Mr G's Java Jive

#8: Math Operators and the Pots Program

With this handout you'll learn about the standard **math operators** in Java and write a program to ship pots from your ceramics shop.

Please Excuse My Dear Aunt Sally

If that heading made any sense to you at all, then you know that I'm talking about the standard six math operations of **parentheses**, **exponentiation**, **multiplication**, **division**, **addition**, and **subtraction**, because this is the mnemonic device used for remembering the **order of operations**.

Java has five of the standard math operators and adds one that wasn't part of the original list. We'll take a look at them right now:

Operator	Used For
+	addition
-	subtraction
*	multiplication
/	division
%	mod (modulus)
()	grouping

A careful look at this little chart will show that **exponentiation** is the mission operation, and that **mod** is the newly added one. More on this in a minute. It's time for a short math lesson.

Integer Math

We all know that **2+2=4**, that **7-4=3**, that **9*3=27**, and that **48/8=6**. But what about **17/2?** You're probably tempted to say that the answer is **8.5**, but it's not. At least not in Java. That's because **17** and **2** are both **integers**, and since both numbers are integers, Java does **integer division**.

What's **integer division?** That's when you figure out how many **whole** times one number goes into another. Since **2** only goes into **17** eight **whole** times, the answer is **8**.

I said that Java did **integer division** on those two numbers because they were both **integers** (or of the **int** type). But if **at least one** of those numbers were a **decimal**, then you'd end up with standard **decimal division**. For example, either **17.0/2**, **17/2.0**, or **17.0/2.0** would give the expected answer of **8.5**.

Now that you know this, we can take a look at our new operator.

The Mod Operator

No, this isn't something out of **Austin Powers**. **Mod** (as I knew it in other programming languages) is short for **modulus**, and I'll give a few examples to see if you can figure out how it works:

12/5=2
12%5=2
7/2=3
7%2=1
6/3=2
6%2=0

Give yourself a minute or two to think about this before you turn the page and see the answer.

What Mod Does

So now that you've had a chance to think about it, did you figure out that **mod** gives the **remainder** of a division problem? Let's take a look at the first one. **12/5=2** because **5** only goes into **12** two **whole** times. **12%5=2** because the remainder when you divide **12** by **5** is **2**.

Let's try another one. **7/2=3** because **2** only goes into **7** three **whole** times. **7%2=1** because the remainder when you divide **7** by **2** is **1**.

We'll do one more and then move on. **6/3=2** because **3** goes into **6 exactly** two times. That was simple. However, **6%3=0** because there is **no remainder** when you divide **6** by **3**.

Now, with that little lesson out of the way, it's time to get to work on our next program.

The Pots Program

For this program, you're the owner of a ceramics shop that sells pots and ships them out to people. Unlike Amazon.com, which seems to always ship everything in the largest box, you have three sizes of boxes to choose from: **small**, **medium**, and **large**. Each box holds **1**, **4**, and **9** pots respectively. Your job is to ask your customers how many pots they want, and then tell how many boxes of each size they'll be shipped in.

We'll start off with just the framework of our code to show what we're going to need to do:

```
//Pots
//a program to ship pots from my ceramics shop
//7.13.06 Clark Kent

public class Pots
{//start class
   public static void main()
   {//start main
      //variables

      //get input

      //do calculations

      //give output

   }//end main
}//end class
```

A Simple Conversation

Now that we have our framework, let's just write some simple code that greets the users and thanks them for shopping at our ceramics shop. Then we'll compile it and see if it runs.

```
{//start main
   //variables
     String name;

   //get input
     System.out.print("Hi. Welcome to my ceramics shop. What's your name? ");
     name=gatling.getLine();

   //do calculations

   //give output
     System.out.println("Thanks for shopping at my ceramics shop, "+name+".");
     System.out.println("Please come again!");

}//end main
```

Once we know that this works, it's time to add a little more to the program.

