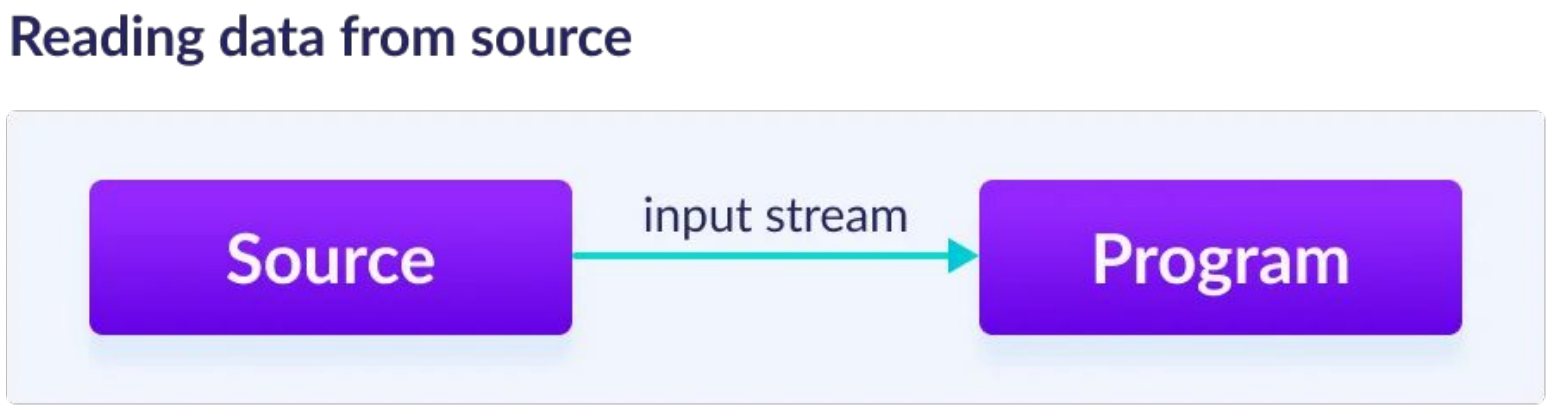
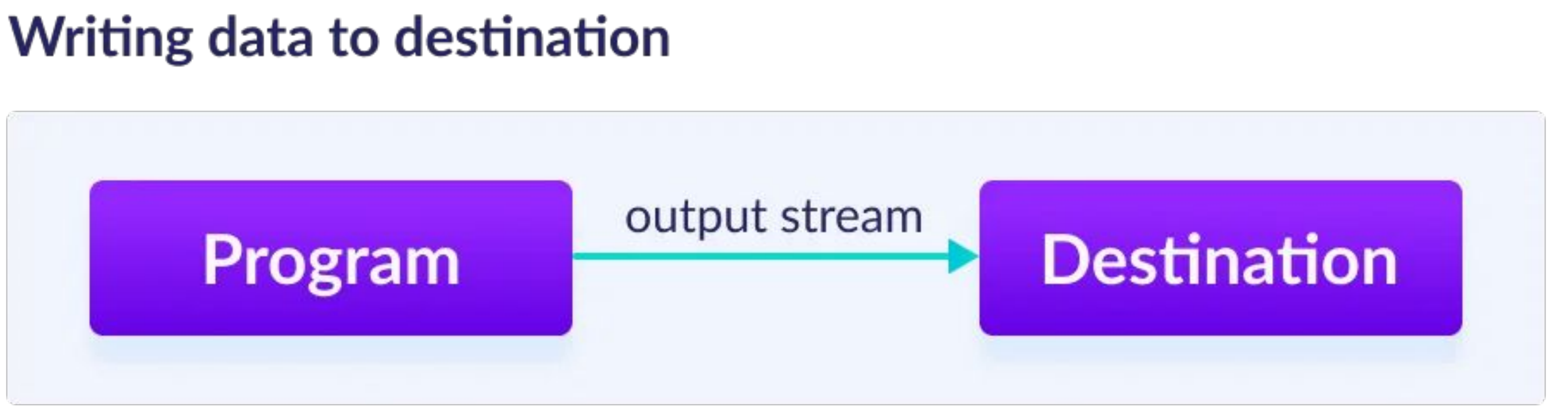
Unit 8
I/O and Streams

Java I/O Stream

- ❖ In Java, streams are the sequence of data that are read from the source and written to the destination.
- ❖ An input stream is used to read data from the source. And, an output stream is used to write data to the destination.
- ❖ Java uses the concept of a stream to make I/O operation fast.
- ❖ The java.io package contains all the classes required for input and output operations.
- ❖ In Java, 3 streams are created for us automatically. All these streams are attached with the console.
  1. **System.out:** standard output stream
  2. **System.in:** standard input stream
  3. **System.err:** standard error stream

Java I/O Stream





java.io package

- This package provides for system input and output through data streams, serialization and the file system.
- We can perform file handling in Java by Java I/O API.

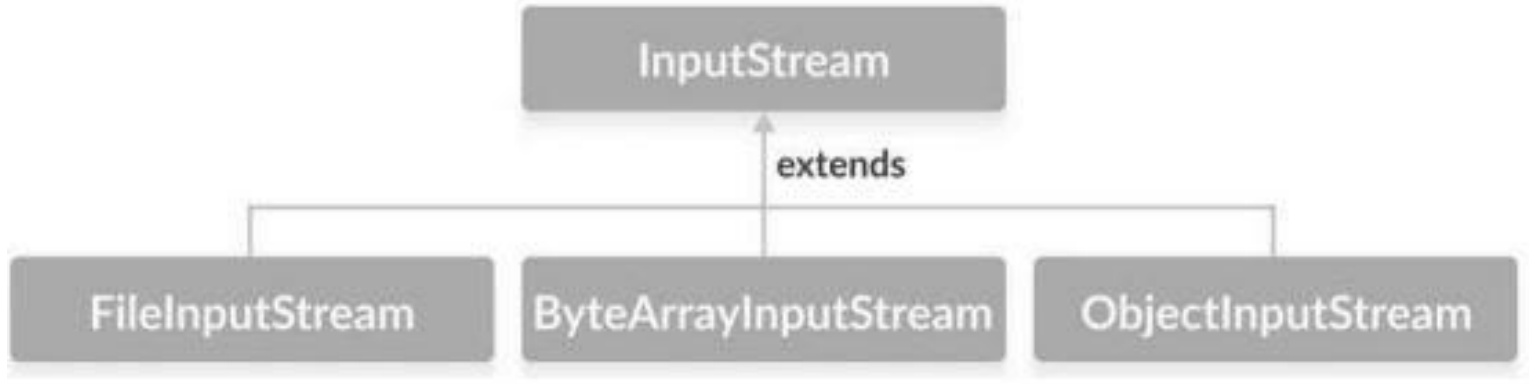
Java I/O Streams

❖ Depending upon the data a stream holds, it can be classified into:

1. Byte Stream
   - Byte stream is used to read and write a single byte (8 bits) of data.
   - All byte stream classes are derived from base abstract classes called InputStream and OutputStream
2. Character Stream
   - Character stream is used to read and write a single character of data.
   - All the character stream classes are derived from base abstract classes Reader and Writer

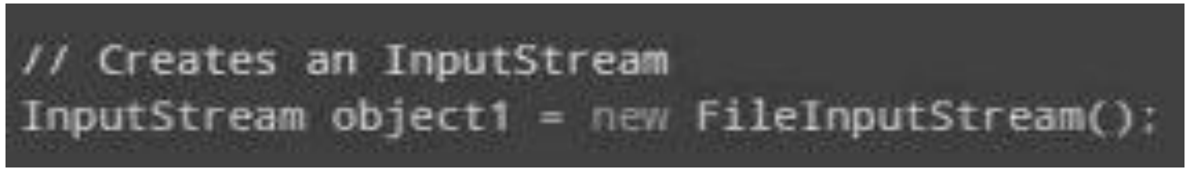
Java InputStream

- ❖ The InputStream class of the java.io package is an abstract superclass that represents an input stream of bytes.
- ❖ Since InputStream is an abstract class, it is not useful by itself. However, its subclasses can be used to read data.



