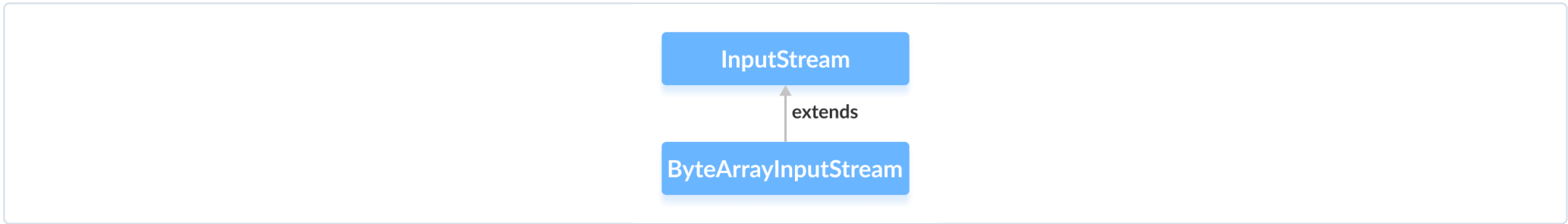
Java ByteArrayInputStream Class

In this tutorial, we will learn about Java ByteArrayInputStream and its methods with the help of examples.

The `ByteArrayInputStream` class of the `java.io` package can be used to read an array of input data (in bytes).

It extends the `InputStream` abstract class.



> **Note**: In `ByteArrayInputStream`, the input stream is created using the array of bytes. It includes an internal array to store data of that particular byte array.

Create a ByteArrayInputStream

In order to create a byte array input stream, we must import the `java.io.ByteArrayInputStream` package first. Once we import the package, here is how we can create an input stream.

```
// Creates a ByteArrayInputStream that reads entire array
ByteArrayInputStream input = new ByteArrayInputStream(byte[] arr);
```

Here, we have created an input stream that reads entire data from the `arr` array. However, we can also create the input stream that reads only some data from the array.

```
// Creates a ByteArrayInputStream that reads a portion of array
ByteArrayInputStream input = new ByteArrayInputStream(byte[] arr, int start, int length);
```

Here the input stream reads the number of bytes equal to `length` from the array starting from the `start` position.

Methods of ByteArrayInputStream

The `ByteArrayInputStream` class provides implementations for different methods present in the `InputStream` class.

read() Method

- `read()` - reads the single byte from the array present in the input stream
- `read(byte[] array)` - reads bytes from the input stream and stores in the specified array
- `read(byte[] array, int start, int length)` - reads the number of bytes equal to `length` from the stream and stores in the specified array starting from the position `start`

Example: ByteArrayInputStream to read data

```
import java.io.ByteArrayInputStream;

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

    // Creates an array of byte
    byte[] array = {1, 2, 3, 4};

    try {
      ByteArrayInputStream input = new ByteArrayInputStream(array);

      System.out.print("The bytes read from the input stream: ");

      for(int i= 0; i < array.length; i++) {

        // Reads the bytes
        int data = input.read();
        System.out.print(data + ", ");
      }
      input.close();
    }

    catch(Exception e) {
      e.getStackTrace();
    }
  }
}
```

Output

```
The bytes read from the input stream: 1, 2, 3, 4,
```

In the above example, we have created a byte array input stream named `input`.

```
ByteArrayInputStream input = new ByteArrayInputStream(array);
```

Here, the input stream includes all the data from the specified array. To read data from the input stream, we have used the `read()` method.

---

available() Method

To get the number of available bytes in the input stream, we can use the `available()` method. For example,

```java
import java.io.ByteArrayInputStream;

public class Main {

  public static void main(String args[]) {

    // Creates an array of bytes
    byte[] array = { 1, 2, 3, 4 };

    try {
```

```
      ByteArrayInputStream input = new ByteArrayInputStream(array);

      // Returns the available number of bytes
      System.out.println("Available bytes at the beginning: " + input.available());

      // Reads 2 bytes from the input stream
      input.read();
      input.read();

      // Returns the available number of bytes
      System.out.println("Available bytes at the end: " + input.available());

      input.close();
    }

    catch (Exception e) {
      e.getStackTrace();
    }
  }
```

Output

```
Available bytes at the beginning: 4
Available bytes at the end: 2
```

In the above example,

1. We have used the `available()` method to check the number of available bytes in the input stream.
2. We have then used the `read()` method 2 times to read 2 bytes from the input stream.
3. Now, after reading the 2 bytes, we have checked the available bytes. This time the available bytes decreased by 2.

skip() Method

To discard and skip the specified number of bytes, we can use the `skip()` method. For example,

```java
import java.io.ByteArrayInputStream;

public class Main {

  public static void main(String args[]) {

    // Create an array of bytes
    byte[] array = { 1, 2, 3, 4 };
    try {
      ByteArrayInputStream input = new ByteArrayInputStream(array);

      // Using the skip() method
      input.skip(2);
      System.out.print("Input stream after skipping 2 bytes: ");

      int data = input.read();
      while (data != -1) {
        System.out.print(data + ", ");
        data = input.read();
      }

      // close() method
      input.close();
    }

    catch (Exception e) {
      e.getStackTrace();
    }
  }
```

Output

```
Input stream after skipping 2 bytes: 3, 4,
```

In the above example, we have used the `skip()` method to skip 2 bytes of data from the input stream. Hence `1` and `2` are not read from the input stream.

---

close() Method

To close the input stream, we can use the `close()` method.

However, the `close()` method has no effect in `ByteArrayInputStream` class. We can use the methods of this class even after the `close()` method is called.

Other Methods Of ByteArrayInputStream

Methods	Descriptions
`finalize()`	ensures that the `close()` method is called
`mark()`	marks the position in input stream up to which data has been read
`reset()`	returns the control to the point in the input stream where the mark was set
`markSupported()`	checks if the input stream supports `mark()` and `reset()`