Lesson 33: Arithmetic Evaluation
Balboa HS

Jill Seman
Student Teacher – Balboa AP CS

Do Now

Write an **iterative** method called multiply() that takes two arguments, int a and int b, and computes their product using repeated additions. (Recall PS #4a). Method may look like this:

```
public static int multiply(int a, int b)
 {
        int product = ?;
        while( ?? )
        {
                //  Add code here
        }
return // ??;
}
```

Do Now

Now write the recursive version of multiply().

- New project: Lesson33
- In class MultiplyRec:

```
public static void main(String[] args)
{
        System.out.println( multiply(5,3) );
}
```

```
public static int multiply(int a, int b)
{ //your recursive implementation...
}
```

- Plan how the recursive method should work, discussing with others
  - How does knowing the answer to multiply(5,2) help solve multiply(5,3)
  - Multiply(5,1) should trigger a base case

Arithmetic Operations

Determine what is printed to the screen in each example:

```
int a = 7;
int b = 4;
```

```
System.out.print(a + b);

System.out.print(a  -  b);

System.out.print(a  *  b);

System.out.print(a  /  b);
```

What type would java assign the object printed to the screen here:

```
System.out.println ( 9 / 4);
```

Arithmetic Operations

```
System.out.println ( 9 / 4);

// 2!   Java auto evaluates this as an integer.
```

Java is a strongly typed language.

- Java wants both operands need to have the same type to evaluate.
- The result is of that same type.

Arithmetic Operations: Modulo

Modulo - % - calculates the remainder after two numbers are divided:

a % b = remainder when a is divided by b.

6 % 3 = 0 // 6 / 3 = 2 R0

7 % 3 = 1 // 7 / 3 = 2 R1

7 % 4 = ?

// 3

Modulo uses:

Calculating the remainder – or find out how much is left over after you divide two numbers. (Can be used for decimals.)

- Finding even numbers. If a number is even, it doesn't have a remainder when divided by 2:

```
If  (n % 2 == 0 )
        {// do operation for even number; }
else    { // do operation for odd number; }
```

** HW example from 4a **

- Modulus can also be used to find even 10s, 100s.
- Find things that happen on cyclical bases – hourly, daily, weekly.

Using Modulo – Part 2

HW example from 4a (Litvin Ch.4 #2)

The last problem set had a problem where you had to write a method that would return the value of:

$$1 - 1 / 2 + 1 / 3 - \ldots +/- \quad 1/n$$

for a given n:

Solution to Litvin Ch.4, #2

```
public static double naturalLogOfTwo(int n)
{
        double sum = 0;
        int i = 1; // current term will be 1/i
        int j = 0; // if even, add term...

        while ( i <= n ) {
                if ( j % 2 == 0 ) { //even
                sum += 1.0/i;
                 } else { //odd
                    sum -= 1.0/i;
                }
                i++;
                j++;
        }
        return sum;
}
```

Order of Operations

Operator	Operation	Order
()	**P**arentheses	Executes first
Math.pow(a,b)	**E**xponents	Executes next
*	**M**ultiplication	Executes next
/	**D**ivision	
+	**A**ddition	Executes next
-	**S**ubtraction	

Use **PEMDAS** acronym from math to remember

Order of Operations

Which operations with the same precedence?

- Multiplication & Division
- Addition & Subtraction

How are ties resolved?

Ties are broken by executing operations from left to right.

Where does modulo fit in PEMDAS?

( ), x^y, [* / %], [+ -]

Arithmetic Operations

```
int n = 4  + 2 * 3  - 6 / 2 * 2;
```

```
= 4  + 2 * 3  - 6 / 2 * 2
 = 4 +    6     - 6 / 2 * 2
= 4 +    6     -  3     * 2
= 4 + 6         -          6
= 10            -          6
 =                4
```

Arithmetic Operations

We can use parentheses to group terms, like in Algebra:

Algebra: $\frac{(1 + 5) * 3 + 2}{5}$

Java: ((1 + 5) * 3 + 2) / 5

Arithmetic Equations:

```
// dimensions of rectangle
int length = 5;
int width = 8;
```

Write java expressions for:

```
int area  =
int area  =  length * width ;

int perimeter =
int perimeter =  2 * (length + width);
```

Operation types

Determine what is output to the screen:

```
int a = 10;
int b = 9;
System.out.println(10 / 9);

System.out.println(10/6);
```

- Java is a strongly typed language.
- Result type = types of variables evaluated.
- Results are not rounded, they are ….
- How to fix this?

