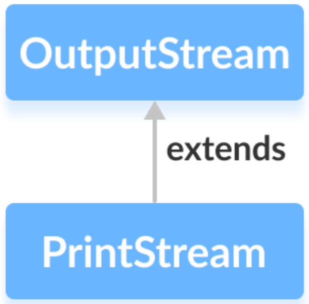
Java PrintStream Class

In this tutorial, we will learn about the Java PrintStream class and its print() and printf() methods with the help of examples.

The `PrintStream` class of the `java.io` package can be used to write output data in commonly readable form (text) instead of bytes.

It extends the abstract class `OutputStream`.



Working of PrintStream

Unlike other output streams, the `PrintStream` converts the primitive data (integer, character) into the text format instead of bytes. It then writes that formatted data to the output stream.

And also, the `PrintStream` class does not throw any input/output exception. Instead, we need to use the `checkError()` method to find any error in it.

> **Note**: The `PrintStream` class also has a feature of auto flushing. This means it forces the output stream to write all the data to the destination under one of the following conditions:

- if newline character `\n` is written in the print stream
- if the `println()` method is invoked
- if an array of bytes is written in the print stream

Create a PrintStream

In order to create a `PrintStream`, we must import the `java.io.PrintStream` package first. Once we import the package here is how we can create the print stream.

1. Using other output streams

```
// Creates a FileOutputStream
FileOutputStream file = new FileOutputStream(String file);

// Creates a PrintStream
PrintStream output = new PrintStream(file, autoFlush);
```

Here,

- we have created a print stream that will write formatted data to the file represented by `FileOutputStream`
- the `autoFlush` is an optional boolean parameter that specifies whether to perform auto flushing or not

2. Using filename

```
 // Creates a PrintStream
PrintStream output = new PrintStream(String file, boolean autoFlush);
```

Here,

- we have created a print stream that will write formatted data to the specified file

- `autoFlush` is an optional boolean parameter that specifies whether to perform autoflush or not

> **Note**: In both the case, the `PrintStream` write data to the file using some default character encoding. However, we can specify the character encoding (**UTF8** or **UTF16**) as well.

```
// Creates a PrintStream using some character encoding
PrintStream output = new PrintStream(String file, boolean autoFlush, Charset cs);
```

Here, we have used the `Charset` class to specify the character encoding. To learn more, visit Java Charset (official Java documentation).

Methods of PrintStream

The `PrintStream` class provides various methods that allow us to print data to the output.

print() Method

- `print()` - prints the specified data to the output stream
- `println()` - prints the data to the output stream along with a new line character at the end

---

Example: print() method with System class

```java
class Main {
    public static void main(String[] args) {

        String data = "Hello World.";
        System.out.print(data);
    }
}
```

Output

```
Hello World.
```

In the above example, we have not created a print stream. However, we can use the `print()` method of the `PrintStream` class.

You might be wondering how is this possible. Well, let me explain what is happening here.

Notice the line,

```
System.out.print(data);
```

Here,

- `System` is a final class that is responsible to perform standard input/output operation
- `out` is a class variable of `PrintStream` type declared in `System` class

Now since `out` is of `PrintStream` type, we can use it to call all the methods of `PrintStream` class.

Example: print() method with PrintStream class

```java
import java.io.PrintStream;

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

        String data = "This is a text inside the file.";

        try {
            PrintStream output = new PrintStream("output.txt");

            output.print(data);
            output.close();
        }
        catch(Exception e) {
            e.getStackTrace();
        }
    }
}
```

In the above example, we have created a print stream named `output`. The print stream is linked with the **output.txt** file.

```java
PrintStream output = new PrintStream("output.txt");
```

To print data to the file, we have used the `print()` method.

Here, when we run the program, the **output.txt** file is filled with the following content.

```
This is a text inside the file.
```

printf() Method

The `printf()` method can be used to print the formatted string. It includes 2 parameters: formatted string and arguments. For example,

```
printf("I am %d years old", 25);
```

Here,

- I am %d years old is a formatted string
- %d is integer data in the formatted string
- 25 is an argument

The formatted string includes both text and data. And, the arguments replace the data inside the formatted string.

Hence the **%d** is replaced by **25**.

Example: printf() method using PrintStream

```java
import java.io.PrintStream;

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

        try {
            PrintStream output = new PrintStream("output.txt");

            int age = 25;

            output.printf("I am %d years old.", age);
            output.close();
        }
        catch(Exception e) {
            e.getStackTrace();
        }
    }
}
```

In the above example, we have created a print stream named `output`. The print stream is linked with the file **output.txt**.

```java
PrintStream output = new PrintStream("output.txt");
```

To print the formatted text to the file, we have used the `printf()` method.

Here, when we run the program, the **output.txt** file is filled with the following content.

```
I am 25 years old.
```

Other Methods Of PrintStream

Methods	Descriptions
`close()`	closes the print stream
`checkError()`	checks if there is an error in the stream and returns a boolean result
`append()`	appends the specified data to the stream