WIT COMP1000

Data Types and

Mathematical Expressions

Review

- Data values are stored in memory
  - Stored with bits (0, 1)
  - 8 bits make up one byte (0-255)
- A program uses the *type* of the data to tell Java two things
  - How much memory is required (how many bytes)
  - How to interpret the bits (is it a number or a character, etc)

Data Types - Numbers

- **`int`**
  - » Integer, whole numbers
  - » Examples: 0, 15, -100464, 420712003, -1
  - » Range: -2^{31} (-2147483648) to $2^{31}-1$ (2147483647)
  - » 4 bytes of memory
- **`double`**
  - » Numbers with a fractional component (15 digit precision)
  - » Examples: 11.23, -959.75, 0.5, -1.0
  - » Range: $\sim 10^{-308}$ to $\sim 10^{308}$, positive or negative
  - » 8 bytes of memory

Data Types - Alphanumeric

- `char`
  - Single character or symbol
  - Always put in <u>single</u> quotes
  - Examples: `'a'`, `'C'`, `'3'`, `'.'`, `'$'`
  - 1 byte of memory
- `String`
  - A sequence of characters and/or symbols
  - Always put in <u>double</u> quotes
  - Examples: `"Hello World"`, `"475!"`, `"a"`, `"$"`

Notes about Strings

- The `String` type is actually a Java *class*
  - Others we've discussed are *primitive* types
- We'll talk more about classes later
- For now, it means that there extra *methods* that you can use with every `String` variable
  - For example, there is a `length()` method that will tell you how long a string is

String Example

```
public class ClassExamples {

    public static void main(String[] args) {
        String message = "May the force be with you.";
        System.out.println(message);
        System.out.println("The above string is this long: " + message.length());
    }

}
```

Data Types - Boolean

- **`boolean`**
  - Boolean valued
  - Only values: `true`, `false`
  - At least one byte of memory

Mixing Types

- In general, you can not assign a value of one type to a variable of another type
  - » There are some exceptions that come with caveats
  - » There are ways to force the conversion in some cases
- Rule of thumb: don't mix types except when necessary, and always be careful when you do

Mixing Types

- You normally can't assign a double value to an **int** variable because you would lose information
  - » `int sum = 1.99;  //compiler error`
  - » Same is true when assigning from a **double** variable
- You can assign an **int** value to a **double** variable without any problems
- Strings and characters don't mix in either direction
  - » `char letter = "A";  //compiler error`
  - » `String name = 'a';  //compiler error`

Mixing Types

- Characters and integers are interchangeable using the ASCII character codes
  - » http://www.asciitable.com
  - » Example:
    - `char letter = 33;  //letter will be '!'`
    - `int letter = 'A';  //letter will be 65`

Mathematical Operators

- Used with numeric types (**int**, **double**)
  - » Addition (+): `total = part1 + part2;`
  - » Subtraction (-): `left_over = total – used;`
  - » Multiplication (*): `force = mass * acceleration;`
  - » Division (/): `item_wt = total / num_items;`
- When both operands are of type **int**, the result is also of type **int**
- When one or both operands are of type **double**, the result is also of type **double**

Assignment Statements

- Notice in all the previous examples the math statements look like: `VARIABLE = FORMULA;`
- This is because they are NOT formulas!
  - » In other words, they are NOT statements of fact like in normal mathematical equations
- Every "math" statement in Java is used to calculate a *one time* result when that line executes and then the "equation" is no longer remembered
  - » Sequential execution!

Assignment Statements

- The result of one of these one-time math calculations can be stored in a variable
- The variable name must go on the left side of the expression
- Example: `total_inches = yards * 36;`
  - » When this statement is executed (and ONLY then), Java plugs in the current value of the `yards` variable, multiplies by 36, and updates the value of `total_inches` to be the result

Common Mistake

```
import java.util.Scanner;

public class ClassExamples {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);
        int input_value;
        int squared_value;

        squared_value = input_value * input_value;

        System.out.print("Enter the value: ");
        input_value = input.nextInt();

        System.out.println(input_value + "^2=" + squared_value);

    }

}
```

Uninitialized variable error!