Creating an InputStream

- ❖ In order to create an InputStream, we must import the java.io.InputStream package first.
- ❖ Once we import the package, here is how we can create the input stream



```
// Creates an InputStream
InputStream object1 = new FileInputStream();
```

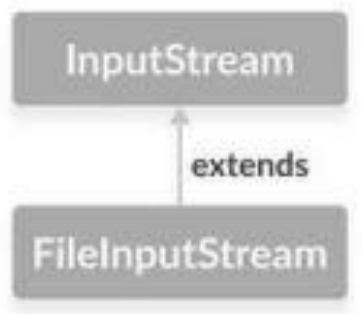
- ❖ Here, we have created an input stream using FileInputStream.
- ❖ It is because InputStream is an abstract class. Hence we cannot create an object of InputStream

Methods of InputStream

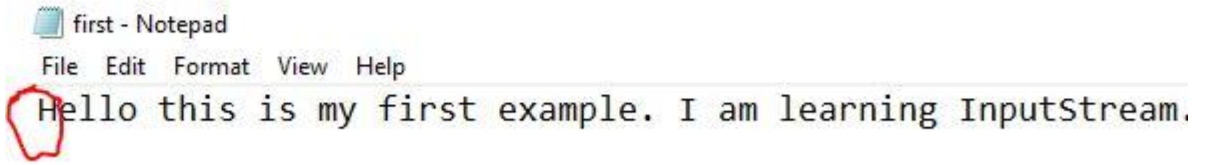
- ❖ The InputStream class provides different methods that are implemented by its subclasses. Here are some of the commonly used methods:
    - read() - reads one byte of data from the input stream
    - read(byte[] array) - reads bytes from the stream and stores in the specified array
    - available() - returns the number of bytes available in the input stream
    - mark() - marks the position in the input stream up to which data has been read
    - reset() - returns the control to the point in the stream where the mark was set
    - markSupported() - checks if the mark() and reset() method is supported in the stream
    - skip() - skips and discards the specified number of bytes from the input stream
    - close() - closes the input stream

Java FileInputStream Class

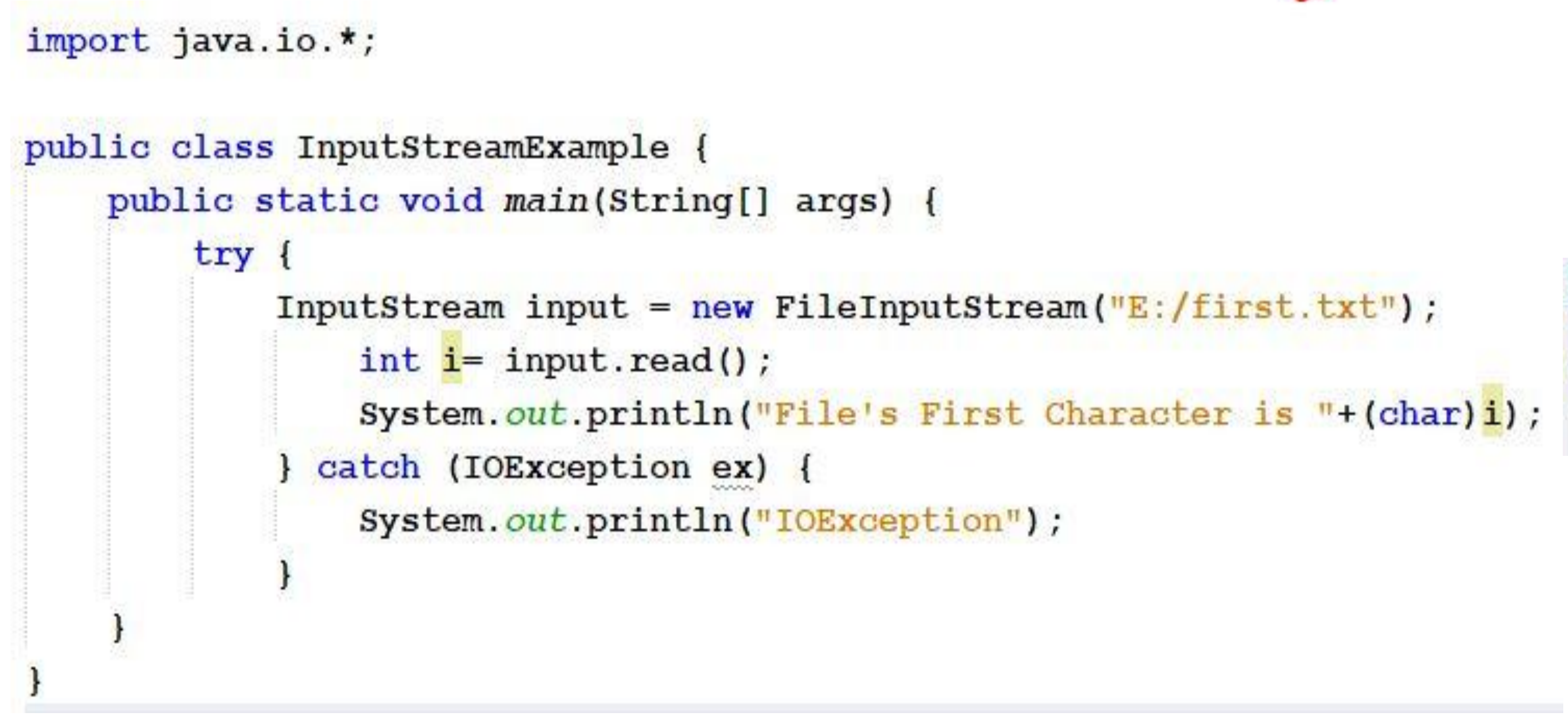
- The FileInputStream class of the java.io package can be used to read data (in bytes) from files.
- It extends the InputStream abstract class



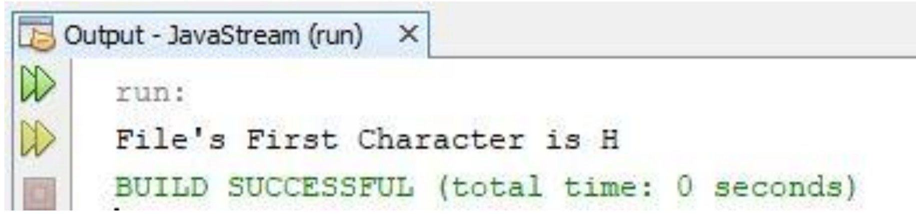
Reading a file using FileInputStream



```
Hello this is my first example. I am learning InputStream.
```

