Convenience classes for working with strings in Java

1. `StringTokenizer` class

And now a few more common scenarios involving working with strings. How do you split a string into several parts? There are several ways to do this.

`split()` method

The first way to split a string into multiple parts is to use the `split()` method. A regular expression that defines a special delimiting string must be passed as an argument. You will learn what a regular expression is in the *Java Multithreading* quest.

Example:

Code	Result

`String str = "Good news everyone!";`
`String[] strings = str.split("ne");`
`System.out.println(Arrays.toString(strings));`
	The result will be an array of three strings:
	`["Good ", "ws everyo", "!"]`

Simple, but sometimes this approach is excessive. If there are a lot of delimiters (for example, spaces, newline characters, tabs, periods), then you have to construct a rather complex regular expression. It is difficult to read and therefore difficult to modify.

`StringTokenizer` class

Java has a special class whose whole job is to split a string into substrings.

This class doesn't use regular expressions: instead, you simply pass in a string consisting of delimiters. The advantage of this approach is that it does not break the entire string into pieces all at once, but instead moves from beginning to end one step at a time.

The class has a constructor and two important methods. We pass the constructor a string that we split into parts, and a string comprised of a set of delimiting characters.

Methods	Description
`String nextToken()`	Returns the next substring
`boolean hasMoreTokens()`	Checks whether there are more substrings.

This class is somehow reminiscent of the Scanner class, which also has `nextLine()` and `hasNextLine()` methods.

You can create a `StringTokenizer` object with this command:

```
StringTokenizer name = new StringTokenizer(string, delimiters);
```

Where `string` is the string to be divided into parts. And `delimiters` is a string, and each character in it is treated as a delimiter. Example:

Code	Console output

`String str = "Good news everyone!";`
 	
`StringTokenizer tokenizer = new StringTokenizer(str,"ne");`
`while (tokenizer.hasMoreTokens())`
`{`
`  String token = tokenizer.nextToken();`
`  System.out.println(token);`
`}`
	Good
	ws
	v
	ryo
	!

Note that each character in the string passed as the second string to the `StringTokenizer` constructor is considered a separator.

2. `String.format()` method and `StringFormatter` class

Another interesting method of the String class is `format()`.

Let's say you have various variables storing data. How do you display them on the screen in one line? For example, we have some data (left column) and desired output (right column):

Code	Console output

`String name = "Amigo";`
`int age = 12;`
`String friend = "Diego";`
`int weight = 200;`
	`User = {name: Amigo, age: 12 years, friend: Diego, weight: 200 kg.}`

```
String name = "Amigo";
int age = 12;
String friend = "Diego";
int weight = 200;

System.out.println("User = {name: " + name + ", age:" + age
+ " years, friend: " + friend+", weight: " + weight + "
kg.}");
```

Such code is not very readable. And if the variable names were longer, then the code would become even more difficult:

```
class User {
    ......
    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public List<String> getFriends() {
        return friends;
    }

    public ExtraInformation getExtraInformation() {
        return extraInformation;
    }
}

User user = new User();

System.out.println("User = {name: " + user.getName() + ",
age:" + user.getAge() + " years, friend: " +
user.getFriends().get(0) + ", weight: " +
user.getExtraInformation().getWeight() + " kg.}");
```

Not very readable, is it?

But this is a common situation in real-world programs, so I want to tell you about a way to write this code more simply and more concisely.

String.format

The String class has a static `format()` method: it lets you specify a pattern for assembling a string with data. The general appearance of the command is as follows:

```
String name = String.format(pattern, parameters);
```

Code	Result
`String.format("Age=%d, Name=%s", age, name);`	`Age=12, Name=Amigo`
`String.format("Width=%d, Height=%d", width, height);`	`Width=20, Height=10`
`String.format("Fullname=%s", name);`	`Fullname=Diego`

The `format()` method's first parameter is a format string that contains all the desired text along with special characters called format specifiers (such as `%d` and `%s`) in the places where you need to insert data.

The `format()` method replaces these `%s` and `%d` format specifiers with the parameters that follow the format string in the parameter list. If we want to insert a string, then we write `%s`. If we want to insert a number, then the format specifier is `%d`. Example:

Code	Result
`String s = String.format("a=%d, b=%d, c=%d", 1, 4, 3);`	`s` is equal to `"a=1, b=4, c=3"`

Here is a short list of format specifiers that can be used inside the format string:

Specifier	Meaning
%s	`String`
%d	interger: `byte`, `short`, `int`, `long`
%f	real number: `float`, `double`
%b	`boolean`
%c	`char`
%t	`Date`
%%	`%` character

These specifiers indicate the type of data, but there are also specifiers that indicate the order of the data. To get an argument by its number (the

numbering starts from one), you need to write "%1$d" instead of "%d". Example:

These specifiers indicate the type of data, but there are also specifiers that indicate the order of the data. To get an argument by its number (the numbering starts from one), you need to write "%1$d" instead of "%d". Example:

Code	Result
`String s = String.format("a=%3$d, b=%2$d, c=%d", 11, 12, 13);`	`s` is equal to `"a=13, b=12, c=11"`

`3$d` will get the 3rd argument, `%2$d` will get the second argument, and `%d` will get the very first argument. The `%s` and `%d` format specifiers refer to arguments regardless of specifiers like `%3$d` or `%2$s`

3. String Pool

Every string specified in code as a string literal is stored in an area of memory called the `StringPool` while the program is running. **`StringPool`** is a special array for storing strings. Its purpose is to optimize string storage:

First, the strings specified in code must be stored somewhere, right? Code consists of commands, but data (especially, large strings) must be stored in memory separately from the code. Only references to string objects appear in code.

Second, all identical string literals must be stored in memory only once. And that's just how it works. When your class code is loaded by the Java machine, all string literals are added to the `StringPool` if they are not already there. If they are already there, then we simply use a string reference from the `StringPool`.

Accordingly, if you assign the same literal to several `String` variables in your code, then these variables will contain the same reference. A literal will be added to the `StringPool` only once. In all other cases, the code will get a reference to the string already loaded in the `StringPool`.

Here's roughly how it works:

Code	Working with the StringPool

`String a = "Hello";`
`String b = "Hello";`
`String c = "Bye";`
	`String[] pool = {"Hello", "Bye"};`
	`a = pool[0];`
	`b = pool[0];`
	`c = pool[1];`

That is why the a and b variables will store the same references.

`intern()` method

And the best part is that you can programmatically add any string to the `StringPool`. To do this, you just need to call the `String` variable's `intern()` method.

The `intern()` method will add the string to the `StringPool` if it is not already there, and will return a reference to the string in the `StringPool`.

If two identical strings are added to the `StringPool` using the `intern()`method, the method returns the same reference. This can be used to compare strings by reference. Example:

Code	Note

`String a = new String("Hello");`
`String b = new String("Hello");`
`System.out.println(a == b);`
	
	`false`

`String a = new String("Hello");`
`String b = new String("Hello");`
 	
`String t1 = a.intern();`
`String t2 = b.intern();`
`System.out.println(a == b);`
`System.out.println(t1 == t2);`
	
	`false`
	`true`

You're unlikely to use this method often, but people love to ask about it in interviews. So it's better to know about it than to not know.