L30

Friday, January 28, 2022 1:18 AM

Text Formatting

Q1. Java provides support for printf style formatting of strings. **Printf** style formatting contains a **format string** and a number of different data type **parameters**. The **format string** contains **plain text** as well as **format specifiers**, which are special characters that format the parameters. The function produces an output string with parameters applied as per the format specifiers.

For example:

System.out.format(**"Fees = %f, Age = %d and Name = %s"**, **12.95**, **24**, **"Raj"**);
produces an output of : **Fees = 12.950000, Age = 24 and Name = Raj**

In the above example,

- the format string is : "Fees = %f, Age = %d and Name = %s"
- the format specifiers are : %f, %d and %s
- the parameters are **12.95**, **24** and **"Raj"**

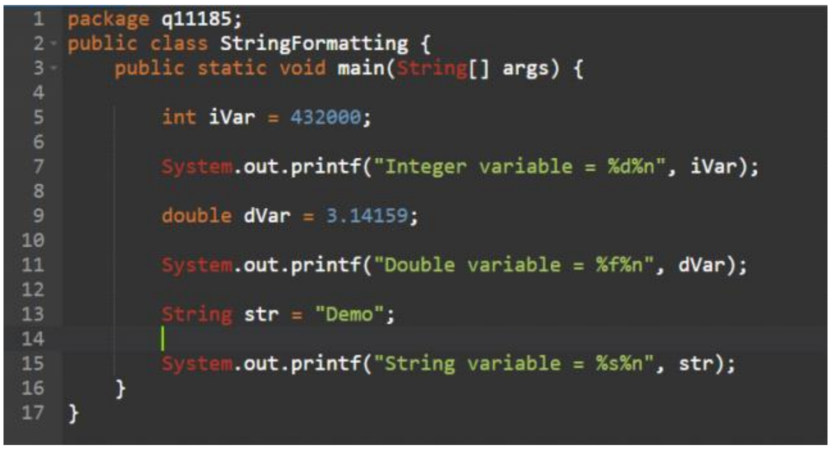
The format(...) method formatted the **parameter values** as per the **format specifiers** and produced the output string.

Java's PrintStream.format(String format, Object... args), PrintStream.printf(String format, Object... args) and String.format(String format, Object... args) methods are equivalent methods that provide this functionality.

There are many converters, flags and specifiers, which are documented in java.util.Formatter class.

For example, if you want to know about the formatting character **%n**, click on the above link to **Formatter** and go to the **formatting character conversion table** under **Conversions** section.

We will learn more about formatting in the later sections. Retype the code below and submit.



```
package q11185;
public class StringFormatting {
    public static void main(String[] args) {

        int iVar = 432000;

        System.out.printf("Integer variable = %d%n", iVar);

        double dVar = 3.14159;

        System.out.printf("Double variable = %f%n", dVar);

        String str = "Demo";

        System.out.printf("String variable = %s%n", str);
    }
}
```

Q2. The format specifier for integers is %d. For float or doubles, it is %f. For String, the format specifier is %s. To put a new line character in the string, we should use %n.

Some additional flags can be specified to these data types.

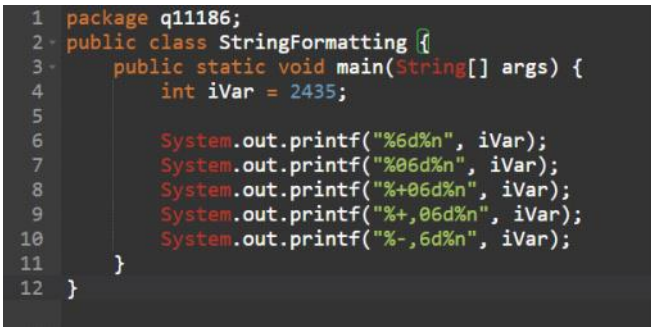
A number after %. Example: **%8d**	Means the output value should have a minimum of 8 characters for width. If the number is smaller, it puts spaces in the front .
Zero and a number after %. Example: **%08d**	Same as above, but instead of spaces, it puts zeros in the front.
+ Example: **%+d**	Shows a + or - sign in the output depending on the parameter value.
, Example: **%,d**	Formats the output with commas (or any other locale specific grouping characters).
- Example: **%-8d**	Left justifies the output value. Should be used in combination with a number.
. Example: **%.3f** or **%8.3f**	For displaying decimal points. Should be followed by a number which indicates the number of decimal points.

Invalid combination of the formatters will result in an error.
For example, %.3d is invalid because it is trying to specify 3 decimal places for integer data type.