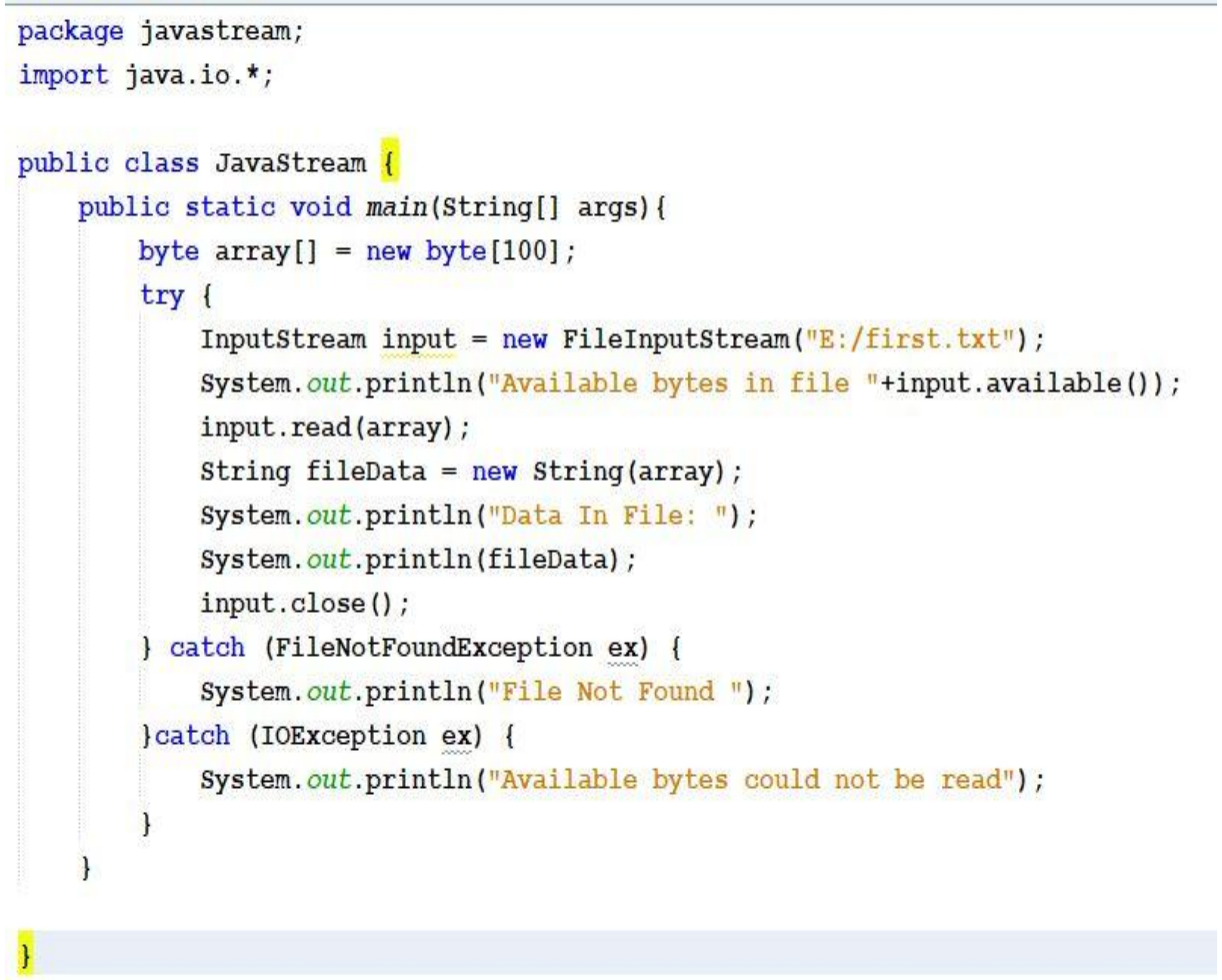


```
import java.io.*;

public class InputStreamExample {
    public static void main(String[] args) {
        try {
            InputStream input = new FileInputStream("E:/first.txt");
                int i= input.read();
                System.out.println("File's First Character is "+(char)i);
            } catch (IOException ex) {
                System.out.println("IOException");
            }
    }
}
```

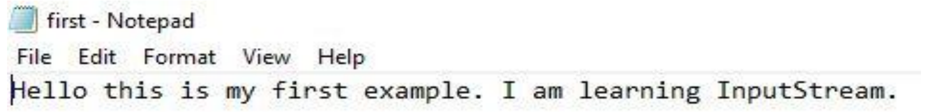


```
run:
File's First Character is H
BUILD SUCCESSFUL (total time: 0 seconds)
```

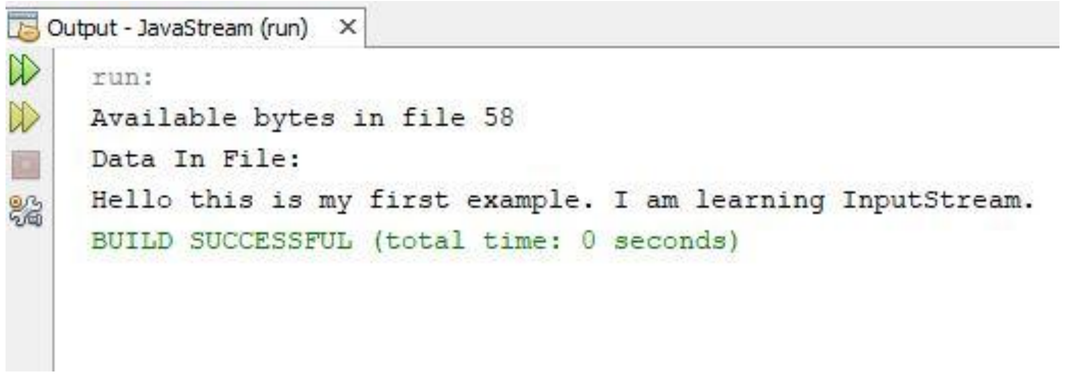
Reading a file using FileInputStream



```
package javastream;
import java.io.*;

public class JavaStream {
    public static void main(String[] args){
        byte array[] = new byte[100];
        try {
            InputStream input = new FileInputStream("E:/first.txt");
            System.out.println("Available bytes in file "+input.available());
            input.read(array);
            String fileData = new String(array);
            System.out.println("Data In File: ");
            System.out.println(fileData);
            input.close();
        } catch (FileNotFoundException ex) {
            System.out.println("File Not Found ");
        }catch (IOException ex) {
            System.out.println("Available bytes could not be read");
        }
    }

}
```

