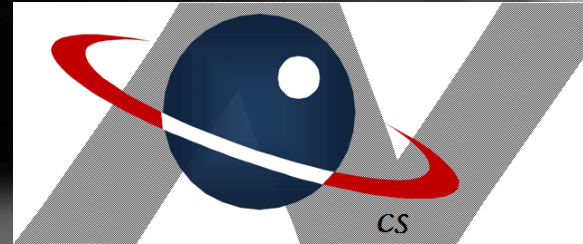


# *O(N) CS LESSONS*

*Lesson 1D – More **printf** formatting features*



*By John B. Owen*

*All rights reserved*

*©2011, revised 2015*

# Table of Contents



- [Objectives](#)
- [Field width, format specifiers, flags](#)
- [Calendar class suffixes](#)
- [Lesson Summary / Labs](#)
- [Contact Information for supplementary materials](#)

- [1. Field Width](#)
- [2. Different Field Widths](#)
- [3. Left Align, the - flag](#)
- [4. Field Width w/ integers](#)
- [5. Left Align w/ integers](#)
- [6. Overflow](#)
- [7. Negative values](#)
- [8. The + flag](#)
- [9. The +- flag combo](#)
- [10. Zero flag](#)
- [11. Comma flag](#)
- [12. \( flag](#)
- [13. Decimal values](#)
- [14. Formatted decimal values](#)
- [15. Dollar formatting](#)
- [16. Multiple columns](#)

# Objective #1

- The student will understand and apply advanced *printf* features dealing with field width and format flags using JAVA programming examples.
- CS1 TEKS 126.33c2(E) improve numeric display by optimizing data visualization;



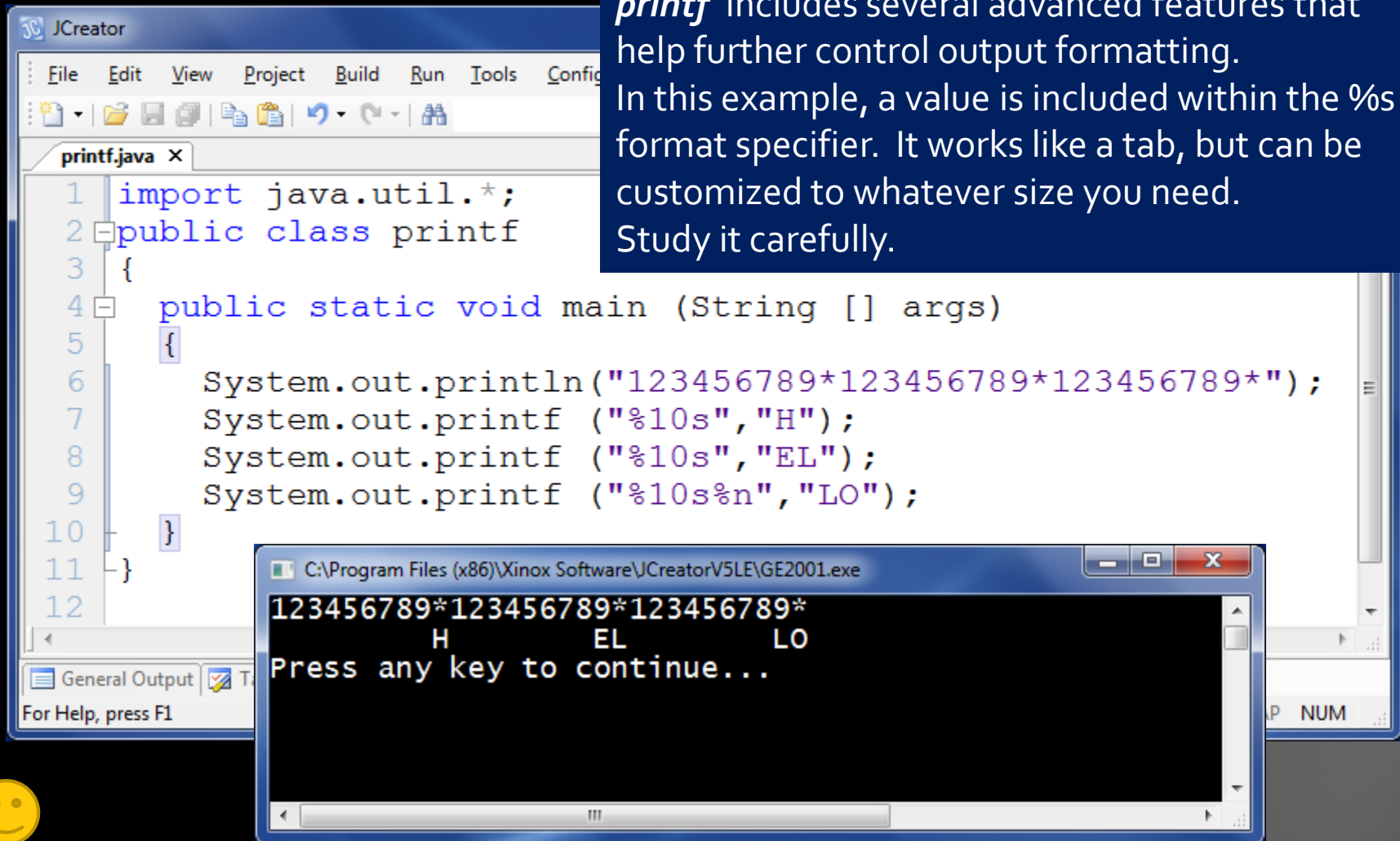
# Objective #2

- The student will understand and apply the various single character suffixes that are appended to the %t Calendar format specifier to output different aspects of dates and times.
- The student will understand how to use the set command to change the value of the current date and time.



# Example #1 -Field width specifier

*printf* includes several advanced features that help further control output formatting. In this example, a value is included within the %s format specifier. It works like a tab, but can be customized to whatever size you need. Study it carefully.



The screenshot shows the JCreator IDE with a file named `printf.java` open. The code defines a `main` method that prints a long string of asterisks, followed by three strings "H", "EL", and "LO" each formatted with a width of 10 spaces. The output window shows the result: the asterisks, then "H", "EL", and "LO" each right-aligned within 10-space fields, followed by a prompt to press any key to continue.

```
1 import java.util.*;
2 public class printf
3 {
4     public static void main (String [] args)
5     {
6         System.out.println("123456789*123456789*123456789*");
7         System.out.printf ("%10s", "H");
8         System.out.printf ("%10s", "EL");
9         System.out.printf ("%10s%n", "LO");
10    }
11 }
12
```

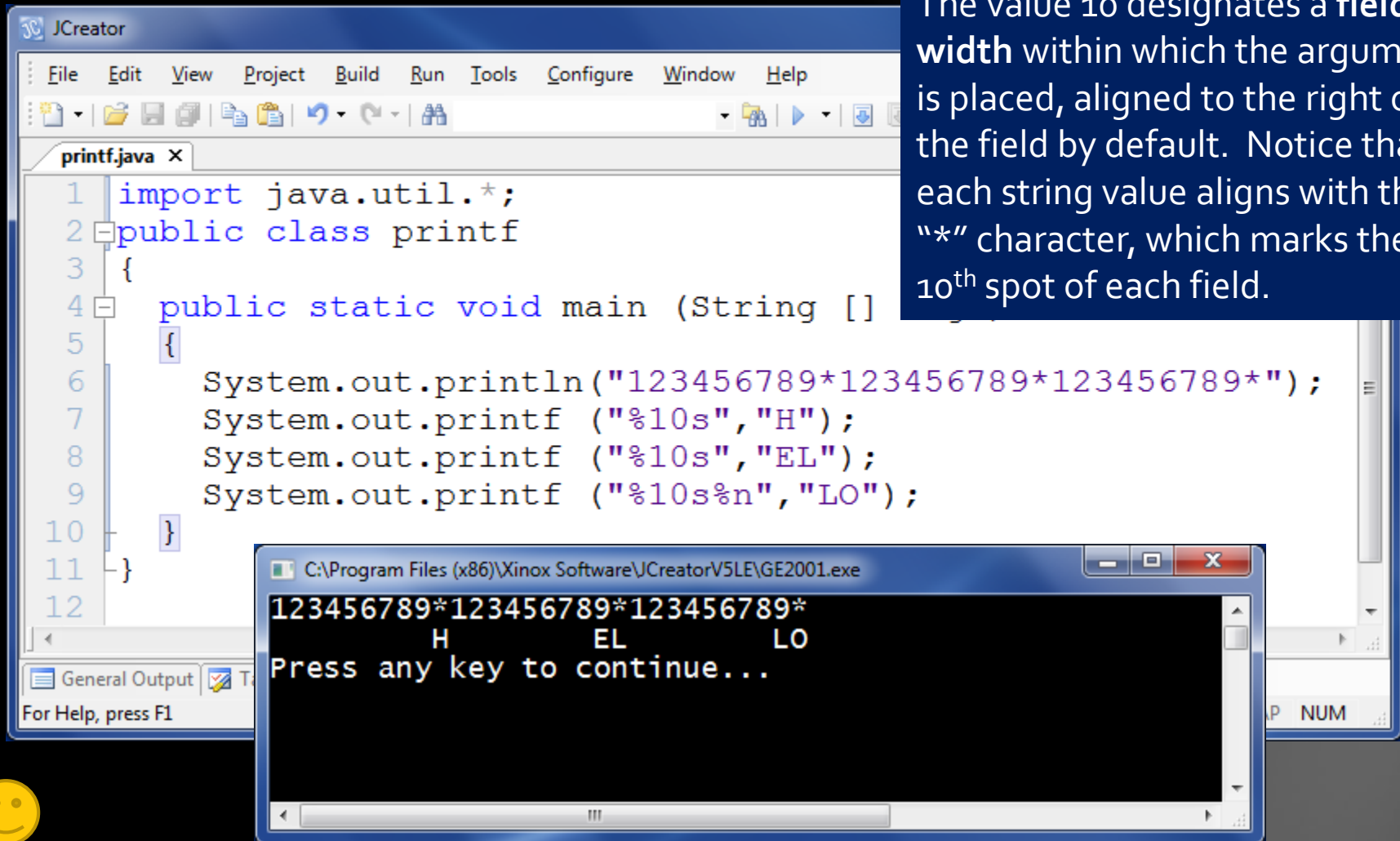
Output:

```
123456789*123456789*123456789*
                        H      EL      LO
Press any key to continue...
```



# Example #1 -Field width specifier

The value 10 designates a **field width** within which the argument is placed, aligned to the right of the field by default. Notice that each string value aligns with the "\*" character, which marks the 10<sup>th</sup> spot of each field.



The screenshot shows the JCreator IDE with a file named `printf.java` open. The code is as follows:

```
1 import java.util.*;
2 public class printf
3 {
4     public static void main (String []
5     {
6         System.out.println("123456789*123456789*123456789*");
7         System.out.printf ("%10s", "H");
8         System.out.printf ("%10s", "EL");
9         System.out.printf ("%10s%n", "LO");
10    }
11 }
12
```

Below the code editor, a "General Output" window displays the program's execution results:

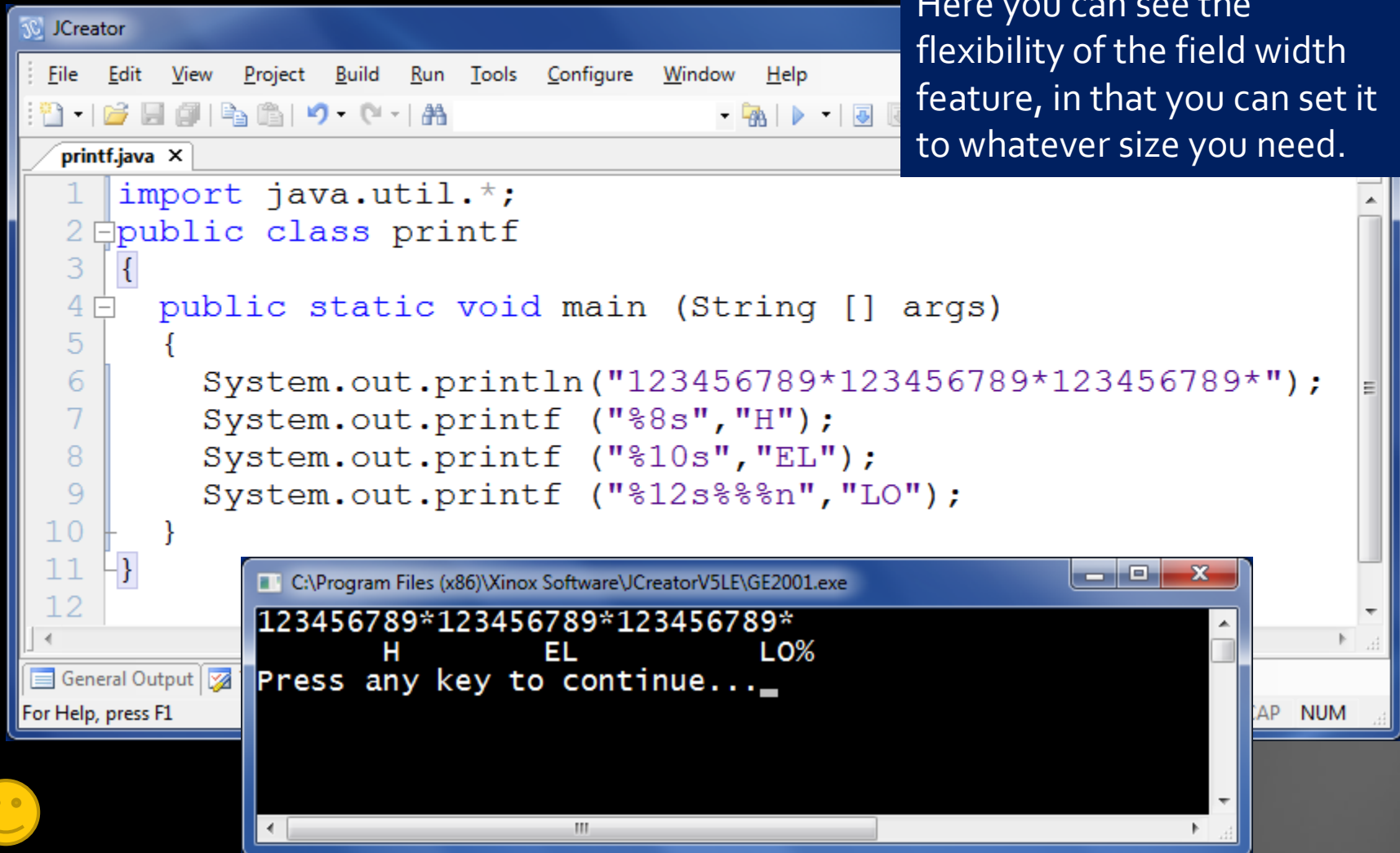
```
123456789*123456789*123456789*
          H          EL          LO
Press any key to continue...
```

The output demonstrates that the `%10s` format specifier aligns the strings "H", "EL", and "LO" to the right within a field of width 10, with the asterisk character marking the 10th position in each field.



