assignment and types

- A variable can only store a value of its own type.
  - int x = 2.5; **// ERROR: incompatible types**
- An int value can be stored in a double variable.
  - The value is converted into the equivalent real number.
  - double myGPA = 4;

`myGPA`	`4.0`

  - double avg = **11 / 2**;

`avg`	**`5.0`**

    - Why does avg store 5.0 and not 5.5 ?

Type Casting

- Conversion can be forced with type casting
- What was illegal

```
int integerPart = realNumber; // ILLEGAL!
```

- Can be made legal

```
int integerPart = (int)realNumber;
```

  by preceding the desired type in parentheses
- This is considered a unary operator

Type Casting

```
int i = (int) 2.45; // i = 2
int j = 2;
double k = j; //k = 2.0
double d = 11 / 2; // d = 5.0
double  d = (double) 11 / 2; // d = 5.5
double  e = (double) (11 / 2); // e=5.0
```

Note: Remember the precedence of arithmetic operators. Cast operators are considered unary in the precedence table.

Creating Constants

- Many programs have data that does not need to be changed.
- Littering programs with literal values can make the program hard do read and maintain.
- Replacing literal values with constants remedies this problem.
- Constants allow the programmer to use a name rather than a value throughout the program.
- Constants also give a singular point for changing those values when needed.

Creating Constants

- Constants keep the program organized and easier to maintain.
- Constants are identifiers that can hold only a single value.
- Constants are declared using the keyword `final`.
- Constants need not be initialized when declared; however, they must be initialized before they are used or a compiler error will be generated.

Creating Constants

- Once initialized with a value, constants cannot be changed programmatically.
- By convention, constants are all upper case and words are separated by the underscore character.

```
final int CAL_SALES_TAX = 725;
```

The Math Class

- Math.pow method:

```
double r = Math.pow(2.0, 3.0); // r = 8.0
```

- Math.sqrt method:

```
double s = Math.sqrt(9.0); // s = 3.0
```

- Math.round method:

```
long t = Math.round(9.584); // t = 10
```

.....and many other mathematical functions and constants such as:

Math.PI = 3.141592653589793

Java Doc API

Java Documentation of the Application Programming Interface for all class libraries:

Visit:

http://download.oracle.com/javase/7/docs/api/