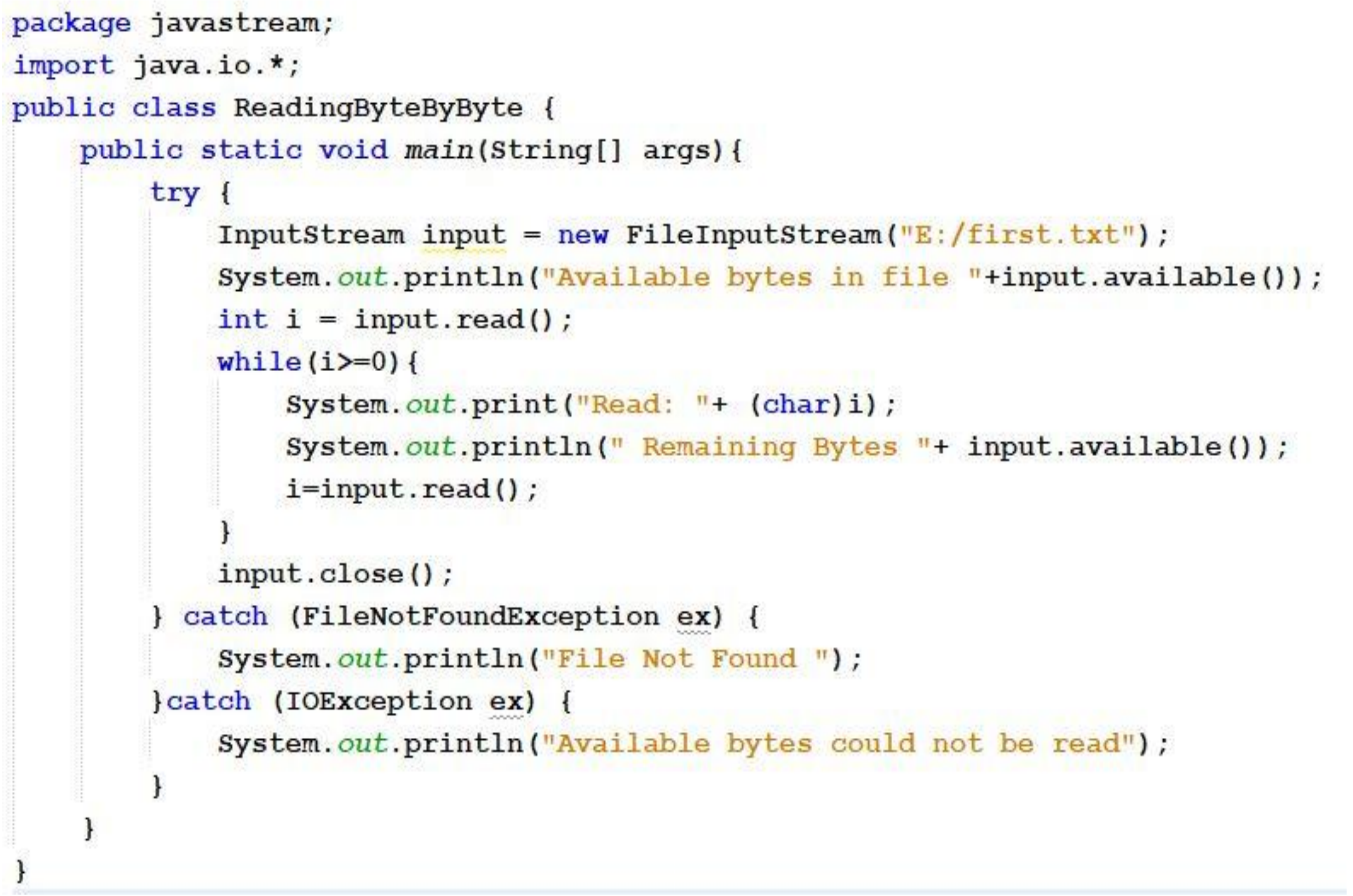


```
Hello this is my first example. I am learning InputStream.
```

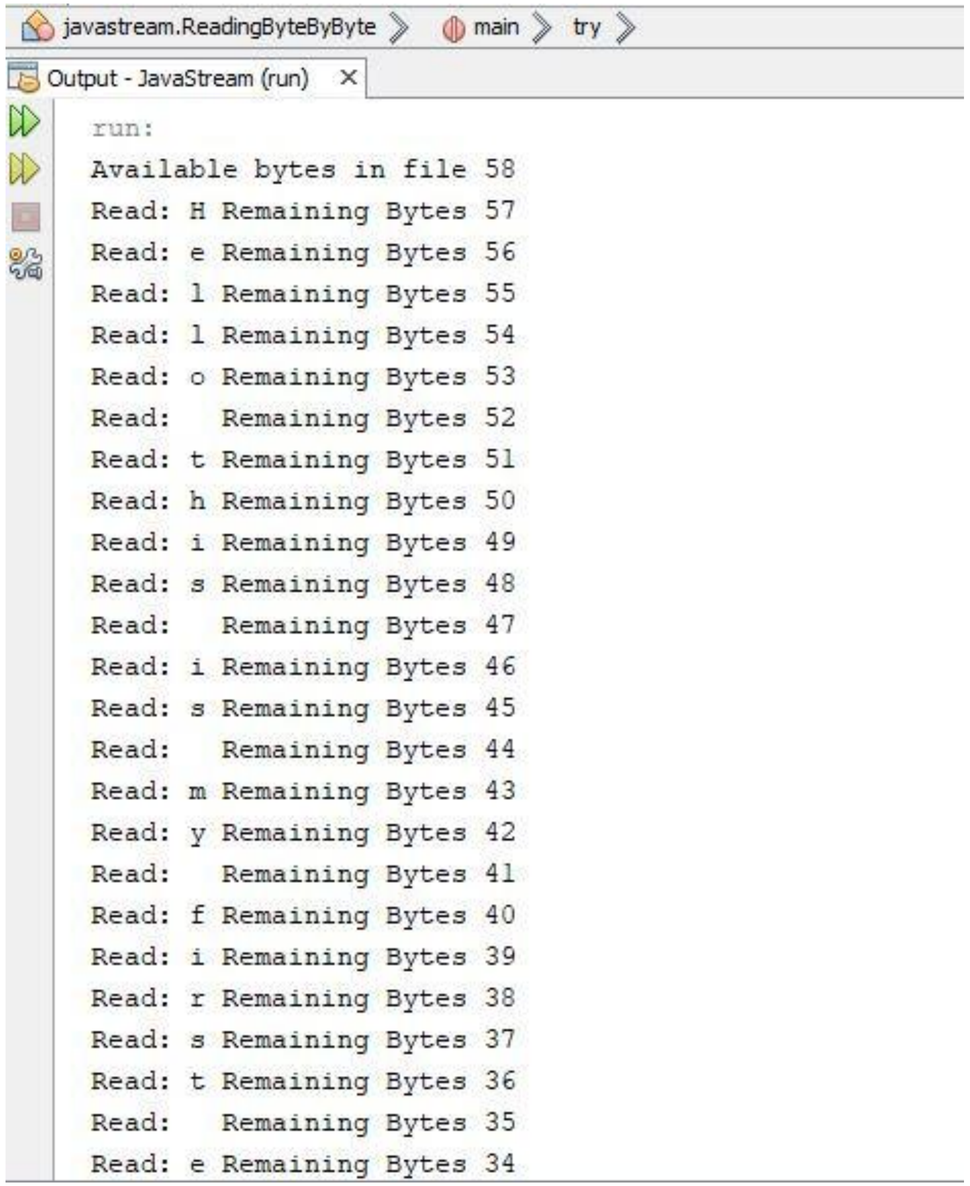


```
run:
Available bytes in file 58
Data In File:
Hello this is my first example. I am learning InputStream.
BUILD SUCCESSFUL (total time: 0 seconds)
```

