Lesson 5…..Mixed Data Types, Casting, and Constants

So far we have looked mostly at simple cases in which all the numbers involved in a calculation were either **all** integers or **all** *doubles*. Here, we will see what happens when we **mix** these types in calculations.

Java doesn't like to lose data:

Here is an important principle to remember: Java **will not** normally store information in a variable if in doing so it would **lose** information. Consider the following two examples:

1. An example of when we would **lose** information:

```
double d = 29.78;
int  i = d;  //won't compile since i is an integer and it would have to chop-off
             // the .78 and store just 29 in i….thus, it would lose information.
```

   There is a way to make the above code work. We can **force** compilation and therefore result in 29.78 being "stored" in *i* as follows (actually, just 29 is stored since *i* can only hold integers):

```
int i = (int)d;  //(int) "casts" d as an integer… It converts d to integer form.
```

2. An example of when we would **not** lose information:

```
int j = 105;
double d = j; //legal, because no information is lost in storing 105 in the
              // double variable d.
```

The most precise:

In a math operation involving **two different data types**, the result is given in terms of the **more precise** of those two types…as in the following example:

```
int i = 4;
double d = 3;
double ans = i/d;  //ans will be 1.33333333333333…the result is double precision
```

20 + 5 * 6.0 returns a *double*. The 6.0 might look like an integer to us, but because it's written with a decimal point, it is considered to be a floating point number…a *double*.

Some challenging examples:

What does 3 + 5.0/2 + 5 * 2 – 3 return? **12.5**

What does 3.0 + 5/2 + 5 * 2 – 3 return? **12.0**

What does (int)(3.0 + 4)/(1 + 4.0) * 2 – 3 return? **-.2**

Don't be fooled:

Consider the following two examples that are very similar…but have different answers:

```
double d = (double)5/4;  //same as 5.0 / 4…(double) only applies to the 5
System.out.println(d);  //1.25

int j = 5;
int k = 4;
double d = (double)(j / k); //(j / k) is in its own little "world" and performs
                            //integer division yielding 1 which is then cast as
                            //a double, 1.0
System.out.println(d);  //1.0
```

Constants:

Constants follow all the rules of variables; however, once initialized, they **cannot be changed**. Use the keyword ***final*** to indicate a constant. Conventionally, constant names have all capital letters. The rules for legal constant names are the same as for variable names. Following is an example of a constant:

```
final double PI = 3.14159;
```

The following illustrates that constants can't be changed:

```
final double PI = 3.14159;
PI = 3.7789;  //illegal
```

When in a method, constants may be initialized after they are declared.

```
final double PI;  //legal
PI = 3.14159;
```

Constants can also be of type *String*, *int* and other types.

```
final String NAME= "Peewee Herman";
final int LUNCH_COUNT = 122;
```

The real truth about compound operators:

In the previous lesson we learned that the compound operator expression *j+= x;* was equivalent *to j = j + x;*. Actually, for **all compound operators** there is also an **implied cast** to the type of *j*. For example, if *j* is of type *int*, the real meaning of *j+= x;* is:

```
j = (int)(j + x);
```

Project… Mixed Results

Create a new project called *MixedResults* with a class called *Tester*. Within the *main* method of *Tester* you will eventually printout the result of the following problems. However, you should first calculate by hand what you expect the answers to be. For example, in the parenthesis of the first problem, you should realize that strictly integer arithmetic is taking place that results in a value of 0 for the parenthesis.

```
double d1 = 37.9;  //Initialize these variables at the top of your program
double d2 = 1004.128;
int i1 = 12;
int i2 =  18;
```

Problem 1: 57.2 * (i1 / i2) +1
Problem 2: 57.2 * ((double)i1 / i2) + 1
Problem 3: 15 – i1 * (d1 * 3) + 4
Problem 4: 15 – i1 * (int)(d1 * 3) + 4
Problem 5: 15 – i1 * ((int)d1 * 3) + 4

Your printout should look like the following:

```
Problem 1: 1.0
Problem 2: 39.13333333333333
Problem 3: -1345.39999999999
Problem 4: -1337
Problem 5: -1313
```

Exercise on Lesson 5

Unless otherwise instructed in the following problems, state what gets printed.

1. Write code that will create a constant *E* that's equal to 2.718.

2. Write the simplest type constant that sets the number of students, *NUM_STUDENTS*, to 236.

3. What's wrong, if anything, with the following code in the *main* method?
   final double Area;
   Area = 203.49;

4. int cnt = 27.2;
   System.out.println(cnt);
   What's printed?

5. double d = 78.1;
   int fg = (int)d;
   System.out.println(fg);
   What's printed?

6. Is *double f4 = 22;* legal?

7. The following code stores a 20 in the variable *j*:
   double j = 61/3;
   What small change can you make to this single line of code to make it produce the "real" answer to the division?

8. System.out.println((double)(90/9));