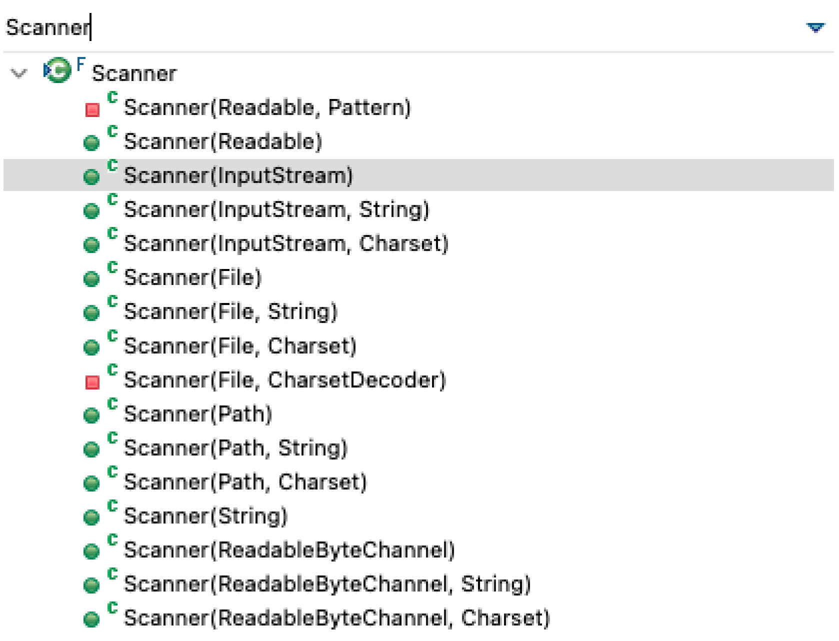

```
Scanner
Scanner
    Scanner(Readable, Pattern)
    Scanner(Readable)
    Scanner(InputStream)
    Scanner(InputStream, String)
    Scanner(InputStream, Charset)
    Scanner(File)
    Scanner(File, String)
    Scanner(File, Charset)
    Scanner(File, CharsetDecoder)
    Scanner(Path)
    Scanner(Path, String)
    Scanner(Path, Charset)
    Scanner(String)
    Scanner(ReadableByteChannel)
    Scanner(ReadableByteChannel, String)
    Scanner(ReadableByteChannel, Charset)
```

Scanner Class Constructors

Most of the constructors are using one of the three objects:

1. InputStream - the most common where we pass System.in to receive user input.
2. File or Path - We can scan file data too and work with the values from the file.
3. String - We can create a scanner for a string source too and parse values from it.

If you look at the second argument, it's to specify a character set if you don't want to use the default character set for parsing.

Important Methods of Scanner Class

Let's look at some of the most commonly used Scanner class methods.

- useDelimiter(String pattern) - the delimiter to be used for the scanner. The default delimiter is whitespace.
- hasNext() - returns true if there is another token in the input. It's a blocking method and it will keep waiting for user input.
- next() - returns the next token from the scanner. It's used in conjunction with the hasNext() method.
- close() - scanner is resource heavy, so once you are done with it, use this method to close it and release system resources.

There are many utility methods to check and directly parse the input token in int, short, long, byte, BigDecimal, etc.

Steps to Initialize and Use Scanner

1. The first step is to initialize the scanner class by using the appropriate constructor based on the input type such as InputStream, File, or String. If needed, set the delimiter and character set to use.
2. The second step is to wait for the input token using hasNext() method.
3. Then use the next() method to read the token and process them one by one.

How Does Scanner Work?

- The Scanner class breaks its input into tokens using the specified delimiter pattern.
- The next() methods is used to read the tokens one by one and process them.
- Finally, close the Scanner instance to release the system resources.

Scanner Examples

Let's look at some of the common usages of the Scanner class with sample code snippets.

1. Reading user input

This is the most common use of the Scanner class. We can instantiate with System.in as input source and read the user input.