# Example #2 - Different field widths

Here you can see the flexibility of the field width feature, in that you can set it to whatever size you need.



```
1 import java.util.*;
2 public class printf
3 {
4     public static void main (String [] args)
5     {
6         System.out.println("123456789*123456789*123456789*");
7         System.out.printf ("%8s", "H");
8         System.out.printf ("%10s", "EL");
9         System.out.printf ("%12s%%n", "LO");
10    }
11 }
12
```

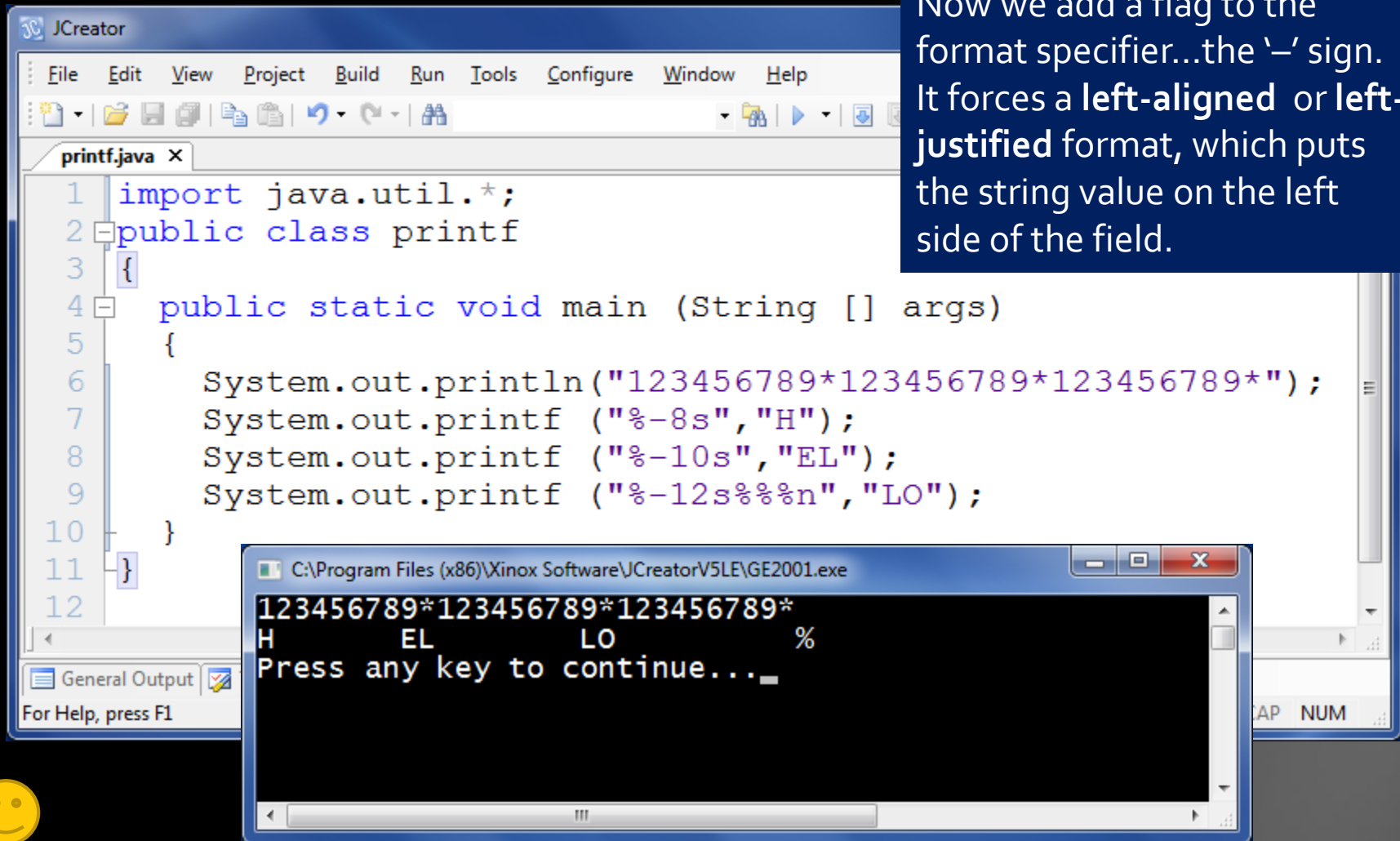
General Output  
For Help, press F1

CA\Program Files (x86)\Xinox Software\JCreatorV5LE\GE2001.exe

```
123456789*123456789*123456789*
      H      EL      LO%
Press any key to continue...
```

# Example #3 - Left-align flag '-'

Now we add a flag to the format specifier...the '-' sign. It forces a **left-aligned** or **left-justified** format, which puts the string value on the left side of the field.



The screenshot shows the JCreator IDE with a file named `printf.java` open. The code is as follows:

```
1 import java.util.*;
2 public class printf
3 {
4     public static void main (String [] args)
5     {
6         System.out.println("123456789*123456789*123456789*");
7         System.out.printf ("% -8s", "H");
8         System.out.printf ("% -10s", "EL");
9         System.out.printf ("% -12s%%n", "LO");
10    }
11 }
12
```

The output window shows the following text:

```
123456789*123456789*123456789*
H          EL          LO          %
Press any key to continue...
```

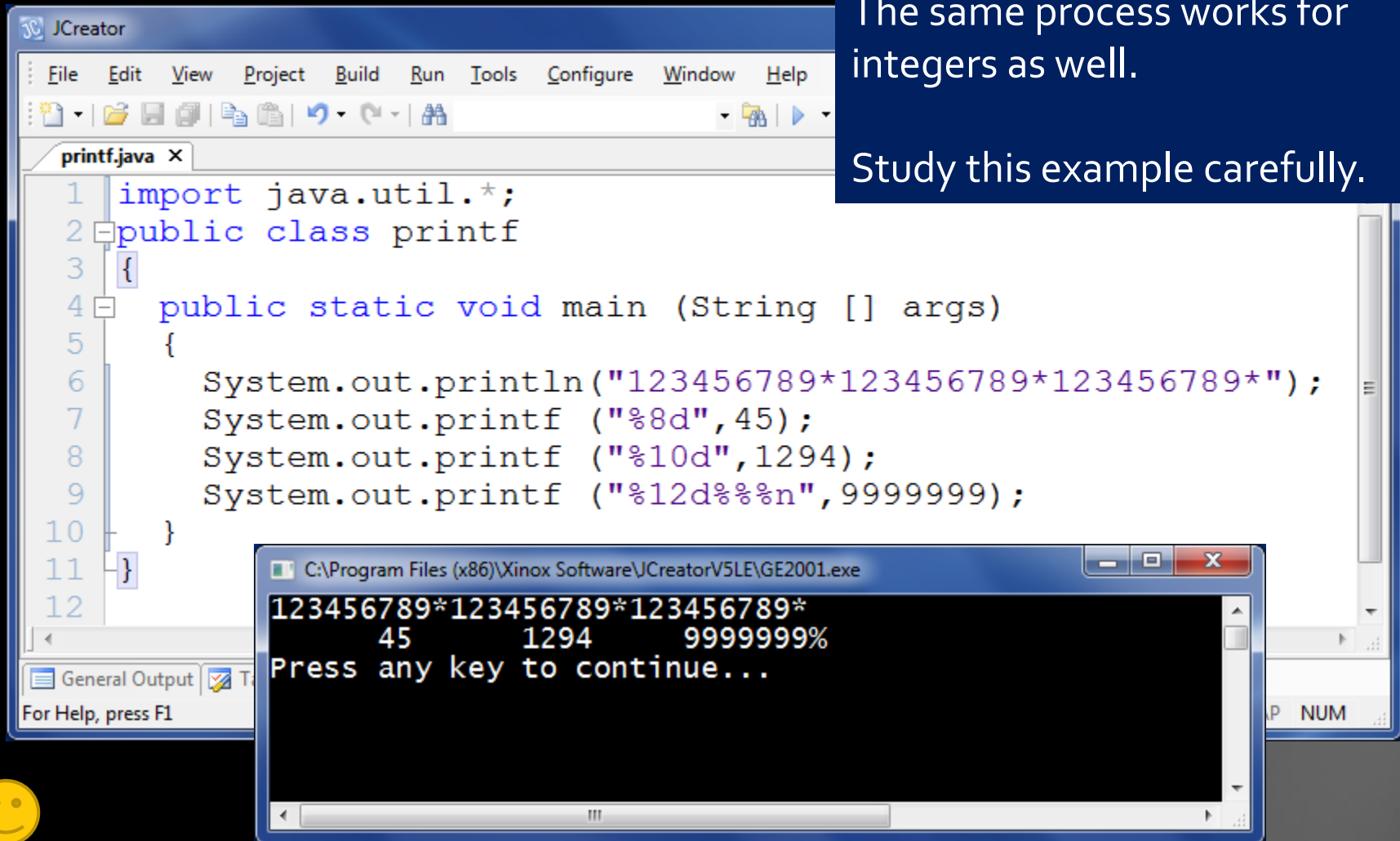




# Example #4 - Field width with integers

The same process works for integers as well.

Study this example carefully.



The screenshot shows the JCreator IDE with a file named `printf.java` open. The code is as follows:

```
1 import java.util.*;
2 public class printf
3 {
4     public static void main (String [] args)
5     {
6         System.out.println("123456789*123456789*123456789*");
7         System.out.printf ("%8d", 45);
8         System.out.printf ("%10d", 1294);
9         System.out.printf ("%12d%%n", 9999999);
10    }
11 }
12
```

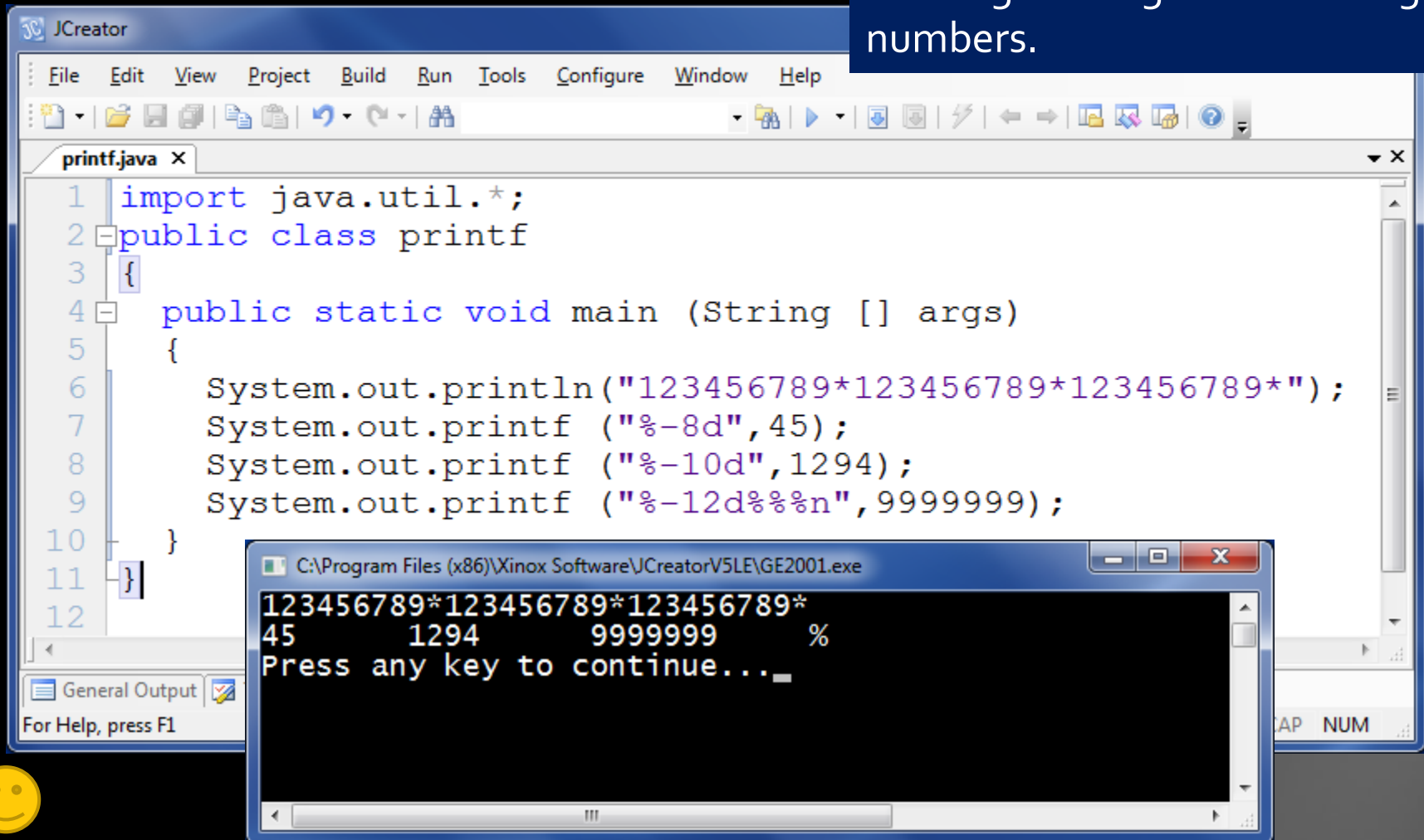
Below the code editor, the 'General Output' window is visible, showing the execution results:

```
123456789*123456789*123456789*
      45      1294      9999999%
Press any key to continue...
```

A yellow smiley face icon is located in the bottom left corner of the slide.

# Example #5 - Left-aligned works too

The negative sign also left-aligns numbers.