Common Mistake

- The previous program won't compile because the programmer forgot about sequential execution
- The math statement comes *before* the `input_value` variable is initialized (given a value)
  - » Before the `input_value = input.nextInt()` line in this case
- Java is smart enough to realize that the variable won't have a value and gives us a compiler error
- To fix it, move the math statement *after* `input_value` has been initialized (but before you print out the result!)

Corrected

```java
import java.util.Scanner;

public class ClassExamples {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);
        int input_value;
        int squared_value;

        System.out.print("Enter the value: ");
        input_value = input.nextInt();

        squared_value = input_value * input_value;

        System.out.println(input_value + "^2=" + squared_value);

    }

}
```

Integer Division

- When dividing two integers, the result is an integer
  - Any fractional value is thrown away!
  - Example: `int answer = 7 / 2; // answer = 3`
  - The *remainder* of an integer division can be accessed with the % (mod, modulus) operator
  - Example: `int remainder = 7 % 2; // remainder = 1`

Long Division Review

16 divided by 5:

```
      3
5 | 16
  - 15
    1
```

16/5 = 3

16%5 = 1

35 divided by 3:

```
      11
3 | 35
  - 30
     5
   - 3
     2
```

35/3 = 11

35%3 = 2

Constant Values

- When you include actual numbers in a mathematical expression, they will be treated as either **`int`** or **`double`** values
- If it is a whole number, it will be an **`int`**
  - » 4, 0, -11, 999999, -101
- If it has any numbers after the decimal point, it will be a **`double`**
  - » 7.2, 0.1234, -15.2, 5.0, -1.0

Yes, these are doubles!

Integer Division

- It doesn't matter what type of variable stores the result, only what the two values being divided are
  - » If both the numerator and denominator are `int` values, then the result is an `int` value
- Example: `double result = 5 / 4;`
  - » `result` is 1.0!
  - » 5 and 4 are integers, so the result is an `int` value of 1, which is converted to a `double` value of 1.0

Examples

```
int x = 5/2; // x is 2
double x = 5/2; // x is 2.0, 5/2 is 2, but x is a double
int x = 5.0/2; // compiler error! can't assign
                  double (2.5) to int
double x = 5.0/2; // x is 2.5
int x = 5/4*4; // x is 4, / and * are the same, so
                  evaluate left-to-right: (5/4)*4
int x = 5/(4*4); // x is 0, 4*4 is 16, 5/16 is 0

double x = 5/4*4.0;// x is 4.0, / and * are the same, so
                   evaluate left-to-right: (5/4)*4.0
int x = 5.0/4*8;  // compiler error! can't assign
                  double (10.0) to int
```

Exercise

- What is the output of the following?

```
int x = 5;
int y = 10;
double z = 2.5;
double a;

a = x / y;
System.out.println(a);
a = x / 1.0 * y;
System.out.println(a);
a = x / 2.0 * y;
System.out.println(a);
a = y / z;
System.out.println(a);
a = x % 3;
System.out.println(a);
```

Answer

- Run the code and see!

Multiplication Note

- In normal mathematical notation, we can omit the multiplication sign and everyone understands to multiply the numbers
- This does NOT work in Java, you have to have the multiplication signs
- Example: y = 5x
  - In Java: `y = 5*x; // y = 5x; will give you an error`

Complex Expressions

- Many operations can be combined in a single expression
  - » Use parentheses to specify order of evaluation
  - » Otherwise, default precedence rules are followed
  - » In general, use parentheses to be sure it is right
  - » Examples
    - `double ans = (b*b) - 4*a*c;` // $b^2 - 4ac$
    - `int result = x*(y + z);` // x(y+z)

Operator Precedence

Evaluated First

`( )`	parentheses
`*, /, %`	multiplication, division, mod
`+, -`	addition, subtraction
`=`	assignment

Evaluated Last

Multiple operators at the same level will be evaluated left-to-right.

Exercise

- Write a Java program that reads exactly three integers from the user, calculates the average of the three numbers, and prints out the average

Answer

```
import java.util.Scanner;

public class ClassExamples {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        int a, b, c;
        double average;

        System.out.println("Enter three integers:");
        a = input.nextInt();
        b = input.nextInt();
        c = input.nextInt();

        average = (a + b + c) / 3.0;

        System.out.println("The average is " + average);

    }

}
```

The ".0" after 3 is necessary to get a **double** result!