InputStream-Reading a File Byte by Byte

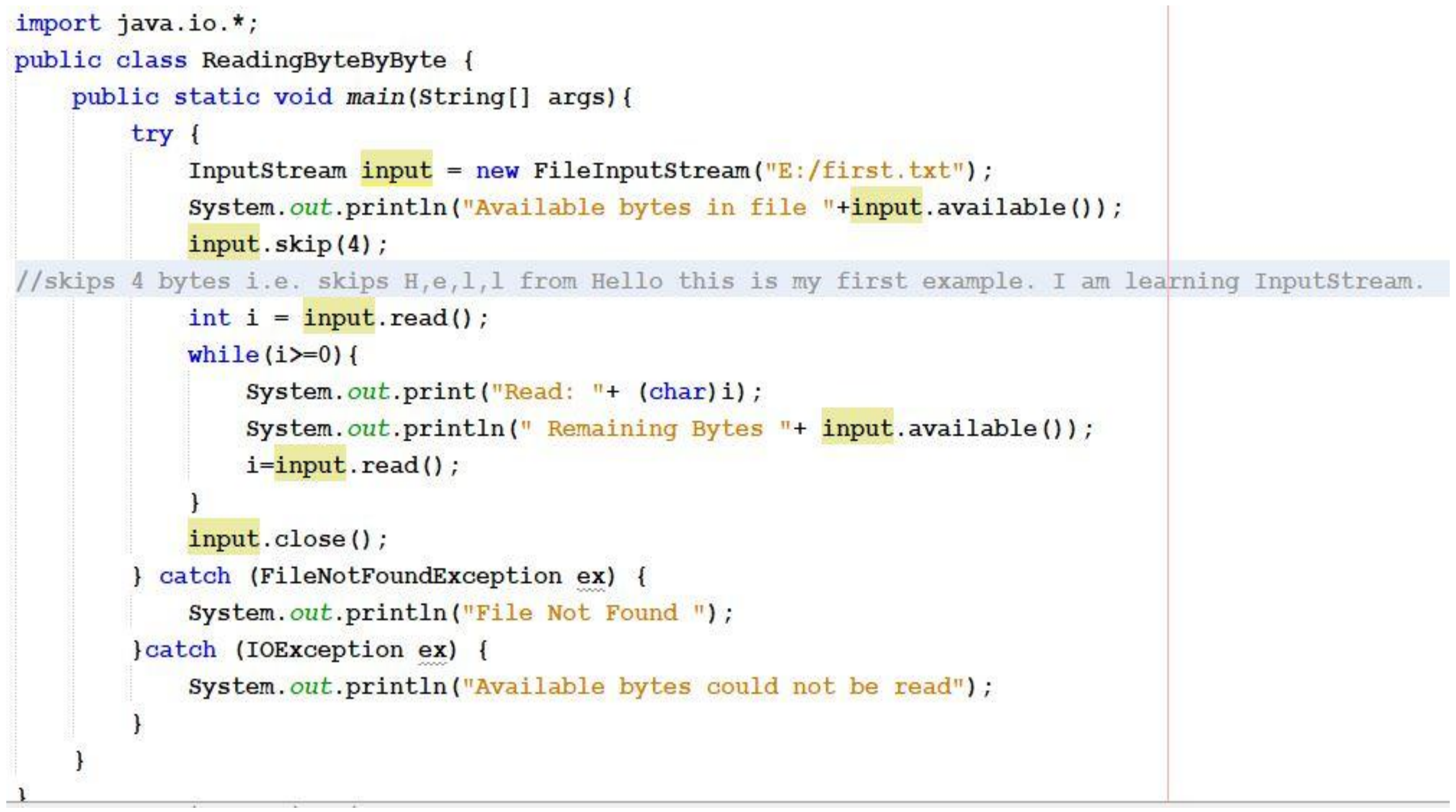


```java
package javastream;
import java.io.*;
public class ReadingByteByByte {
    public static void main(String[] args){
        try {
            InputStream input = new FileInputStream("E:/first.txt");
            System.out.println("Available bytes in file "+input.available());
            int i = input.read();
            while(i>=0){
                System.out.print("Read: "+ (char)i);
                System.out.println(" Remaining Bytes "+ input.available());
                i=input.read();
            }
            input.close();
        } catch (FileNotFoundException ex) {
            System.out.println("File Not Found ");
        }catch (IOException ex) {
            System.out.println("Available bytes could not be read");
        }
    }
}
```

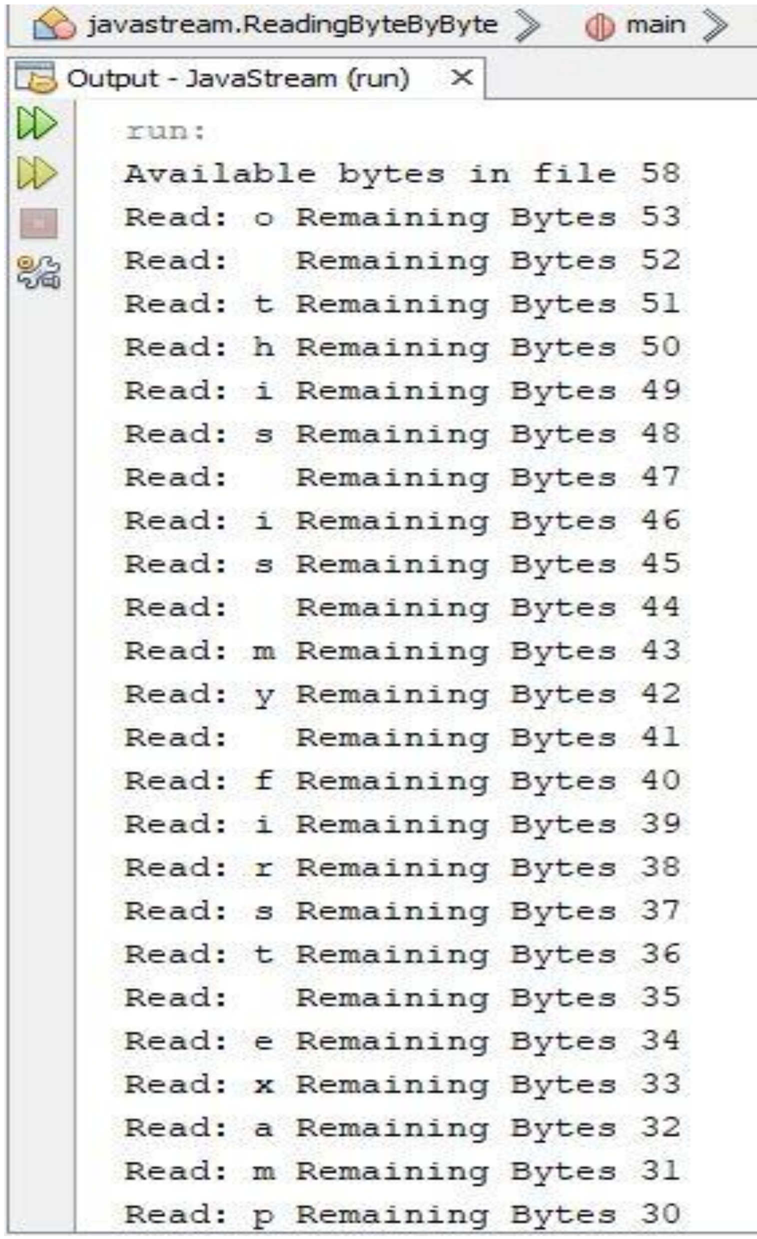


```
run:
Available bytes in file 58
Read: H Remaining Bytes 57
Read: e Remaining Bytes 56
Read: l Remaining Bytes 55
Read: l Remaining Bytes 54
Read: o Remaining Bytes 53
Read:   Remaining Bytes 52
Read: t Remaining Bytes 51
Read: h Remaining Bytes 50
Read: i Remaining Bytes 49
Read: s Remaining Bytes 48
Read:   Remaining Bytes 47
Read: i Remaining Bytes 46
Read: s Remaining Bytes 45
Read:   Remaining Bytes 44
Read: m Remaining Bytes 43
Read: y Remaining Bytes 42
Read:   Remaining Bytes 41
Read: f Remaining Bytes 40
Read: i Remaining Bytes 39
Read: r Remaining Bytes 38
Read: s Remaining Bytes 37
Read: t Remaining Bytes 36
Read:   Remaining Bytes 35
Read: e Remaining Bytes 34
```