The screenshot shows the JCreator IDE with a file named `printf.java` open. The code defines a `printf` class with a `main` method that prints a multiplication string and three integers using `printf` with left-aligning format specifiers. A console window in the foreground shows the output of the program.

```
1 import java.util.*;
2 public class printf
3 {
4     public static void main (String [] args)
5     {
6         System.out.println("123456789*123456789*123456789*");
7         System.out.printf ("% -8d", 45);
8         System.out.printf ("% -10d", 1294);
9         System.out.printf ("% -12d%%n", 9999999);
10    }
11 }
12
```

Output:

```
123456789*123456789*123456789*
45      1294      9999999      %
Press any key to continue...
```



# Example #6 - Field width overflow

Hmmm...only room for 8 digits, but I have 9. I'll just take what I need!

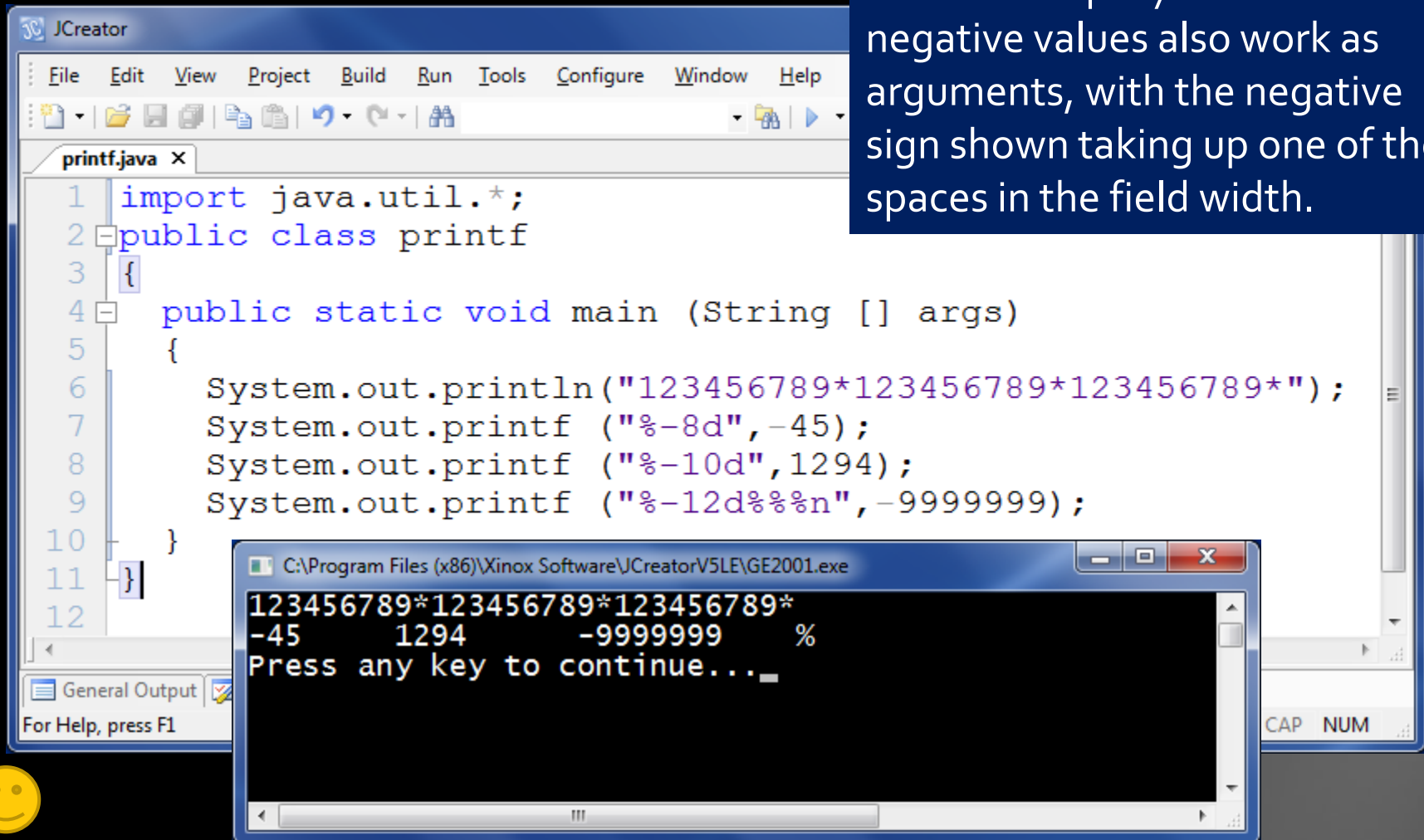
This example shows what happens when the field width is not "wide" enough for the value. The number simply takes what it needs to output itself and ignores the field width value entirely!

```
public static void main (String [] args)
{
    System.out.println("123456789*123456789*123456789*");
    System.out.printf ("%8d",123456789);
    System.out.printf ("%10d",1294);
    System.out.printf ("%12d%%\n",9999999);
}
```

```
123456789*123456789*123456789*
123456789      1294      9999999%
Press any key to continue...
```

# Example #7 – Negative values

In this example you can see that negative values also work as arguments, with the negative sign shown taking up one of the spaces in the field width.



The screenshot shows the JCreator IDE with a file named `printf.java` open. The code defines a `public class printf` with a `main` method. The `main` method contains four lines of code: a `println` statement with a string of asterisks, and three `printf` statements. The first `printf` uses `"%-8d"` and `-45`. The second uses `"%-10d"` and `1294`. The third uses `"%-12d%%\n"` and `-9999999`. Below the code editor, a 'General Output' window displays the program's output. The output shows the string of asterisks, followed by `-45` (right-aligned in a field of 8), `1294` (right-aligned in a field of 10), and `-9999999` (right-aligned in a field of 12, followed by a newline and a percent sign). The output window also shows the prompt 'Press any key to continue...'. A yellow smiley face icon is located in the bottom left corner of the slide.

```
1 import java.util.*;
2 public class printf
3 {
4     public static void main (String [] args)
5     {
6         System.out.println("123456789*123456789*123456789*");
7         System.out.printf ("% -8d", -45);
8         System.out.printf ("% -10d", 1294);
9         System.out.printf ("% -12d%%\n", -9999999);
10    }
11 }
12
```

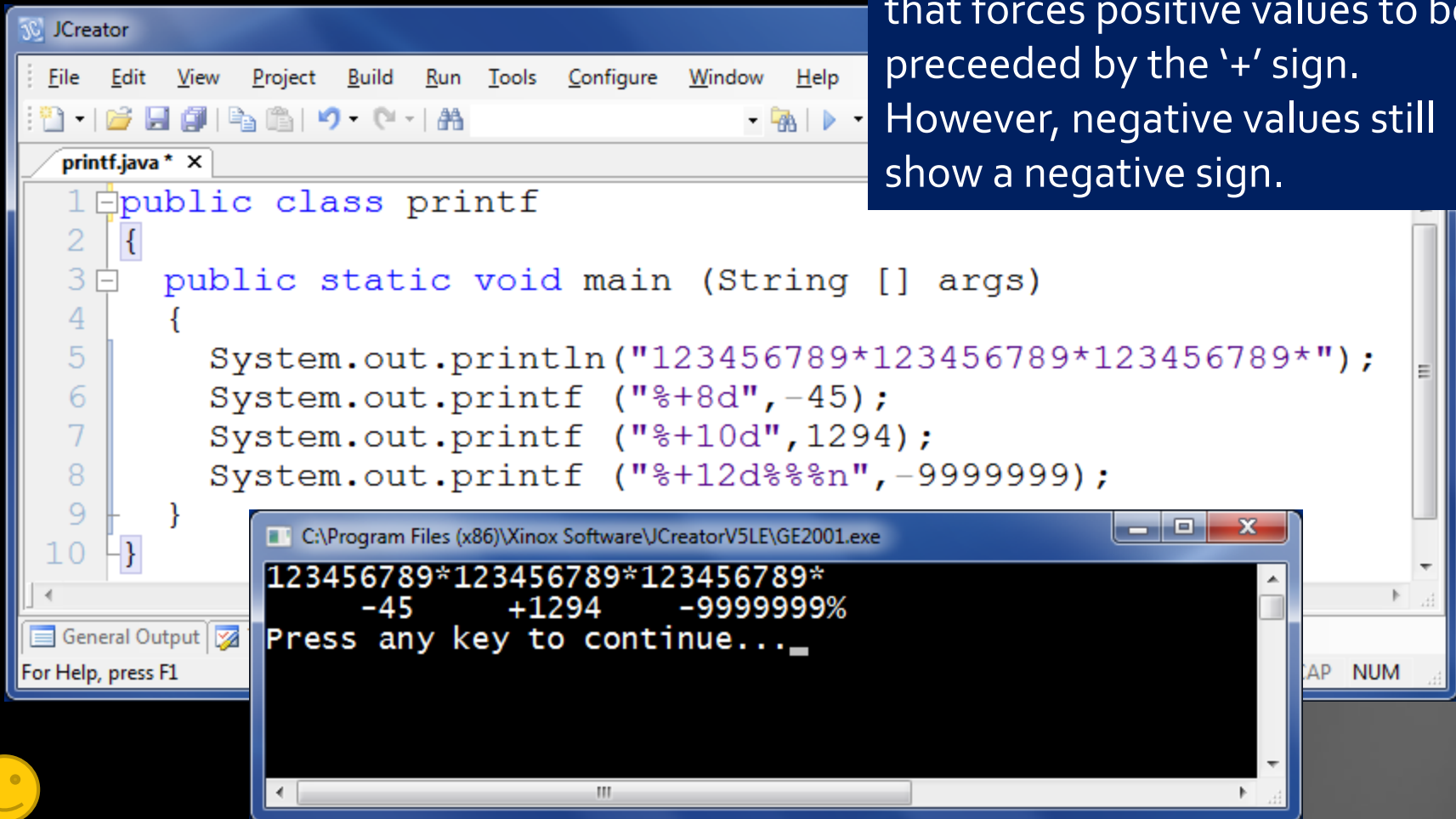
General Output  
For Help, press F1

C:\Program Files (x86)\Xinox Software\JCreatorV5LE\GE2001.exe

```
123456789*123456789*123456789*
-45      1294      -9999999      %
Press any key to continue...
```

# Example #8 – '+' flag shows positive

The '+' sign acts as a format flag that forces positive values to be preceded by the '+' sign. However, negative values still show a negative sign.



The screenshot shows the JCreator IDE with a file named `printf.java` open. The code defines a public class `printf` with a `main` method. The `main` method contains four lines of code that use `System.out` to print formatted strings. The first line prints a string of asterisks. The second line prints the integer `-45` using the format `"%+8d"`. The third line prints the integer `1294` using the format `"%+10d"`. The fourth line prints the integer `-9999999` using the format `"%+12d%%n"`. Below the code editor, there is a window titled `General Output` showing the output of the program. The output consists of three lines: a string of asterisks, the integer `-45` preceded by a plus sign and padded with spaces, the integer `+1294` preceded by a plus sign and padded with spaces, and the integer `-9999999` preceded by a plus sign and padded with spaces. The output window also displays the prompt `Press any key to continue...`.

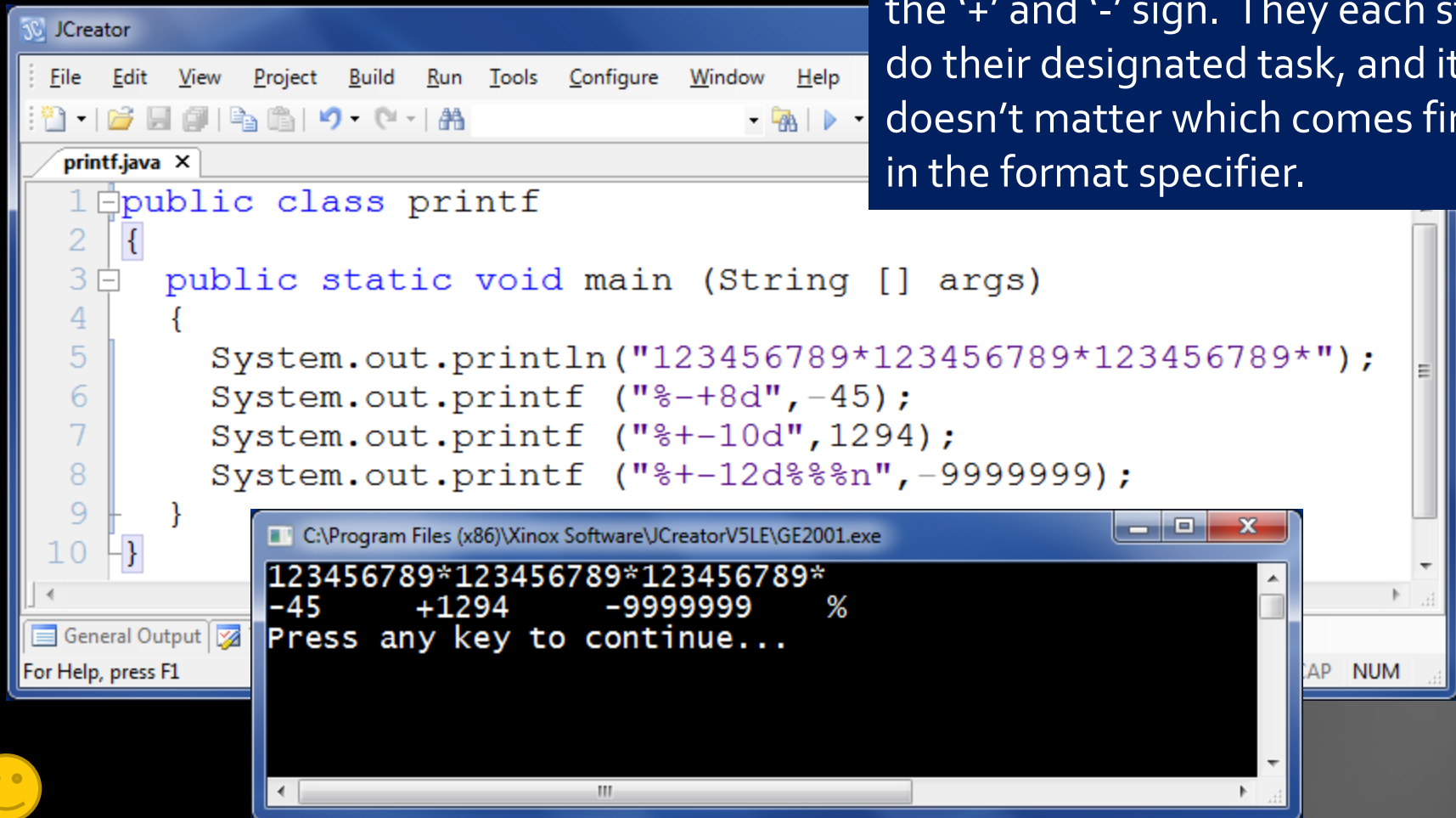
```
1 public class printf
2 {
3     public static void main (String [] args)
4     {
5         System.out.println("123456789*123456789*123456789*");
6         System.out.printf ("%+8d",-45);
7         System.out.printf ("%+10d",1294);
8         System.out.printf ("%+12d%%n",-9999999);
9     }
10 }
```

General Output  
For Help, press F1

```
123456789*123456789*123456789*
      -45      +1294      -9999999%
Press any key to continue...
```

# Example #9 – “-+” flag combination

Here you see a combination of the '+' and '-' sign. They each still do their designated task, and it doesn't matter which comes first in the format specifier.



