Mr G's Java Jive

#18: Dealing With Dates – Part 1

Way back in the beginning, in the **Converse** program, we asked the user what year they were born in and what year it is now. With that information we were able to give the user's **approximate** age. It was approximate because we didn't have the exact dates, and using separate variables for **date**, **month**, and **year** would be a little much for us to tackle at the time. It still is a bit much to tackle, but thanks to Java's own **Date** object type, and few date-related methods in the **gatling** class, we can try to make working with dates as painless as possible for a beginner.

FirstDate

We'll start this off by writing a short program to show you how Java handles dates. It"s called **FirstDate**, and like everything else we're going to do from now on, it's based on the **BigLoop** program. In the first parking space create a method called **datemain()**. This will do all the work for us. It should look like the example shown below:

```
//======== parking space 1
  public static void datemain()
  {//start datemain
     //variables
       Date now = new Date();

     //show output
       System.out.println("Today is "+now);
  }//end datemain
```

As you can see from what we've written here, we've created a new **Date object** called **now** in the **variables** section. The current date and time is assigned to the Date object the moment you create it. In fact, it's the current date and time right down to the millisecond. We'll talk about that later.

In the **show output** section, the program will show us what the current date is. However, there are two things you need to do before any of this can happen.

The first thing you need to do is to make sure that you've called the **datemain** method inside of your **main** method (right underneath where it says **insert method call below**). The second thing you need to do is to **import** the files you need in order to work with dates. In this case the file you want to import is **java.util.***. This not only imports the stuff you need for dates, but for lots of other useful functions too.

Now compile and run this to see if it works so far. If it does, keep telling the program that you want to run again a few times and take a look at the difference in times displayed on the screen. Here's an example of a few runs I did of it this morning as I was writing this handout:

The default format for showing dates gives you six things: the **day of the week**, the **month**, the **day of the month**, the **time**, the **time zone**, and the **year**. Since this format shows you the time down to the last second, you should notice that each time you run this, the time is a little different.

TMI and Date Formats

If you haven't been under a rock for the past few years, you probably know that this stands for **Too Much Information**. That's exactly what Java gives us with its default date format. Most of the time you don't need to know this much. The good news is that there are ways to fix this so that you only get what you want.

Remember how we were able to set up **number formats** to tell how many decimal places to show with our **double** values? Well, Java also has **date formats** you can use for dates. The three we'll deal with here are **SHORT**, **MEDIUM**, and **LONG**.

First create a new **String** object in your **variables** section and call it **nstring**. This stands for **n**ow **string**, and is the string that we'll put our formatted date into.

Next, create a new section under **variables** and call it **date formats**. Enter the code shown below into it.

```
//date formats
  DateFormat sh = DateFormat.getDateInstance(DateFormat.SHORT);
  DateFormat md = DateFormat.getDateInstance(DateFormat.MEDIUM);
  DateFormat lg = DateFormat.getDateInstance(DateFormat.LONG);
```

Don't you just love how Java makes creating these formats really easy? But to be fair, what looks really cumbersome to beginners actually gives the really advanced programmers a lot of flexibility. So just bear with me a little longer, I'll make things a lot easier for you in a little bit.

Now that you've created these three new date formats, you get to use them to see what they look like. Make the following changes in your **show output** section:

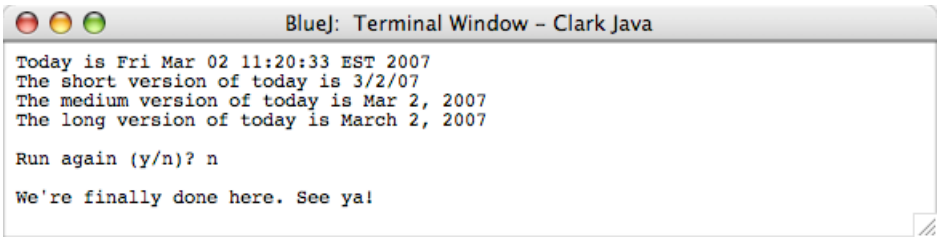
```
//show output
  System.out.println("Today is "+now);
  nstring=sh.format(now);
  System.out.println("The short version of today is "+nstring);
  nstring=md.format(now);
  System.out.println("The medium version of today is "+nstring);
  nstring=lg.format(now);
  System.out.println("The long version of today is "+nstring);
```

Let's take a close look at this new code to try to figure out what's going on.

First of all, the original output line will still show the date in the **default**, really long format. The line below that formats **now** with our newly defined **sh** format, and then puts the formatted date into **nstring**. Below that, the **println** statement tells us that we're looking at the **short** version and then outputs what's in **nstring**.

Now that we're done with the short version, we can reuse **nstring** to hold the **medium** version and print that out.