Type conversion - Casting

One way to fix type issues is by adjusting type:

```
double a = 9.0;
 double b = 2.0;
 double c = a / b;
                c = 4.5     ☺
```

Another way is to convert the variable type. This is called **CASTING**. This is done with the (type) command:

```
int a = 9;
 double b = (double) a
                                  //    b = 9.0
```

Type conversion - Casting - 2

CASTING:

```
int a = 9;
int b = 2;
double c = (double) a /  (double) b;
```

= (double) 9 / (double) 2
= 9.0 / 2.0
= 4.5

c = 4.5 ☺

Type conversion - Casting

Evaluate:

```
int a = 7;
int b = 3;
double c = a / b
        // value of c = 2.0   ☹
```

How to fix?

* Initialize double b = 3.0;
* double c = double a / b;

Type conversion - Casting

- Casting (double) a in an equation, does not change a's value into a double. It is still stored as an int. It is only converted for the equation.
- Need to insure that type conversion converts will be within acceptable range.
- Converting results in loss of precision.
- Why cast at all? Why not just use doubles all the time?

Type conversion: Promotion

Is this valid in java?:

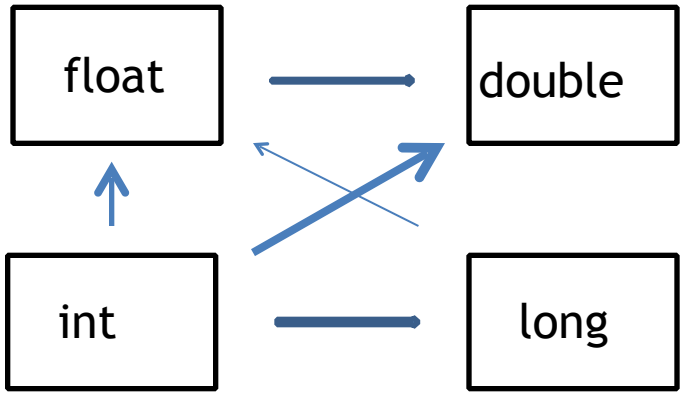
```
double result1 = 9.4 / 2;
```

- Both operands of an operation need to have the same type.
- Java will automatically convert one of the types. A process called **PROMOTION**.
- Java will promote the type of lower value up to double: Java will execute the following:

```
double result1 = 9.4 / 2.0;
```

result1 = 4.7 (a double)

Graph of Promotions



- Double cannot be promoted
- Nothing can be promoted to char or byte.
- Always a chance that you may lose precision when converting types.

Operation fun – java style:

```
3 / 2                                   // 1

2 / 3                                   // 0

1 * (2 / 3)                             // 0

1 / (2 / 3)                             // Exception

double z = 7.5 + 5.2                    // 13.7

int z = 7.5 + 5.2                       // Exception

double z =  7.5 – 5.4                   // 2.0999999999999996
```

Temperature Conversion

Calculate Fahrenheit:

$$C = 5/9\ (F - 32)$$

```
int degreesCelsius
        = 5 / 9 * (degreesFahrenheit – 32);
              =  0 * (degreesFahrenheit – 32);

        =  (double) 5 / 9 * (degreesFahrenheit – 32);
        = (double) (0) * (degreesFahrenheit – 32);

         = 5.0 / 9.0 * (degreesFahrenheit – 32);
                  //    Compiler gives type mismatch

         = 5.0 / 9.0 * (degreesFahrenheit – 32);
        =  21.111111111111111       //  This is correct!   ☺
```

Casting Comparisons

What will the output be here:

```
double c = 2.0;
int d = 2;
System.out.println(c == d);
```

Output is = true.

Even though c and d are 2 different types, because c and d are in fact equal.

Casting Comparisons

What will the output be here:

```
double c = 2.0;
 double tiny = 0.00000000000000000001;
  c += tiny;
  int d = 2;
 System.out.println(c == d);
```

Output is = true.

- Integers and doubles don't actually have to be equal for them to evaluate as equal!
- Be aware when mixing types in comparisons.
- Common mistake in the 'real world'

HW

Work on problem set 4 of PS #5. Try to complete most of all of it by next class.

Start looking at the two programming exercises in PS 5. The earlier you start thinking about the problems, the easier you'll find solving them!