The screenshot shows the JCreator IDE with a file named `printf.java` open. The code defines a public class `printf` with a `main` method. The `main` method contains four `System.out` statements: a `println` for a string of asterisks, and three `printf` calls using format specifiers that combine '-' and '+' flags. The output window shows the results of these statements: a line of asterisks, followed by the numbers -45, +1294, and -9999999, and a percentage sign. The output window also displays the file path `C:\Program Files (x86)\Xinox Software\JCreatorV5LE\GE2001.exe` and the prompt `Press any key to continue...`.

```
1 public class printf
2 {
3     public static void main (String [] args)
4     {
5         System.out.println("123456789*123456789*123456789*");
6         System.out.printf ("% -+8d", -45);
7         System.out.printf ("% + -10d", 1294);
8         System.out.printf ("% + -12d%%n", -9999999);
9     }
10 }
```

General Output  
For Help, press F1

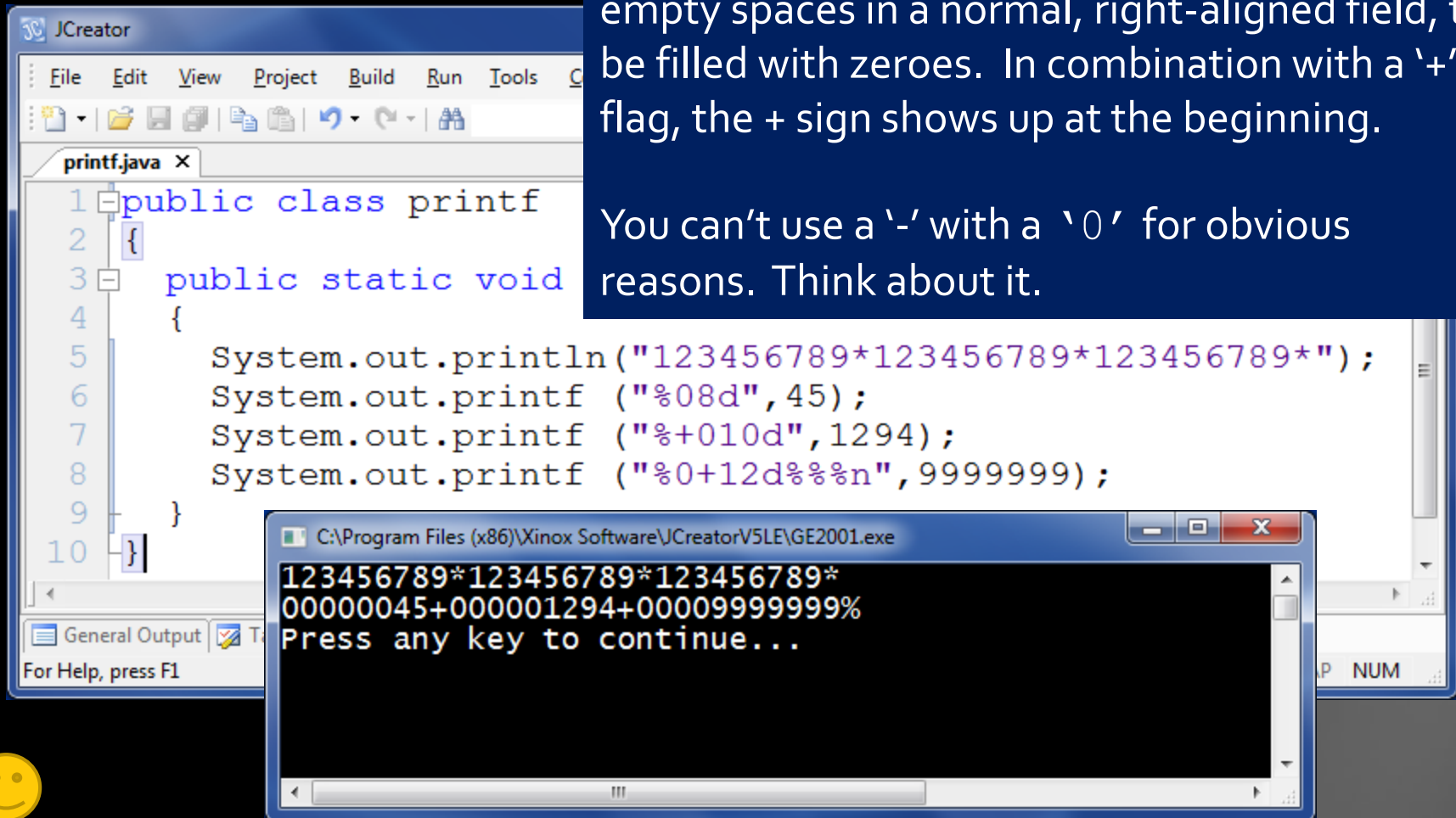
C:\Program Files (x86)\Xinox Software\JCreatorV5LE\GE2001.exe

123456789\*123456789\*123456789\*  
-45 +1294 -9999999 %  
Press any key to continue...

# Example #10 – Zero flag '0'

When a zero '0' is used as a flag, it causes any empty spaces in a normal, right-aligned field, to be filled with zeroes. In combination with a '+' flag, the + sign shows up at the beginning.

You can't use a '-' with a '0' for obvious reasons. Think about it.



The screenshot shows the JCreator IDE with a file named `printf.java` open. The code is as follows:

```
1 public class printf
2 {
3     public static void
4     {
5         System.out.println("123456789*123456789*123456789*");
6         System.out.printf ("%08d", 45);
7         System.out.printf ("%+010d", 1294);
8         System.out.printf ("%0+12d%%n", 9999999);
9     }
10 }
```

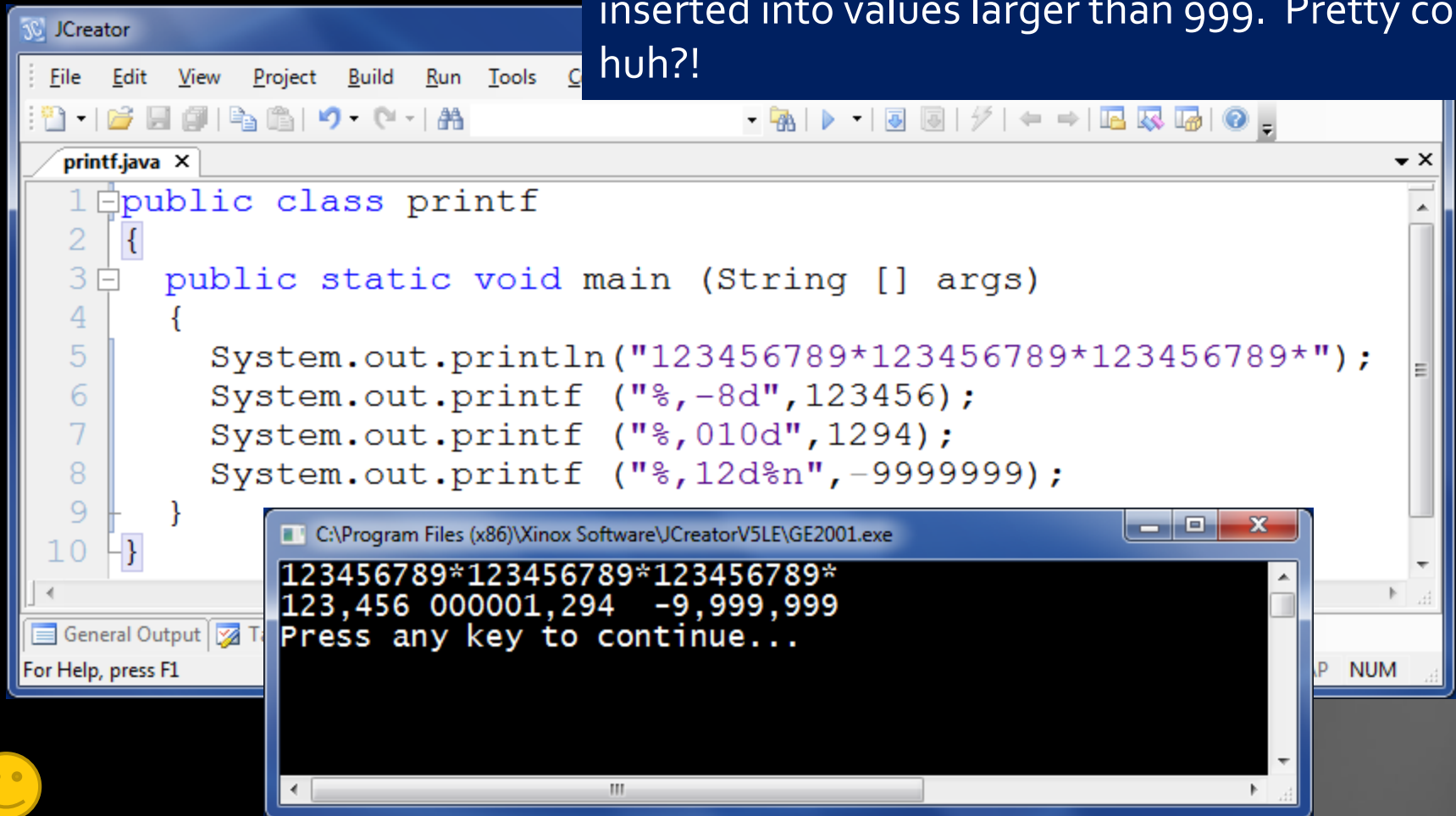
Below the code editor, the output window displays the following text:

```
123456789*123456789*123456789*
00000045+000001294+0000999999%
Press any key to continue...
```



# Example #11 – Comma flag `,'

The comma `,' flag causes commas to be inserted into values larger than 999. Pretty cool, huh?!



The screenshot shows the JCreator IDE with a file named `printf.java` open. The code is as follows:

```
1 public class printf
2 {
3     public static void main (String [] args)
4     {
5         System.out.println("123456789*123456789*123456789*");
6         System.out.printf ("%,-8d",123456);
7         System.out.printf ("%010d",1294);
8         System.out.printf ("%12d%n",-9999999);
9     }
10 }
```

Below the code editor, the 'General Output' window displays the program's execution results:

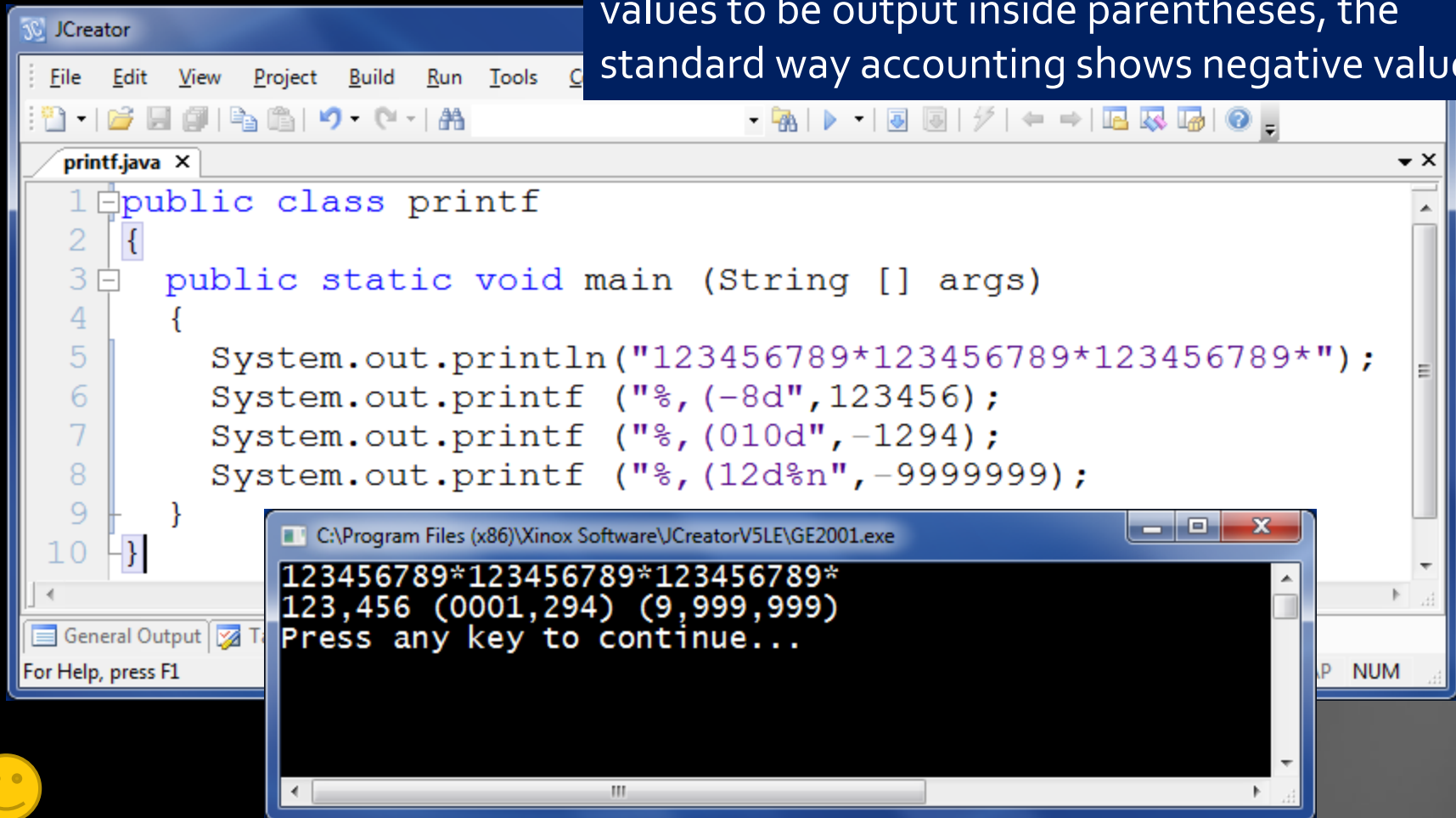
```
123456789*123456789*123456789*
123,456 000001,294  -9,999,999
Press any key to continue...
```





# Example #12 – Parentheses flag '('

The left parentheses '(' flag causes any negative values to be output inside parentheses, the standard way accounting shows negative values.