And finally, we reuse **nstring** one last time to hold the **long** version of the date and print that out. Once you've got all this code entered into your program, compile and run it. Here's what mine looked like:



As you can see, the **default** format continues to give you way too much information, while the **short** one almost isn't enough. And the only difference between the **medium** and **long** versions is that the **long** version gives you the full name of the month, while **medium** simply gives you the abbreviated name of the month.

Getting Dates

Having Java get the current date and time and displaying it was really easy. Having Java **get** the any date from a user is a little harder. Fortunately, there's a method in **gatling.java** that can do that for you easily. It's called **getDate**.

If you bother to take the time to look at the actual code for it, you'll see that **getDate** lets you input the date in three different standard formats: **default**, **short**, and **long**. It checks to see if what you typed in is a valid date in one of those formats, and if it is, it accepts it and puts it into a Date object. If it's not valid, it asks you to try again – just like **gatling.getInt** and **gatling.getDouble**.

Let's make some changes to **FirstDate** so that it will ask the user when they were born. First, in **datemain**, add a new **Date object** and call it **bday**. I'm hoping that it's fairly obvious that **bday** is short for **birthday**, which is what's going to go into it. While you're there, create a new **String object** called **bstring**. I hope I don't have to tell you what that stands for.

Now that we have our two new variables, create a new section between **date formats** and **show output** called **get input**. The code for it is shown below:

```
//get input
  nstring=lg.format(now);
  System.out.print("Today is "+nstring+", when were you born? ");
  bday=gatling.getDate();
```

The code we just wrote got the user's birthdate. Now we need to rewrite the **show output** section to show all the information we have so far. Rather than going through a confusing process of showing you which lines to keep and which ones to toss, I'm going to tell you to just gut the entire **show output** section. Delete everything except for the **section heading**. Now that you've done that, put in the following code.

```
//show output
  bstring=lg.format(bday);
  System.out.println("Cool. You were born on "+bstring);
```

Now compile and run it!

Format Recognition and Two-Digit Years

As I believe I mentioned earlier, the **getDate** method was written to give the user a little flexibility in entering dates. This means that you should be able to enter the **long** formatted date of **August 20, 2002**, the **medium** formatted date of **Aug 20, 2002**, and the **short** formatted date of **8/20/2002** without any problem and get the same result (be careful, though, and always remember to put the comma in after the day of the month in the long and medium formats). In addition, when you enter a date in the short format, Java will handle a **two-digit year** halfway decently – within reason. Within what **it** considers reason.

If you enter the long format date of **August 20, 02**, Java will happily assume that the year you really want is 0002. The same thing will happen if you use a two-digit year in a medium format date. But because two-digit years are common in the short format, Java's Date class was written to deal with them, and will try to give you the appropriate year. They key phrase here is "try to."

For example, if you enter **8/20/2002**, your program should give you a response of **August 20, 2002**. That seems like a perfectly obvious thing to do with a four-digit year. Now try the same date in the

short format with a two-digit date, and you'll see that **8/20/02** gives you the same response. This makes sense because 2002 is in this century.

So far we've been using my youngest daughter's birthday. Let's see what happens with my oldest's, who was born way back in the 20th century. When you enter **2/3/93**, what does the program say that the year actually is? Is it 1993 as it should be, or 2093 because it's in this century?

How about my birthday? Enter **6/19/56** and see which century the program says it fell in. Does it say I was born in 1956 or 2056?

Let's do two more. Try my father's birthday. He was born on **3/16/26**. Does this show up as 1926 or 2026? And finally, my grandmother's birthday. She was born on **3/14/08**. What does Java show her year as being?

By now you've probably noticed that Java doesn't seem to behave consistently with two-digit dates. Or does it? Is there some perfectly logical pattern regarding how it recognizes two-digit dates, and can you figure it out? This sounds like it would be perfect for a homework assignment – or a quiz question.

Date Math

Date math in Java is not for the faint of heart. For example, you can't just subtract **bday** from **now** to get a number representing the span of time between them. That would be too easy. You can't do this because they're both Date **objects**, and not **int** primitives. What you need to know is how to extract the integer portion of the information from both Date objects and then do the math on them.

But even then, things aren't as simple as you'd think they should be. Java measures time in **milliseconds**. This means that not only are there 1000 milliseconds per second, but 60,000 per minute, 3,600,000 per hour, etc. You can see where figuring out how many milliseconds there are in a year could get to be a bit much.

In fact, figuring out that many milliseconds **is** too much, and requires the use of a whole new datatype: the **long**. The **long** is an integer type that holds twice as many places as a regular **int**. But for the kind of date math that most people want to do, there's something much simpler. Well, simpler for you. It was a lot of work for me. These are the date handling methods in the **gatling** class.

We'll look at them in the next handout.