InputStream-Reading a File Byte by Byte-skip() method



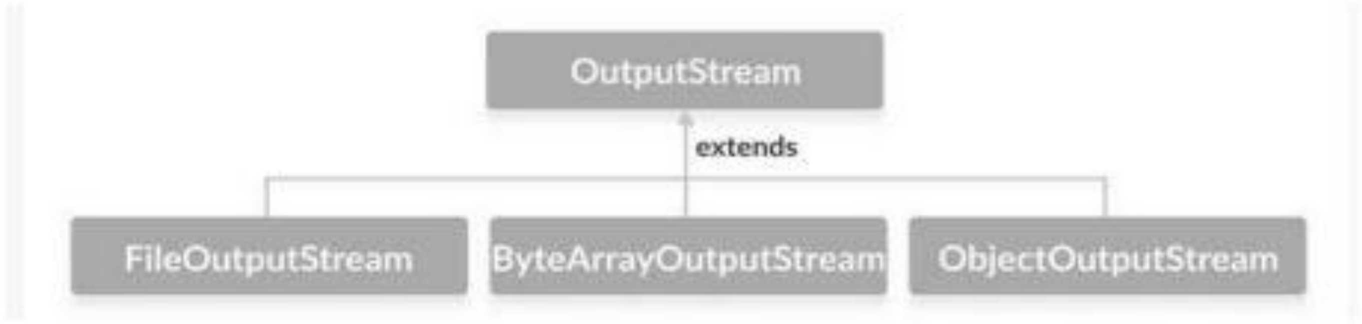
```
import java.io.*;
public class ReadingByteByByte {
    public static void main(String[] args){
        try {
            InputStream input = new FileInputStream("E:/first.txt");
            System.out.println("Available bytes in file "+input.available());
            input.skip(4);
//skips 4 bytes i.e. skips H,e,l,l from Hello this is my first example. I am learning InputStream.
            int i = input.read();
            while(i>=0){
                System.out.print("Read: "+ (char)i);
                System.out.println(" Remaining Bytes "+ input.available());
                i=input.read();
            }
            input.close();
        } catch (FileNotFoundException ex) {
            System.out.println("File Not Found ");
        }catch (IOException ex) {
            System.out.println("Available bytes could not be read");
        }
    }
}
```



```
run:
Available bytes in file 58
Read: o Remaining Bytes 53
Read:   Remaining Bytes 52
Read: t Remaining Bytes 51
Read: h Remaining Bytes 50
Read: i Remaining Bytes 49
Read: s Remaining Bytes 48
Read:   Remaining Bytes 47
Read: i Remaining Bytes 46
Read: s Remaining Bytes 45
Read:   Remaining Bytes 44
Read: m Remaining Bytes 43
Read: y Remaining Bytes 42
Read:   Remaining Bytes 41
Read: f Remaining Bytes 40
Read: i Remaining Bytes 39
Read: r Remaining Bytes 38
Read: s Remaining Bytes 37
Read: t Remaining Bytes 36
Read:   Remaining Bytes 35
Read: e Remaining Bytes 34
Read: x Remaining Bytes 33
Read: a Remaining Bytes 32
Read: m Remaining Bytes 31
Read: p Remaining Bytes 30
```

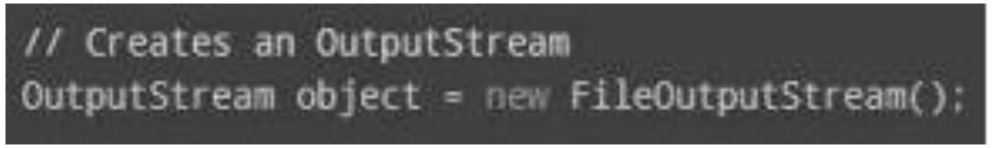
Java OutputStream

- ❖ The OutputStream class of the java.io package is an abstract superclass that represents an output stream of bytes.
- ❖ Since OutputStream is an abstract class, it is not useful by itself. However, its subclasses can be used to write data.



Creating an OutputStream

- ❖ In order to create an OutputStream, we must import the java.io.OutputStream package first.
- ❖ Once we import the package, here is how we can create the output stream



```
// Creates an OutputStream
OutputStream object = new FileOutputStream();
```

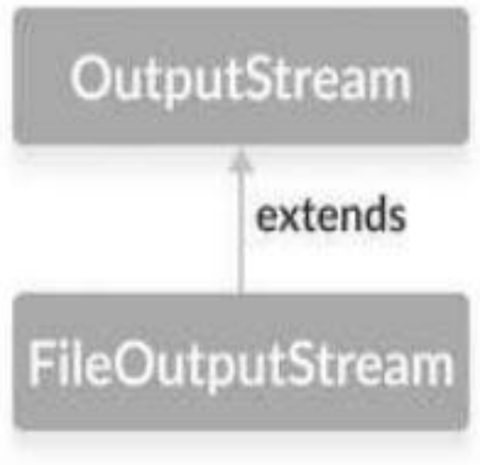
- ❖ Here, we have created an object of output stream using FileOutputStream.
- ❖ It is because OutputStream is an abstract class, so we cannot create an object of OutputStream

Methods of OuputStream

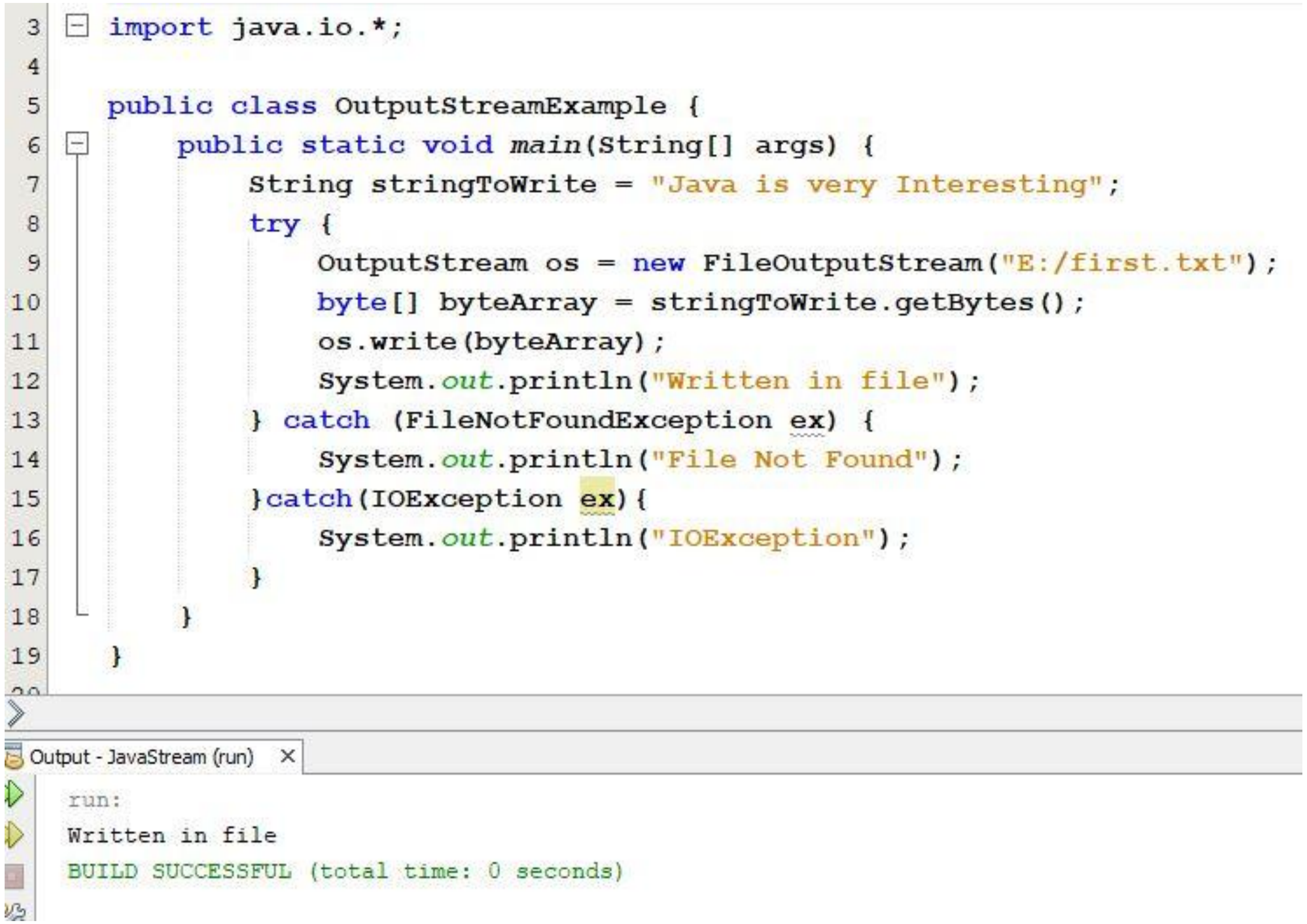
- ❖ The OutputStream class provides different methods that are implemented by its subclasses.
- ❖ Here are some of the methods:
  - write() - writes the specified byte to the output stream
  - write(byte[] array) - writes the bytes from the specified array to the output stream
  - flush() - forces to write all data present in output stream to the destination
  - close() - closes the output stream