The screenshot shows the JCreator IDE with a file named `printf.java` open. The code is as follows:

```
1 public class printf
2 {
3     public static void main (String [] args)
4     {
5         System.out.println("123456789*123456789*123456789*");
6         System.out.printf ("%", (-8d", 123456);
7         System.out.printf ("%", (010d", -1294);
8         System.out.printf ("%", (12d%n", -9999999);
9     }
10 }
```

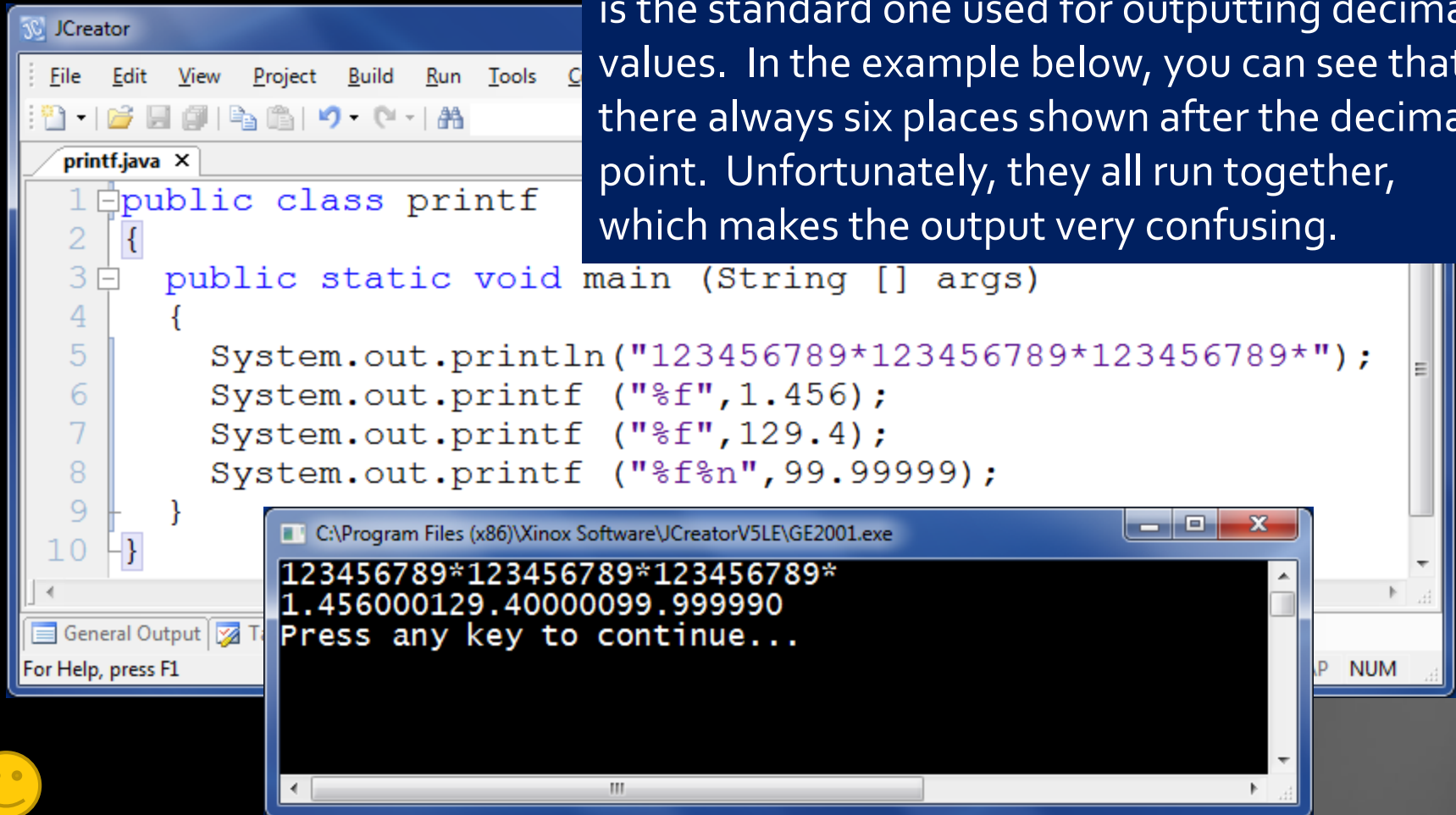
Below the code editor, the 'General Output' window displays the following output:

```
123456789*123456789*123456789*
123,456 (0001,294) (9,999,999)
Press any key to continue...
```



# Example #13 – Decimal values

As you saw in lesson 1C, the **%f** format specifier is the standard one used for outputting decimal values. In the example below, you can see that there always six places shown after the decimal point. Unfortunately, they all run together, which makes the output very confusing.



The screenshot shows the JCreator IDE with a file named `printf.java` open. The code is as follows:

```
1 public class printf
2 {
3     public static void main (String [] args)
4     {
5         System.out.println("123456789*123456789*123456789*");
6         System.out.printf ("%f",1.456);
7         System.out.printf ("%f",129.4);
8         System.out.printf ("%f%n",99.99999);
9     }
10 }
```

Below the code editor, the 'General Output' window displays the program's execution results:

```
123456789*123456789*123456789*
1.456000129.40000099.999990
Press any key to continue...
```

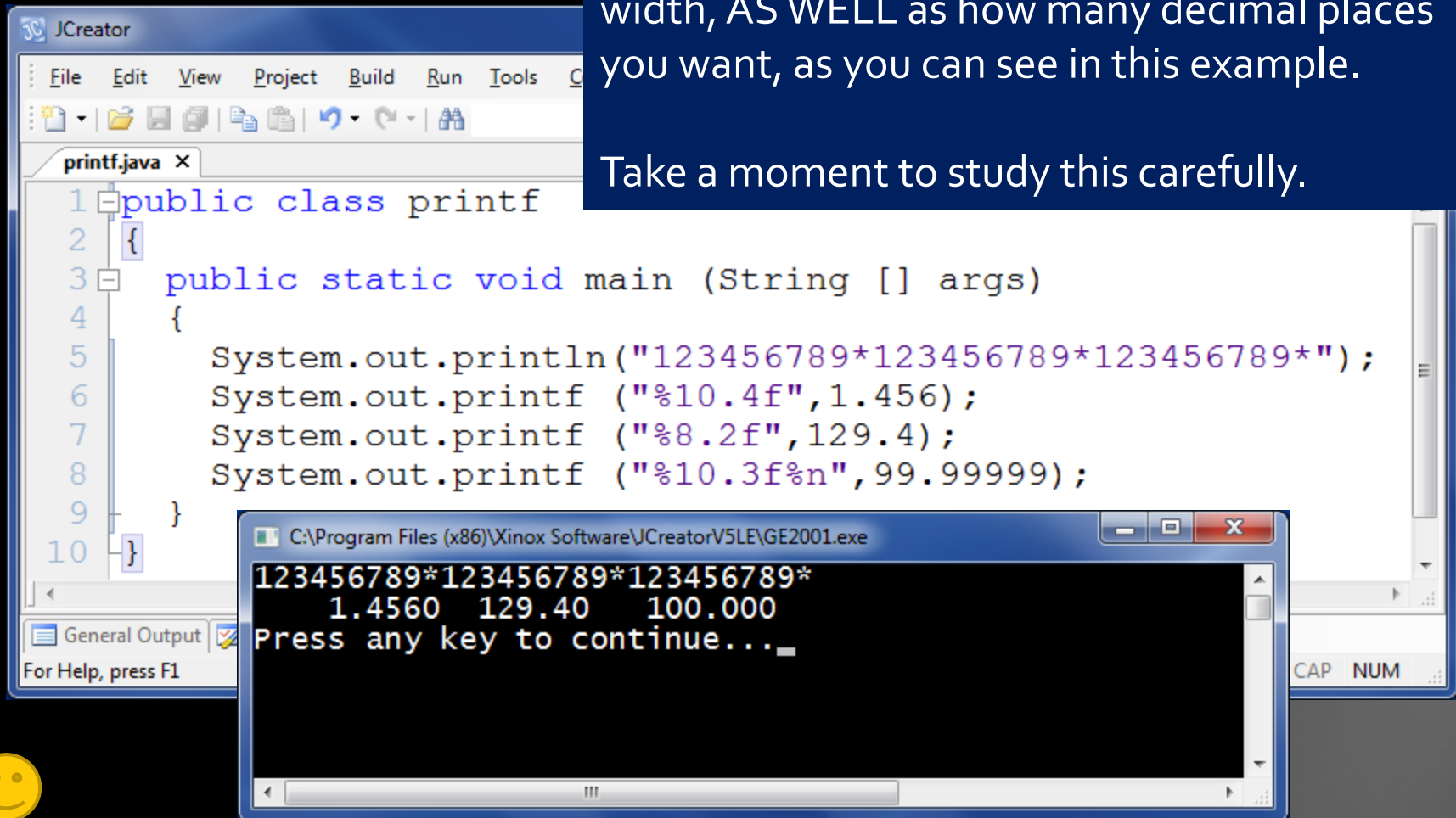
The output demonstrates that the `%f` format specifier prints decimal values with six digits after the decimal point, even if the original number has fewer digits, leading to a confusing concatenation of the results.



# Example #14 – Decimal values, formatted

Fortunately, it is also possible to control the field width, AS WELL as how many decimal places you want, as you can see in this example.

Take a moment to study this carefully.



The screenshot shows the JCreator IDE with a file named `printf.java` open. The code defines a public class `printf` with a `main` method that uses `System.out.println` and `System.out.printf` to display formatted output. The output window shows the results of these operations, including a multiplication string, three formatted floating-point numbers, and a prompt to press any key to continue.

```
1 public class printf
2 {
3     public static void main (String [] args)
4     {
5         System.out.println("123456789*123456789*123456789*");
6         System.out.printf ("%10.4f",1.456);
7         System.out.printf ("%8.2f",129.4);
8         System.out.printf ("%10.3f%n",99.99999);
9     }
10 }
```

General Output  
For Help, press F1

C:\Program Files (x86)\Xinox Software\JCreatorV5LE\GE2001.exe

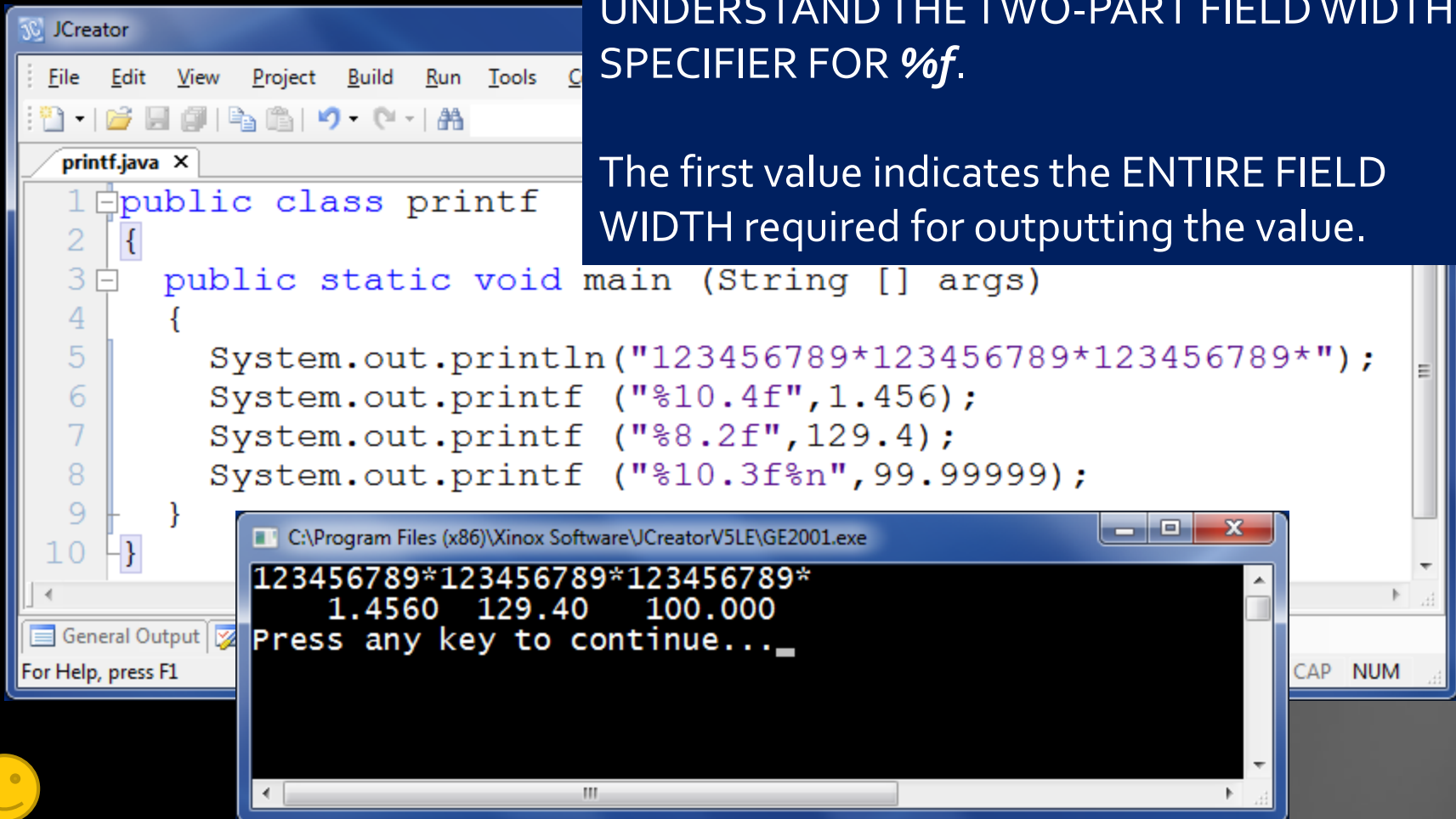
123456789\*123456789\*123456789\*  
1.4560 129.40 100.000  
Press any key to continue...