Find Out What They Want and How They Want It

Now that we have the basics taken care of, and we know that the program works so far, it's time to get some product info from the user and respond to it. Make the following changes:

```
//variables
        String name;
        int pots;

      //get input
        System.out.print("Hi. Welcome to my ceramics shop. What's your name? ");
        name=gatling.getLine();
        System.out.print("How many pots would you like to order, "+name+"? ");
        pots=gatling.getInt();

      //do calculations

      //give output
        System.out.println("Thanks for shopping at my ceramics shop, "+name+".");
        System.out.println("Your "+pots+" pots will be shipped tomorrow.");
        System.out.println("Please come again!");
```

Here we added a variable for the number of **pots** the user will be ordering. We've also added two lines for getting the information from the user and a line for confirming that we have the user's order information. Now it's time to think about the shipping

Finally, We Talk About Constants

There are three box sizes, **small**, **medium**, and **large**, and they hold **1**, **4**, and **9** pots respectively. We could create regular variables to hold these values, but these are special values. They're special because we don't want them to change, we want them to remain constant. In fact, in any other programming language, we'd define them as **constants**.

However, Java is a little funny about this. For some reason it doesn't use the word **constant**. Instead, Java says that when you don't want the value of a variable to change, you call it a **final** because you're setting up the **final** value of it. With this in mind, let's enter the following code:

```
//variables
  final int LG = 9;
  final int MD = 4;
  final int SM = 1;
  String name;
  int pots;
```

There is a programming **convention** that the names of all **constants** (even if they're called **finals**) are done in **all caps**. This is so that everyone knows that they're constants (oops, I mean finals). The computer doesn't really care how you do it, but it's a lot easier for the people who have to read your code.

As wonderful as these values are, we still need a few standard, garden variety, variables.

Boxes and Leftovers

In the code modifications shown below we've added four new variables. One for the **number of each type of box we'll need** and one for how many pots are **leftover** after we've filled the previous size box.

```
//variables
        final int LG = 9;
        final int MD = 4;
        final int SM = 1;
        String name;
        int pots, lgnum, mdnum, smnum, left;
```

Now we're finally ready to do some calculations!

Packing the Boxes

When you're trying to figure out how many boxes of different sizes you need, it's always best to start with the largest boxes and then work down to the smallest. That's exactly what the code shown below does. First it checks to see if there are enough pots to **totally fill** a bunch of large boxes. Then it finds out how many are **left over** and tries to put them into medium boxes. Finally, it finds how many are **left over** from that and tries to put them into small boxes:

```
//do calculations
  lgnum=pots/LG;
  left=pots%LG;
  mdnum=left/MD;
  left=left%MD;
  smnum=left/SM;
  left=left%SM;
```

Additional Output

Now that the program knows how many of each type of box the user needs, it's time to send that information out the screen. Check out the code shown below:

```
//give output
  System.out.println("Thanks for shopping at my ceramics shop, "+name+".");
  System.out.println("Your "+pots+" pots will be shipped tomorrow in:");
  System.out.println("\t"+lgnum+" large box(es).");
  System.out.println("\t"+mdnum+" medium box(es).");
  System.out.println("\t"+smnum+" small box(es).");
  System.out.println("Please come again!");
```

Right now you're probably wondering what on earth **\t** means. It's a **control** character that means to insert a **tab** (usually eight spaces) at that point. If you paid careful attention, you also saw that I changed the ending of the second **println** in this section.

Compile it, run it, and see what happens!

A Waste of Boxes

If you typed in everything right, the program should work just fine so far, but there's one little problem: it's a little too picky about things. What do I mean? Run the program and tell it that you need **18** pots. It should tell you that they'll be shipped in **2 large boxes**. That makes perfect sense because 18 pots fit exactly in that many boxes.

However, if you run it with **17** pots, it tells you that they'll be shipped in **1 large** and **2 medium** boxes. On the one hand that makes perfect sense, since it takes one **totally full** large box and two **totally full** medium boxes to ship 17 pots. On the other hand, if you think about it, that's a waste of boxes and postage. Why should sending 17 pots require **more** boxes than sending 18? It's all because the program is only capable of dealing with **totally full** boxes. This needs to change, and we'll take care of that in handout #9.