Java FileOutputStream Class

- The FileOutputStream class of the java.io package can be used to write data (in bytes) to the files.
- It extends the OutputStream abstract class.



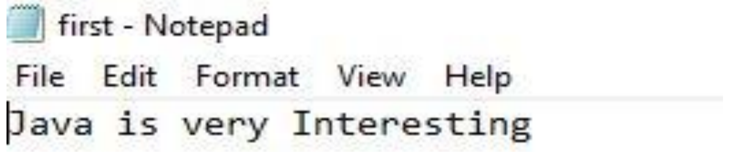
Writing to a file using FileOutputStream



```
import java.io.*;

public class OutputStreamExample {
    public static void main(String[] args) {
        String stringToWrite = "Java is very Interesting";
        try {
            OutputStream os = new FileOutputStream("E:/first.txt");
            byte[] byteArray = stringToWrite.getBytes();
            os.write(byteArray);
            System.out.println("Written in file");
        } catch (FileNotFoundException ex) {
            System.out.println("File Not Found");
        }catch(IOException ex){
            System.out.println("IOException");
        }
    }
}
```

Output - JavaStream (run)

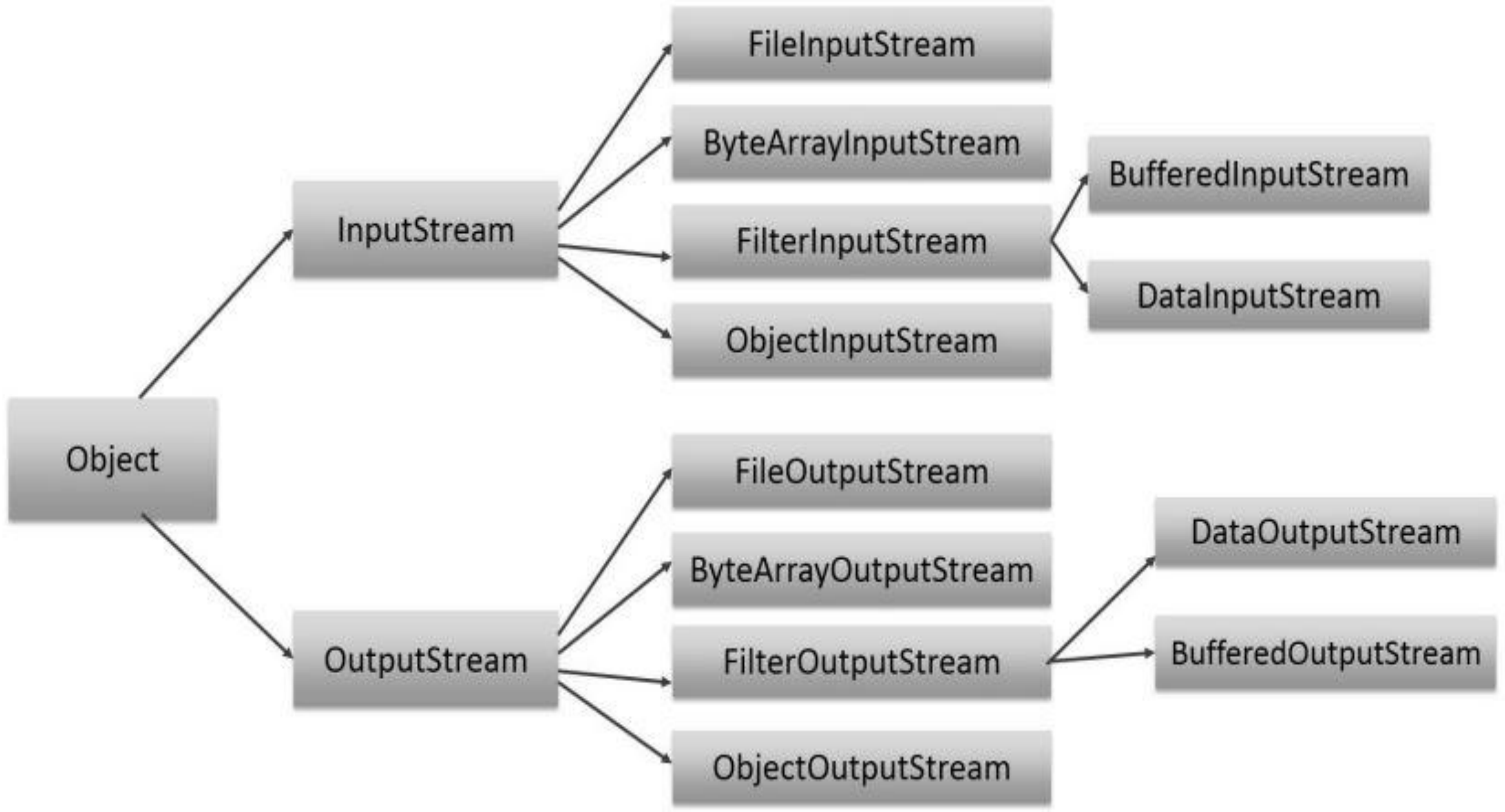
```
run:
Written in file
BUILD SUCCESSFUL (total time: 0 seconds)
```



```
Java is very Interesting
```