# Example #14 – Decimal values, formatted

IT IS ABSOLUTELY CRUCIAL THAT YOU UNDERSTAND THE TWO-PART FIELD WIDTH SPECIFIER FOR `%f`.

The first value indicates the ENTIRE FIELD WIDTH required for outputting the value.



The screenshot shows the JCreator IDE with a file named `printf.java` open. The code is as follows:

```
1 public class printf
2 {
3     public static void main (String [] args)
4     {
5         System.out.println("123456789*123456789*123456789*");
6         System.out.printf ("%10.4f",1.456);
7         System.out.printf ("%8.2f",129.4);
8         System.out.printf ("%10.3f%n",99.99999);
9     }
10 }
```

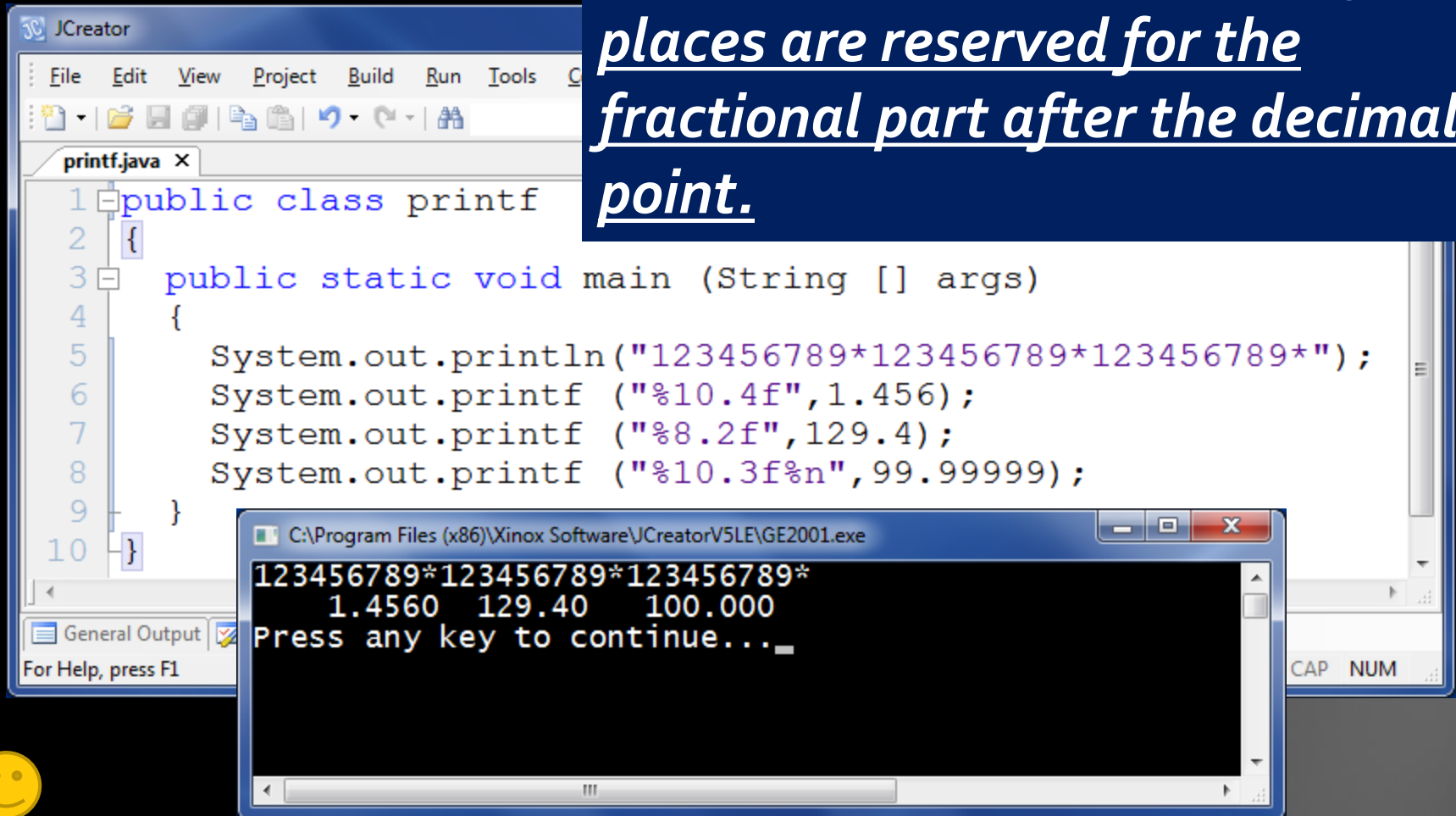
Below the code editor, the 'General Output' window displays the following output:

```
123456789*123456789*123456789*
      1.4560 129.40 100.000
Press any key to continue...
```



# Example #14 – Decimal values, formatted

The second value indicates *how many places are reserved for the fractional part after the decimal point.*



The screenshot shows the JCreator IDE with a file named `printf.java` open. The code defines a `public class printf` with a `main` method that uses `System.out` to print a string and three formatted decimal values. The output window shows the execution results: the string `123456789*123456789*123456789*` followed by the formatted values `1.4560`, `129.40`, and `100.000` on the same line. The prompt `Press any key to continue...` is also visible.

```
1 public class printf
2 {
3     public static void main (String [] args)
4     {
5         System.out.println("123456789*123456789*123456789*");
6         System.out.printf ("%10.4f",1.456);
7         System.out.printf ("%8.2f",129.4);
8         System.out.printf ("%10.3f%n",99.99999);
9     }
10 }
```

General Output  
For Help, press F1

C:\Program Files (x86)\Xinox Software\JCreatorV5LE\GE2001.exe

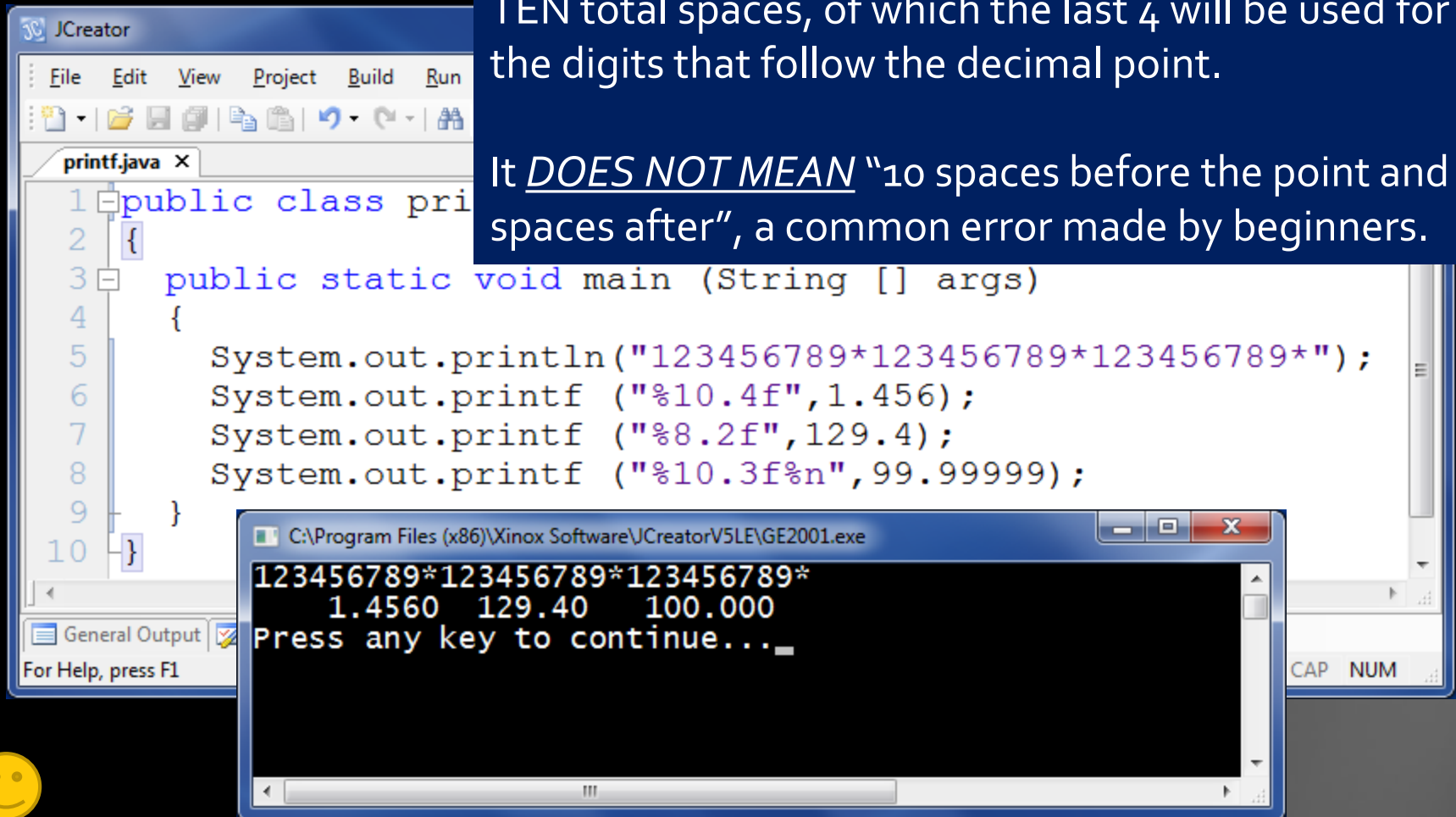
123456789\*123456789\*123456789\*  
1.4560 129.40 100.000  
Press any key to continue...



# Example #14 – Decimal values, formatted

A format specifier of “%10.4f” means that there are TEN total spaces, of which the last 4 will be used for the digits that follow the decimal point.

It DOES NOT MEAN “10 spaces before the point and 4 spaces after”, a common error made by beginners.



The screenshot shows the JCreator IDE with a file named `printf.java` open. The code is as follows:

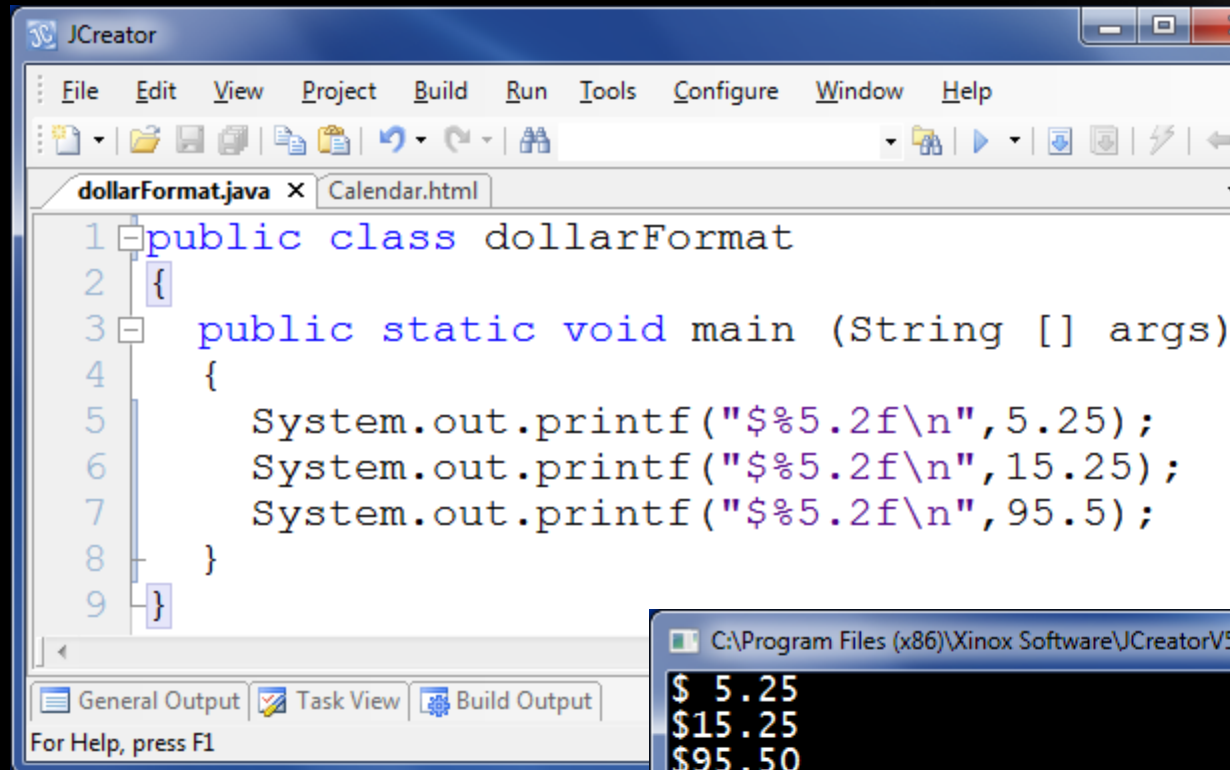
```
1 public class pri
2 {
3     public static void main (String [] args)
4     {
5         System.out.println("123456789*123456789*123456789*");
6         System.out.printf ("%10.4f",1.456);
7         System.out.printf ("%8.2f",129.4);
8         System.out.printf ("%10.3f%n",99.99999);
9     }
10 }
```

Below the code editor, the 'General Output' window is visible, showing the execution results:

```
123456789*123456789*123456789*
      1.4560 129.40 100.000
Press any key to continue...
```

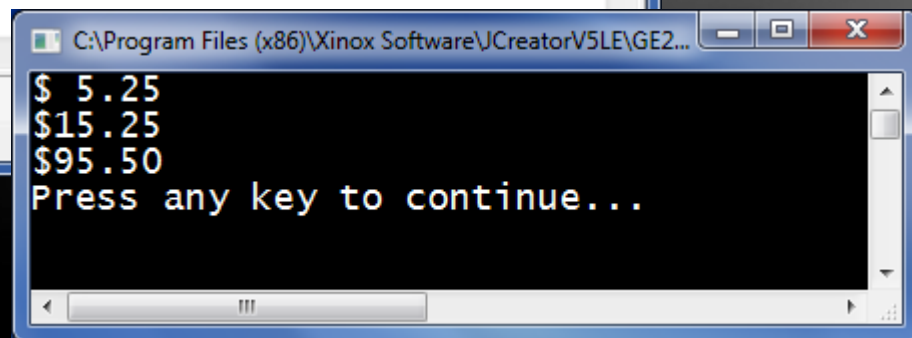


# Example #15 – Dollar formatting



```
1 public class dollarFormat
2 {
3     public static void main (String [] args)
4     {
5         System.out.printf("$%5.2f\n", 5.25);
6         System.out.printf("$%5.2f\n", 15.25);
7         System.out.printf("$%5.2f\n", 95.5);
8     }
9 }
```

The most obvious benefit of decimal value formatting is expressing dollar amounts in columns. Here is an example of this.