Similarly if the combination of the formatter and the data is not matching, Java flags an error.
For example, if we specify %f for integer data type, Java will flag an error.

See and retype the code below.

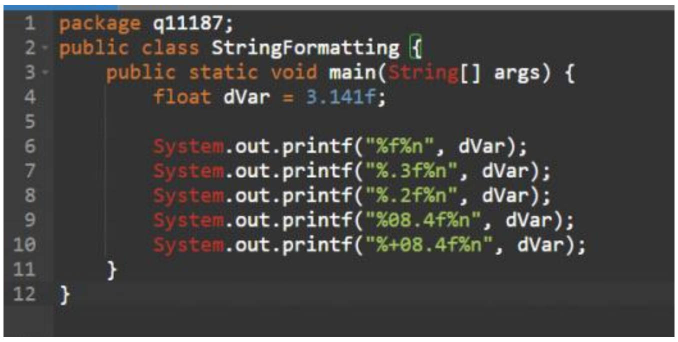
```
//  2435
// 002435
// +02435
// +2,435
// 2,435
```



```java
package q11186;
public class StringFormatting {
    public static void main(String[] args) {
        int iVar = 2435;

        System.out.printf("%6d%n", iVar);
        System.out.printf("%06d%n", iVar);
        System.out.printf("%+06d%n", iVar);
        System.out.printf("%+,06d%n", iVar);
        System.out.printf("%-,6d%n", iVar);
    }
}
```

Q3. See and retype the code below.



```java
package q11187;
public class StringFormatting {
    public static void main(String[] args) {
        float dVar = 3.141f;

        System.out.printf("%f%n", dVar);
        System.out.printf("%.3f%n", dVar);
        System.out.printf("%.2f%n", dVar);
        System.out.printf("%08.4f%n", dVar);
        System.out.printf("%+08.4f%n", dVar);
    }
}
```

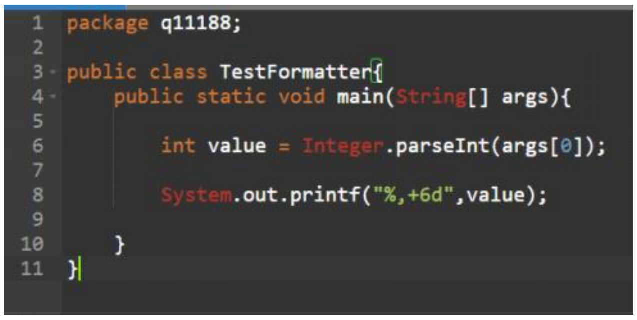
Q4. Create a class TestFormatter with a **main** method. The method receives one command line argument, convert it to an integer.

Format and print the integer such that it has comma (or locale specific) delimiters, has a minimum width of 6 characters (put spaces in the front if the number is smaller), and has a sign indicator (+ or -).

For example:
Cmd Args : 12
…+12

Cmd Args : 345
..+345

[Note: You need not use **%n** at the end of your formatting text.]



```java
package q11188;

public class TestFormatter{
    public static void main(String[] args){

        int value = Integer.parseInt(args[0]);

        System.out.printf("%,+6d",value);

    }
}
```

Q5. Create a class TestFormatter with a **main** method. The method receives one command line argument.

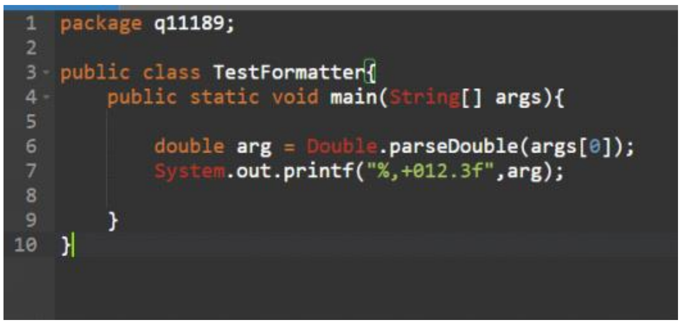
Convert the argument received in the main method into a double value.

Format and print the double value such that it has :
1. comma (or locale specific) delimiters
2. a minimum width of 12 characters (put numbers in the front if the number is smaller)
3. only 3 decimal points
4. a sign indicator (+ or -)
5. instead of spaces put zeros(**0**'s) to the left if the number is smaller

For example:
Cmd Args : 2000
+002,000.000

Cmd Args : 320.5
+0000320.500

[Note: You need not use **%n** at the end of your formatting text.]



```java
package q11189;

public class TestFormatter{
    public static void main(String[] args){

        double arg = Double.parseDouble(args[0]);
        System.out.printf("%,+012.3f",arg);

    }
}
```