Printing Review

- So far you know about two methods to print the value of variables, as well as anything literal in quotes
  - `System.out.println()`
  - `System.out.print()`
- To sequence values/variables together, use the plus (+) operator
  - `System.out.println("Example: " + x);`

Numeric Output

- What if you wanted to output exactly two decimal places of a number (with rounding), or thousands separators (i.e. 1234 vs. 1,234)?
- These methods are generally useful for printing strings and integers, but Java supports even greater control when printing numbers (particularly large numbers/decimals)
  - » `System.out.printf("format", arg1, arg2, …)`

Example

```
public static void main(String[] args) {

    double smallNum = 0.031752;
    int bigNum = 88452;

    System.out.println("Value = " + smallNum); // 0.031752
    System.out.printf("Value = %.3f%n", smallNum); // 0.032
    System.out.printf("Value = %.2e%n", smallNum); // 3.18e-02

    System.out.println();

    System.out.println("Value = " + bigNum); // 88452
    System.out.printf("Value = %,d%n", bigNum); // 88,452

}
```

Format String

- The format string contains literals (items you want outputted verbatim), converters, and flags
  - » A converter looks to the arguments to fill in a value
    - Starts with a % and ends with a single character code
  - » A flag modifies a converter with options
    - Goes between the % and the converter code
- Each time you use a converter, you must supply a corresponding argument (other than newline)

Some Converters, Flags

Converter	Flag	Description
d		An integer
f		A float (includes double)
e		A float in scientific notation.
n		New line
	+	Includes the sign (positive or negative)
	,	Includes grouping characters
	.3	Three places after the decimal.

Many more options exist:
https://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html#syntax

Exercise

Write a program that asks the user for a decimal value – output that value with exactly three decimal places, rounding as necessary.

```
Enter a value: 3.14159

Rounded: 3.142
```

Answer

```
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter a value: ");
    double value = input.nextDouble();
    System.out.printf("Rounded: %.3f%n", value);
}
```

Printing Methods

- When outputting numbers, the `System.out.printf` method allows you more control than `System.out.print/System.out.println`

- The syntax is first a format string, then any number of arguments (with each non-newline converter having an argument)

`Math` Library

- Java also has libraries that contain additional *methods* for doing more complex calculations
  - » Square root
  - » Power
  - » Absolute value
  - » Logarithms
  - » Trigonometric functions
  - » …

Math Library

- Syntax to use the square root function:

```
RESULT = Math.sqrt(VALUE);
```

- For example:

```
double s = Math.sqrt(100.0); // s = 10.0
```

- Syntax to use the power function:

```
RESULT = Math.pow(VALUE, POWER);
```

- For example:

```
double p = Math.pow(5, 2); // p = 25 (5^2)
```

- Note that the parentheses are necessary

Example

```
import java.util.Scanner;

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

        Scanner input = new Scanner(System.in);

        double input_value;
        double squared;
        double square_root;

        System.out.print("Enter a value: ");
        input_value = input.nextDouble();

        squared = Math.pow(input_value, 2);
        square_root = Math.sqrt(input_value);

        System.out.println();
        System.out.println(input_value + "^2=" + squared);
        System.out.println(input_value + "^(1/2)=" + square_root);

    }
}
```

Exercise

- Write a program that reads two values (x and y) from the user, calculates x^y, and prints the answer

Answer

```java
import java.util.Scanner;

public class ClassExamples {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        double x, y, ans;

        System.out.print("Enter x: ");
        x = input.nextDouble();
        System.out.print("Enter y: ");
        y = input.nextDouble();

        ans = Math.pow(x, y);

        System.out.println();
        System.out.println(x + "^" + y + "=" + ans);

    }

}
```

Wrap Up

- Mathematical statements in Java are NOT like "normal" math formulas
- They are used only once to calculate a new value, when the statement is executed in sequential order
- Operator precedence is used just like in your calculator, but it's always best to use parentheses for complex expressions anyway
- When dividing two `int` values, the result is an `int` (use long division and throw away the remainder)