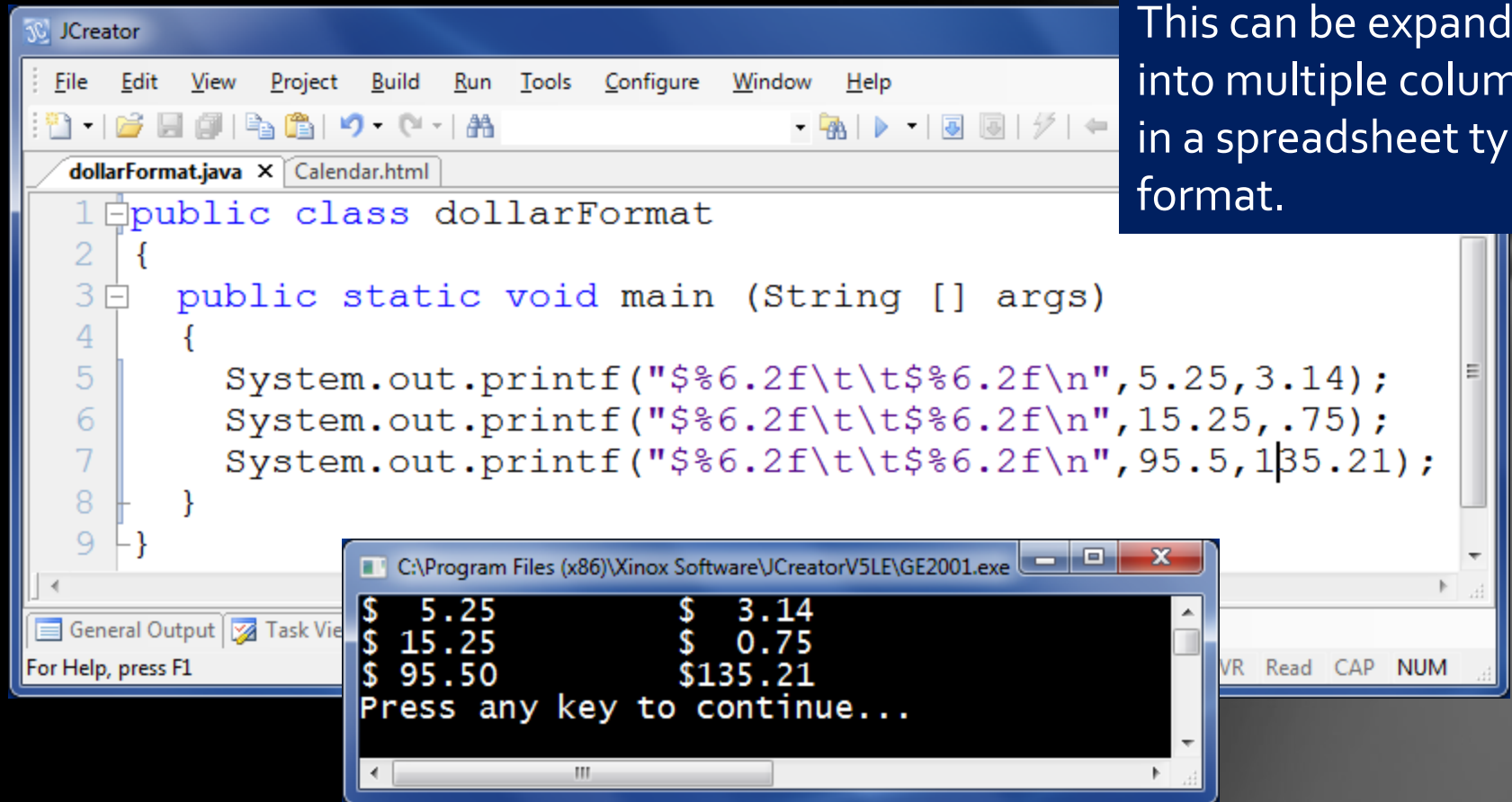


```
$ 5.25
$15.25
$95.50
Press any key to continue...
```



# Example #16 – Multiple columns

This can be expanded into multiple columns in a spreadsheet type format.



The screenshot shows the JCreator IDE with a Java file named `dollarFormat.java`. The code uses `printf` to format floating-point numbers with a dollar sign and two decimal places, separated by a tab character. The output window shows the results of these formatted strings, displaying three rows of data in a two-column format. The output window title is `C:\Program Files (x86)\Xinox Software\JCreatorV5LE\GE2001.exe`.

```
1 public class dollarFormat
2 {
3     public static void main (String [] args)
4     {
5         System.out.printf("$%6.2f\t\t$%6.2f\n", 5.25, 3.14);
6         System.out.printf("$%6.2f\t\t$%6.2f\n", 15.25, .75);
7         System.out.printf("$%6.2f\t\t$%6.2f\n", 95.5, 135.21);
8     }
9 }
```

General Output Task View  
For Help, press F1

C:\Program Files (x86)\Xinox Software\JCreatorV5LE\GE2001.exe

```
$  5.25      $  3.14
$ 15.25      $  0.75
$ 95.50     $135.21
Press any key to continue...
```

VR Read CAP NUM





# Calendar class suffixes

- Let's return to the Calendar class that was introduced in the last lesson.
- The format specifier for this class is %t, and is followed by a suffix character, like 'c' or 'a'
- As mentioned before, there are over 30 of these suffixes.



# Calendar – more features

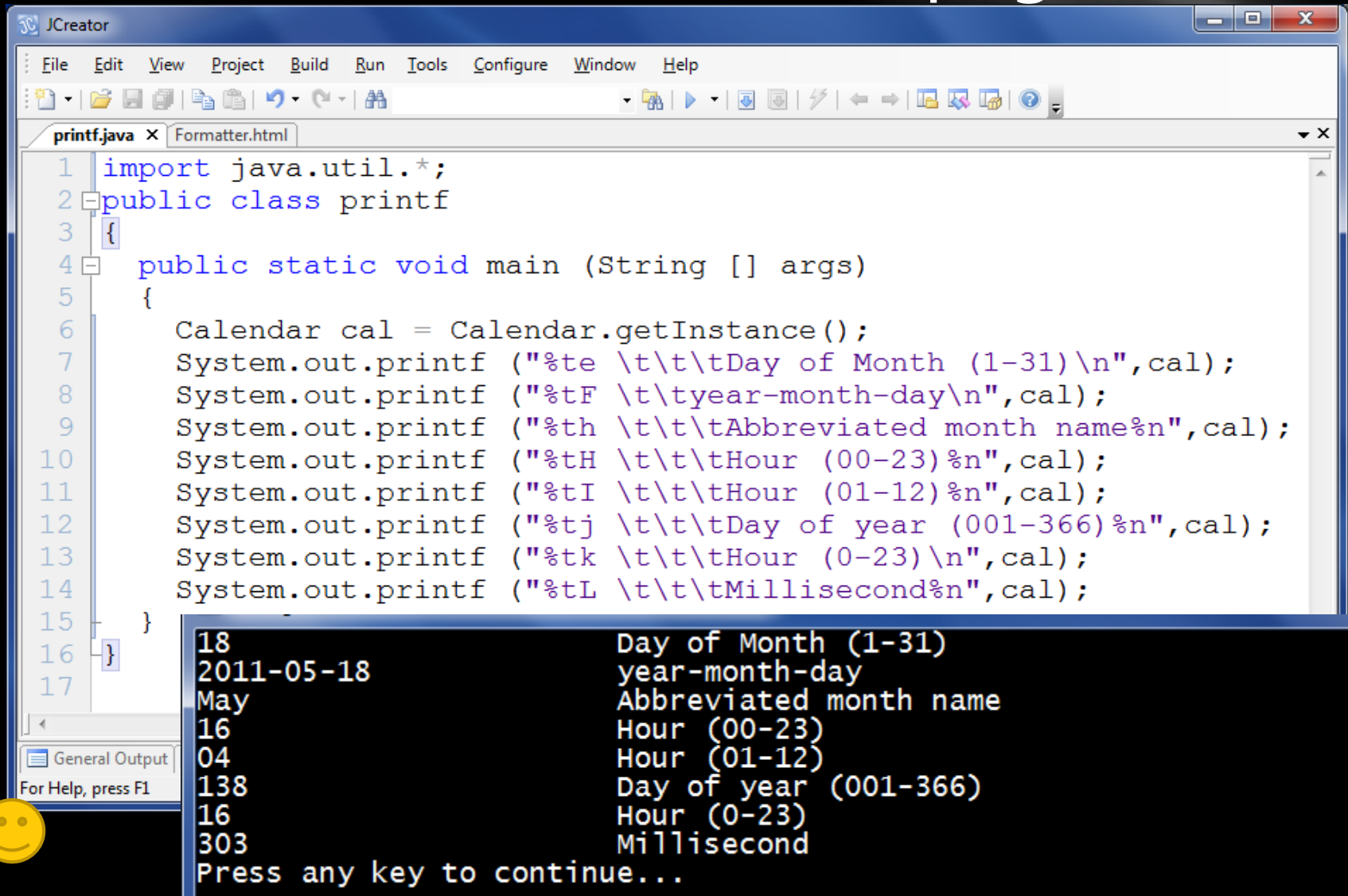
Here is a detailed list of more of the Calendar suffixes that work with the `%t` format specifier.

```
1 import java.util.*;
2 public class printf
3 {
4     public static void main (String [] args)
5     {
6         Calendar cal = Calendar.getInstance();
7         System.out.printf ("%ta \t\t\t\tAbbreviated weekday name\n", cal);
8         System.out.printf ("%tA \t\t\t\tFull weekday name\n", cal);
9         System.out.printf ("%tb \t\t\t\tAbbreviated month name\n", cal);
10        System.out.printf ("%tB \t\t\t\tFull month name\n", cal);
11        System.out.printf ("%tc \tDate and time "
12                            +"hh:mm:ss timeZone year\n", cal);
13        System.out.printf ("%tC \t\t\t\tFirst two digits of the year\n", cal);
14        System.out.printf ("%td \t\t\t\tDay of month (01-31)\n", cal);
15        System.out.printf ("%tD \t\t\t\tmonth/day/year\n", cal);
```

```
C:\Program Files (x86)\Xinox Software\JCreatorV5LE\GE2001.exe
Wed           Abbreviated weekday name
Wednesday    Full weekday name
May           Abbreviated month name
May           Full month name
Wed May 18 10:53:44 CDT 2011 Date and time hh:mm:ss timeZone year
20            First two digits of the year
18            Day of month (01-31)
05/18/11      month/day/year
Press any key to continue..._
```



# Calendar – more features, page 2



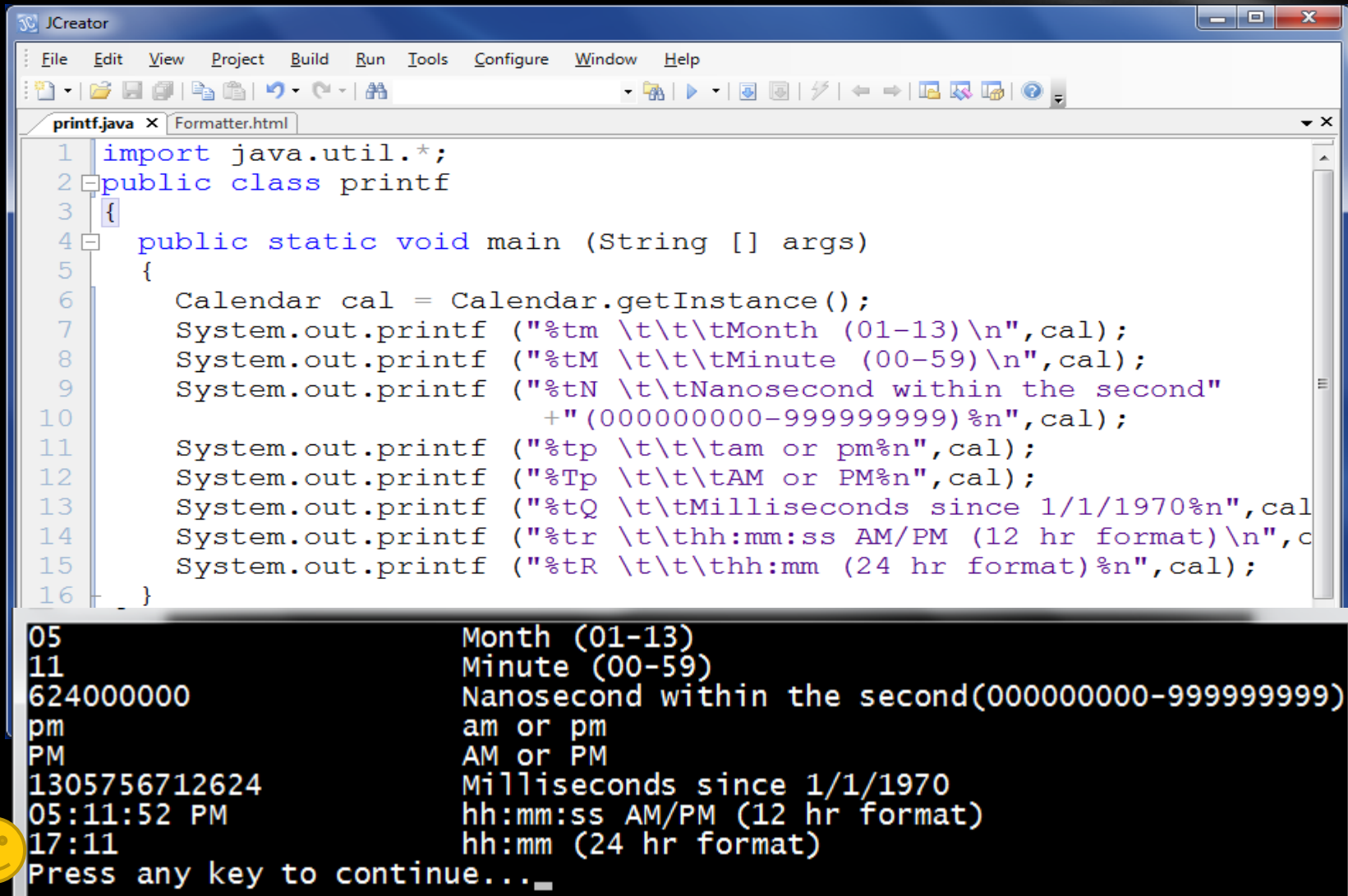
```
1 import java.util.*;
2 public class printf
3 {
4     public static void main (String [] args)
5     {
6         Calendar cal = Calendar.getInstance();
7         System.out.printf ("%te \t\t\tDay of Month (1-31)\n",cal);
8         System.out.printf ("%tF \t\t\tyear-month-day\n",cal);
9         System.out.printf ("%th \t\t\tAbbreviated month name\n",cal);
10        System.out.printf ("%tH \t\t\tHour (00-23)\n",cal);
11        System.out.printf ("%tI \t\t\tHour (01-12)\n",cal);
12        System.out.printf ("%tj \t\t\tDay of year (001-366)\n",cal);
13        System.out.printf ("%tk \t\t\tHour (0-23)\n",cal);
14        System.out.printf ("%tL \t\t\tMillisecond\n",cal);
15    }
16 }
17
```

