Theory: Formatted output

9 minutes 0 / 5 problems solved

§1. Introduction

You are already familiar with `System.out.print()` and `System.out.println()` methods to print output to the console. But when you need complex **formatting** of output, these two methods are not really helpful. There are two methods that you can use in such cases. Those are `System.out.printf()` and `String.format()`. Let's discuss them in detail and focus on string and number formatting.

§2. Introducing printf() method

The `printf()` method usually has two parts. First, you give the string you want to format as the first attribute. This string itself includes rules to format it via **format specifiers**. Some examples of format specifiers are `%d`, `%s`, etc. In the second part, you give the argument list that Java can use to format the string according to format-specifiers.

See the following example to understand the different parts of the `printf()` function.

```
System.out.printf("My Name is %s. I was born in %d", "Mike", 1998);
```

- The first part is `"My Name is %s. I was born in %d"`, where `%s` and `%d` are the format specifiers.
- The second part is the argument list: `"Mike", 1998` .

§3. Different use cases of printf()

It's time to know the different use cases of format-specifiers. It's quite easy to understand it with code samples. Let's try now.

You can display an integer with `%d` format specifier.

```
public static void main(String[] args){
    System.out.printf("Display a Number %d", 15000);
}
```

Java will replace 15000 in place of `%d`. The output of the above code is `Display a Number 15000`.

if you want several integers to display in the output, use several `%d` specifiers.

```
public static void main(String[] args){
    System.out.printf("Sum of %d and %d is %d", 15, 40, 55);
}
```

Java will replace each argument in place of `%d` respectively starting from left. The output of the above code is `Sum of 15 and 40 is 55`.

If you want to display a floating-point value, use `%f` specifier.

```
public static void main(String[] args){
    System.out.printf("Display a Number %f", 15.23);
}
```

Similar to the above cases, Java will replace 15.23 in place of `%f`. The above code will produce the following output. `Display a Number 15.230000` Although it's technically correct, it looks ugly. You don't want so many trailing zeros. You can set **precision** with `printf()` method.

Current topic:

Formatted output

Topic depends on:

✓ String — Stage 2

Topic is required for:

Writing files

Table of contents:

```
public static void main(String[] args){
    System.out.printf("Display a Number %.2f", 15.23);
}
```

`.2f` decided that the number of digits that should appear after the **decimal** place is two. The code given above will output `Display a Number 15.23`

Similarly, you can display Characters and Strings with `printf()` method. Take a look at the following code. If you want to print a character, use `%c`, and if you want to print a String, use `%s`.

```
public static void main(String[] args){
    char abbr = 'H';
    String element = "Hydrogen";
    System.out.printf("%c stands for %s", abbr, element);
}
```

When this code runs, the value of the `abbr` variable will replace `%c` and value of the `element` will replace `%s`. The output of the above code is `H stands for Hydrogen`. That's all about `printf()` method. Let's move on to learn `String.format()` method.

§4. String.format() Method

The `format()` method in the `String` class works very much like `printf()` method. The main difference here is that you return a string instead of printing it. Let's see several examples.

The following code formats an integer using it.

```
public static void main(String[] args){
    int age = 22;
    String str = String.format("My age is %d", age);
    System.out.println(str);
}
```

When you execute this code, Java will create a String called `str` by concatenating `My age is` with the value of the `age` variable. Then it will print the value of `str`. The output is:

```
My age is 22
```

Similarly, you can format other data types as well. See the following code.

```
public static void main(String[] args){
    int age = 22;
    char initial = 'M';
    String surname = "Anderson";
    double height = 1.72;


   String details = String.format("My name is %c. %s.%nMy age is %d.%nMy height i
s %.2f.", initial, surname, age, height);
    System.out.println(details);
}
```

You can see that we have used four types of data types in this example. Java will replace `%c, %s, %d, %f` with `initial`, `surname`, `age`, and `height` respectively. `%n` **newline** character breaks the line every time it was used. The output of our code is:

```
My name is M. Anderson.
My age is 22.
My height is 1.72.
```

§5. Summary

Let's summarize what we've learned in this topic.

Format-Specifier	datatype related	Format printing	Formatting a string
%d	int, short, byte, long	System.out.printf("Display a Integer %d",15000);	String.format("Display a Integer %d",15000)
%c	char	System.out.printf("Display a Character %c",'c');	String.format("Display a Character %c",'c')
%f	double, float	System.out.printf("Display a Floating-point Number %f",123.45);	String.format("Display a Floating-point Number %f",123.45)
%s	String	System.out.printf("Display a String %s","String");	String.format("Display a String %s","String")

§6. Conclusion

When you are writing complex applications combining strings and variables with `+` sign is not recommended. The `printf()` and `format()` methods are specifically created for that. Both these methods work exactly the same way apart from `printf()` method prints the output while `format()` method returns a `String`. In this article, you have learned most of the use cases of both the functions. Enjoy it.