Copy

```
// read user input
Scanner sc = new Scanner(System.in);
System.out.println("Please enter your name");
String name = sc.next();
System.out.println("Hello " + name);
sc.close();
```

Output:

Copy

```
Please enter your name
Pankaj
Hello Pankaj
```

Well, it looks easy and working fine. But, the above code has an issue. Without reading the next section, can you check the code and try to identify it?

Let’s see what happens when I write my full name in the input.

Copy

```
Please enter your name
Pankaj Kumar
Hello Pankaj
```

Now you must have got it, it’s happening because whitespace is the delimiter. The scanner is breaking the input into two tokens - Pankaj and Kumar. But, we are calling the next() method just once, so only “Hello Pankaj” is printed.

How do we fix this?

It’s simple. We can change the delimiter to a newline character using the useDelimiter() method.

Copy

```
Scanner sc = new Scanner(System.in);
sc.useDelimiter(System.getProperty("line.separator"));
System.out.println("Please enter your name");
String name = sc.next();
System.out.println("Hello " + name);
sc.close();
```

2. Parsing File Data using Scanner

Let’s look at a simple example to read and parse CSV files using the scanner class. Let’s say, I have an employees.csv file with the following content.

Copy

```
1,Jane Doe,CEO
2,Mary Ann,CTO
3,John Lee,CFO
```

Let’s read the file and get a list of Employees in our Java program.

Copy

```
package com.journaldev.java;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class ScannerExamples {

        public static void main(String[] args) throws IOException {

                // create scanner for the CSV file
                Scanner sc = new Scanner(new File("employees.csv"));

                // set delimiter as new line to read one line as a single token
                sc.useDelimiter(System.getProperty("line.separator"));

                // create the List of Employees
                List<Employee> emps = new ArrayList<>();

                while (sc.hasNext()) {
                        Employee emp = parseEmployeeData(sc.next());
                        emps.add(emp);
                }

                // close the scanner
                sc.close();

                // print employee records
                System.out.println(emps);
        }

        private static Employee parseEmployeeData(String record) {
                // create scanner for the String record
                Scanner sc = new Scanner(record);

                // set delimiter as comma
```

Log in
Sign up
Blog
Docs
Get Support
Contact Sales
Tutorials Questions Product Docs Cloud Chats Search Community

```
                this.name = name;
        }

        public String getRole() {
                return role;
        }

        public void setRole(String role) {
                this.role = role;
        }

        @Override
        public String toString() {
                return "Emp[" + id + "," + name + "," + role + "]";
        }
}
```

- The first step is to create the scanner for the CSV file and set newline as delimiter.
- Then for each line that contains employee record in CSV format, parse it using another scanner and comma as delimiter. The *parseEmployeeData*() is parsing each line and creating Employee object.
- Finally, we are adding the employee object to the list and printing it.

Output: [Emp[1,Jane Doe,CEO], Emp[2,Mary Ann,CTO], Emp[3,John Lee,CFO]]

3. Java Scanner Regular Expression Example

Let’s say we have a string source and we want to process only integers present in that. We can use the scanner with the non-digit regex to get only integers as tokens to process them.

Copy

```
//using regex to read only integers from a string source
String data = "1a2b345c67d8,9#10";
Scanner sc1 = new Scanner(data);

// setting non-digit regex as delimiter
sc1.useDelimiter("\\D");

while(sc1.hasNext()) {
        System.out.println(sc1.next());
}

// don't forget to close the scanner
sc1.close();
```

Output:

Copy

```
1
2
345
67
8
9
10
```

Conclusion

Java Scanner is a utility class to read user input or process simple regex-based parsing of file or string source. But, for real-world applications, it's better to use CSV parsers to parse CSV data rather than using the Scanner class for better performance.

Reference: API Doc, Regex in Java

Thanks for learning with the DigitalOcean Community. Check out our offerings for compute, storage, networking, and managed databases.

Learn more about our products →