General Output  
For Help, press F1

```
18          Day of Month (1-31)
2011-05-18  year-month-day
May        Abbreviated month name
16         Hour (00-23)
04         Hour (01-12)
138        Day of year (001-366)
16         Hour (0-23)
303        Millisecond
Press any key to continue...
```



# Calendar – more features, page 3



The image shows a screenshot of the JCreator IDE. The main window displays the source code for a Java class named `printf`. The code imports `java.util.*` and uses the `Calendar` class to format and print the current date and time using various `printf` format specifiers. The output of the program is shown in a separate window at the bottom, displaying the results of each `printf` statement. A yellow smiley face icon is visible in the bottom-left corner of the output window.

```
1 import java.util.*;
2 public class printf
3 {
4     public static void main (String [] args)
5     {
6         Calendar cal = Calendar.getInstance();
7         System.out.printf ("%tm \t\t\tMonth (01-13)\n",cal);
8         System.out.printf ("%tM \t\t\tMinute (00-59)\n",cal);
9         System.out.printf ("%tN \t\t\tNanosecond within the second"
10                             +"(000000000-999999999)%n",cal);
11         System.out.printf ("%tp \t\t\tam or pm%n",cal);
12         System.out.printf ("%Tp \t\t\tAM or PM%n",cal);
13         System.out.printf ("%tQ \t\t\tMilliseconds since 1/1/1970%n",cal);
14         System.out.printf ("%tr \t\t\t hh:mm:ss AM/PM (12 hr format)\n",cal);
15         System.out.printf ("%tR \t\t\t hh:mm (24 hr format)%n",cal);
16     }
```

05 Month (01-13)  
11 Minute (00-59)  
624000000 Nanosecond within the second(000000000-999999999)  
pm am or pm  
PM AM or PM  
1305756712624 Milliseconds since 1/1/1970  
05:11:52 PM hh:mm:ss AM/PM (12 hr format)  
17:11 hh:mm (24 hr format)  
Press any key to continue...

# Calendar – more features, page 4

The screenshot shows the JCreator IDE with two tabs: "printf.java" and "Formatter.html". The "printf.java" tab is active, displaying the following code:

```

1 import java.util.*;
2 public class printf
3 {
4     public static void main (String [] args)
5     {
6         Calendar cal = Calendar.getInstance();
7         System.out.printf ("%ts \t\tSeconds since 1/1/1970\n",cal);
8         System.out.printf ("%tS \t\t\tSeconds (00-60)\n",cal);
9         System.out.printf ("%tT \t\t\t hh:mm:ss (24 hour format)\n",cal);
10        System.out.printf ("%ty \t\t\tYear without century(00-99)\n",cal);
11        System.out.printf ("%tY \t\t\tYear with century (0001-9999)\n",cal);
12        System.out.printf ("%ts \t\tOffset from UTC\n",cal);
13        System.out.printf ("%tZ \t\t\tTime zone name\n",cal);
14    }
15 }

```

Below the code editor, there is a black console window showing the output of the program. The output consists of two columns of text, each preceded by a number corresponding to the line number of the printf statement in the code.

1305757046	Seconds since 1/1/1970
26	Seconds (00-60)
17:17:26	hh:mm:ss (24 hour format)
11	Year without century(00-99)
2011	Year with century (0001-9999)
1305757046	offset from UTC
CDT	Time zone name



# Calendar class – more information

- For a more in-depth study of the Calendar class, look in the JAVA help files, or API, available from the [java.sun.com](http://java.sun.com) website.



# Lesson Summary

- This lesson introduced advanced features of the *printf* command specifically dealing with field width and various format flags.
- It also showed the many Calendar suffixes available with printf, producing various date and time output formats.





# Calendar Lab

Write a program using the Calendar class to output a summary of your birthday, just like the one you see on the next slide. The output should match in format exactly, except for the information, of course. The first part of the program is shown to help get you started.

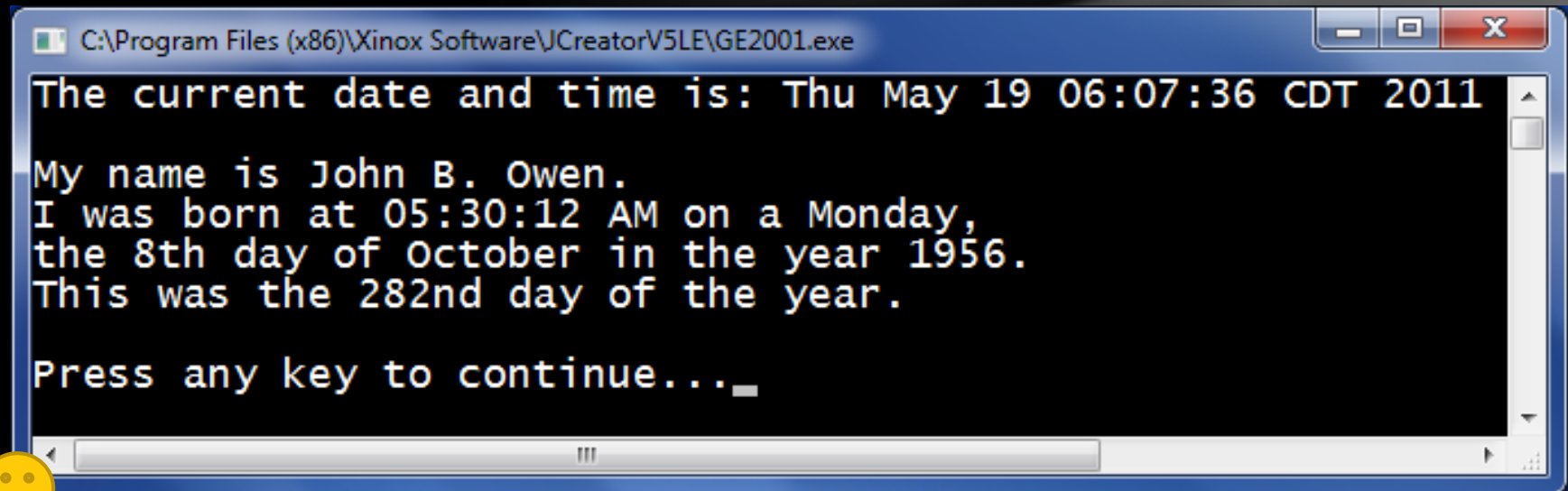




# Calendar Lab

Below is the required output and a portion of the code that produced it. Use *your information* instead of mine. If you don't know the exact time you were born, just make up something.

```
import java.util.*;
public class printf
{
    public static void main (String [] ar
    {
        Calendar cal = Calendar.getInstance();
        System.out.printf("The current date and time is: %tc\n\n",cal);
        cal.set(1956,9,8,5,30,12);
        String name = "John B. Owen";
        System.out.printf ("My name is %s.  \nI was born at %Tr "
            +"on a %tA, \nthe %teth ",name,cal,cal,cal);
    }
}
```



C:\Program Files (x86)\Xinox Software\JCreatorV5LE\GE2001.exe

The current date and time is: Thu May 19 06:07:36 CDT 2011

My name is John B. Owen.

I was born at 05:30:12 AM on a Monday,  
the 8th day of October in the year 1956.  
This was the 282nd day of the year.

Press any key to continue...

# Calendar Lab

Study the program carefully and look back in the lesson for the correct suffixes to produce the rest of the output. Have fun!

```
import java.util.*;
public class printf
{
    public static void main (String [] args)
    {
        Calendar cal = Calendar.getInstance();
        System.out.printf("The current date and time is: %tc\n\n",cal);
        cal.set(1956,9,8,5,30,12);
        String name = "John B. Owen";
        System.out.printf ("My name is %s.  \nI was born at %Tr "
            +"on a %tA, \nthe %teth ",name,cal,cal,cal);
    }
}
```

C:\Program Files (x86)\Xinox Software\JCreatorV5LE\GE2001.exe

The current date and time is: Thu May 19 06:07:36 CDT 2011

My name is John B. Owen.

I was born at 05:30:12 AM on a Monday,  
the 8th day of October in the year 1956.  
This was the 282nd day of the year.

Press any key to continue...



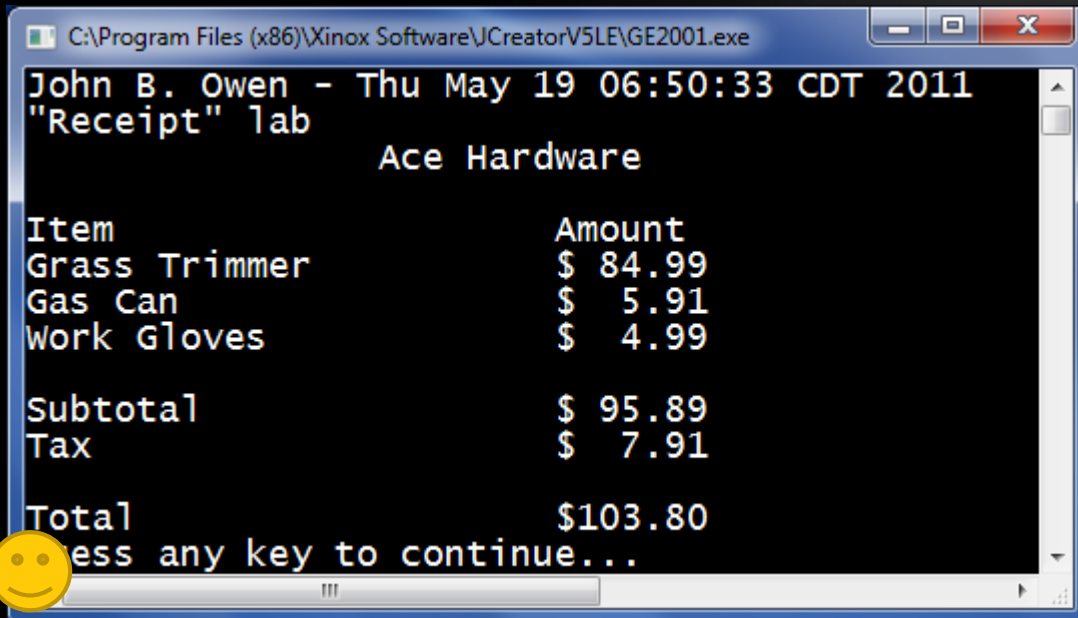
# Receipt Lab

Write a program that outputs a receipt from a store, with at least two but no more than three items. The output should resemble the receipt as closely as possible, with all dollar amounts preceded by the '\$' sign and properly aligned at the decimal point. Include subtotal, tax, and grand total at the bottom.



# Receipt Lab

```
import java.util.*;
public class receiptLab
{
    public static void main (String [] args)
    {
        Calendar cal = Calendar.getInstance();
        System.out.printf("John B. Owen - %tc\n",cal);
        System.out.println("\n\"Receipt\" lab");
        System.out.println("\t\tAce Hardware\n");
        System.out.println("Item\t\t\tAmount");
        System.out.printf("%s\t\t\t$%6.2f\n", "Grass Trimmer",84.99);
        System.out.printf("%s\t\t\t$%6.2f\n", "Gas Can",5.91);
        System.out.printf("%s\t\t\t$%6.2f\n\n", "Work Gloves",4.99);
        System.out.printf("%s\t\t\t$%6.2f\n", "Subtotal",
                           84.99+5.91+4.99);
    }
}
```



```
C:\Program Files (x86)\Xinox Software\JCreatorV5LE\GE2001.exe
John B. Owen - Thu May 19 06:50:33 CDT 2011
"Receipt" lab
Ace Hardware
Item          Amount
Grass Trimmer $ 84.99
Gas Can       $  5.91
Work Gloves   $  4.99
Subtotal      $ 95.89
Tax           $  7.91
Total         $103.80
Press any key to continue...
```

Here is a portion of the program that produced my receipt lab. Study it carefully and adapt it to your own receipt. **Notice how you can “do math” inside the parameter list!** Be sure to include the “timestamp” beside your name.

# CONGRATULATIONS!

- You can now control field width and decimal output, as well as use various Calendar formats in your output!
- *It is now time to move on to Lesson 2, which talks about data types and variables in much greater detail.*



# Thanks, and have fun!



To order supplementary materials for all the lessons in this package, including lesson examples, lab solutions, quizzes, tests, and unit reviews, visit the [O\(N\)CS Lessons](#) website, or contact me at

[John B. Owen](#)  
[captainjbo@gmail.com](mailto:captainjbo@gmail.com)



8/21/2015