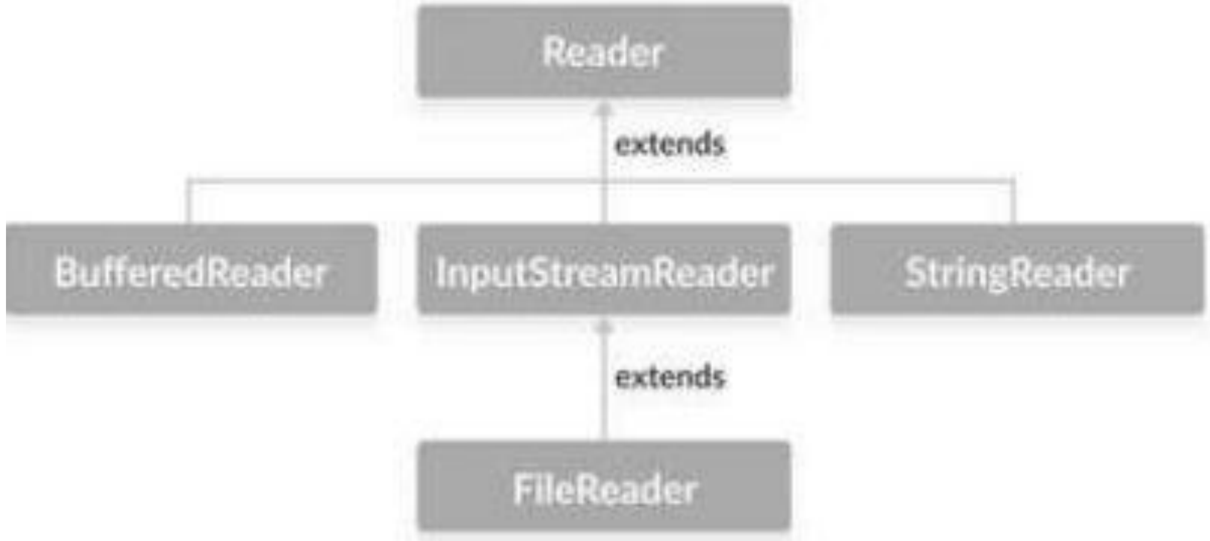
first.txt file

I/O Stream in Java



Java Reader Class

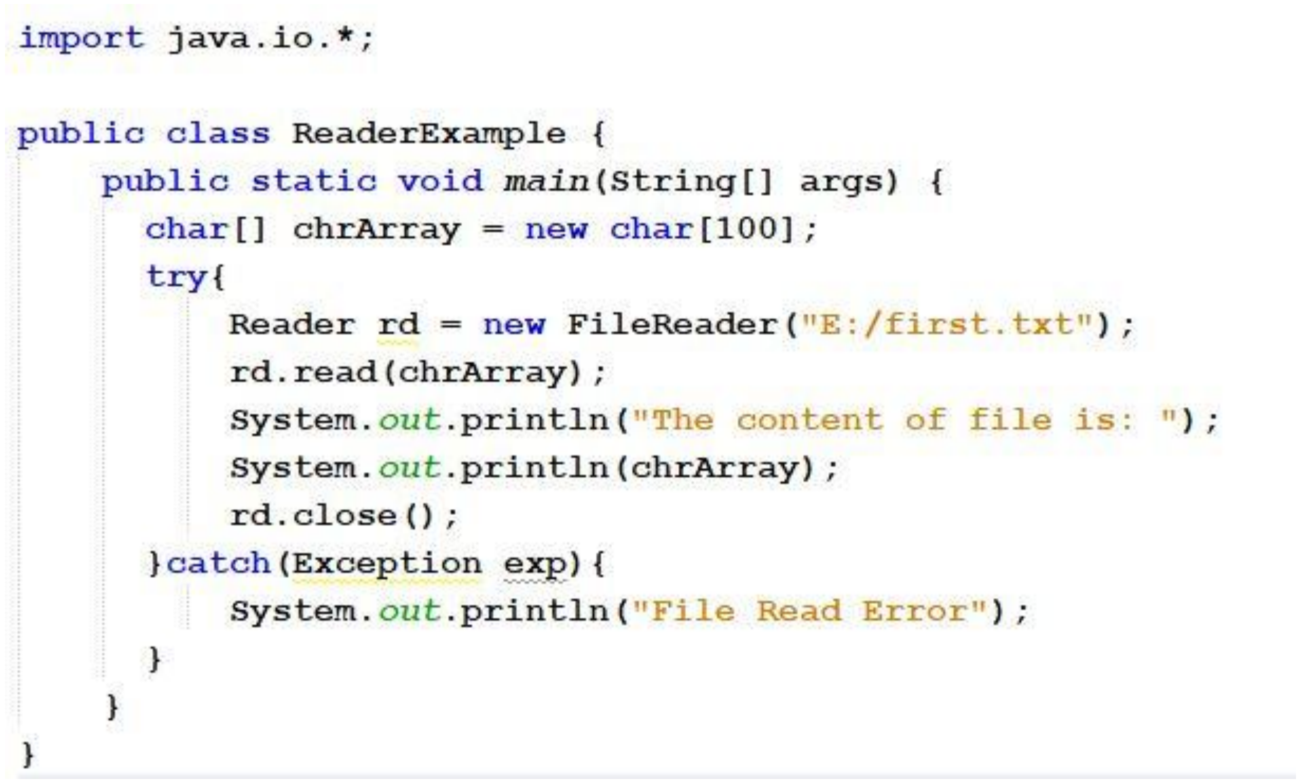
- ❖ The Reader class of the java.io package is an abstract superclass that represents a stream of characters.
- ❖ Since Reader is an abstract class, it is not useful by itself. However, its subclasses can be used to read data.



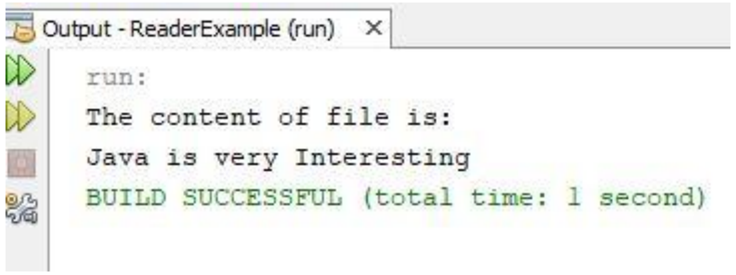
Methods of Reader Class

- ❖ The Reader class provides different methods that are implemented by its subclasses. Here are some of the commonly used methods:
  - ready() - checks if the reader is ready to be read
  - read(char[] array) - reads the characters from the stream and stores in the specified array
  - read(char[] array, int start, int length) - reads the number of characters equal to length from the stream and stores in the specified array starting from the start
  - mark() - marks the position in the stream up to which data has been read
  - reset() - returns the control to the point in the stream where the mark is set
  - skip() - discards the specified number of characters from the stream

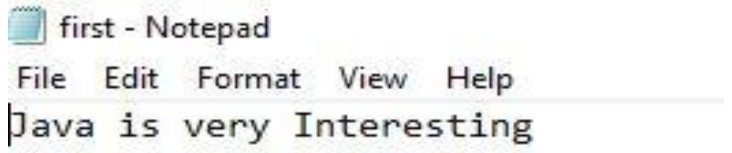
Reader Class Example



```
import java.io.*;

public class ReaderExample {
    public static void main(String[] args) {
      char[] chrArray = new char[100];
      try{
          Reader rd = new FileReader("E:/first.txt");
          rd.read(chrArray);
          System.out.println("The content of file is: ");
          System.out.println(chrArray);
          rd.close();
      }catch(Exception exp){
          System.out.println("File Read Error");
      }
    }
}
```



```
run:
The content of file is:
Java is very Interesting
BUILD SUCCESSFUL (total time: 1 second)
```

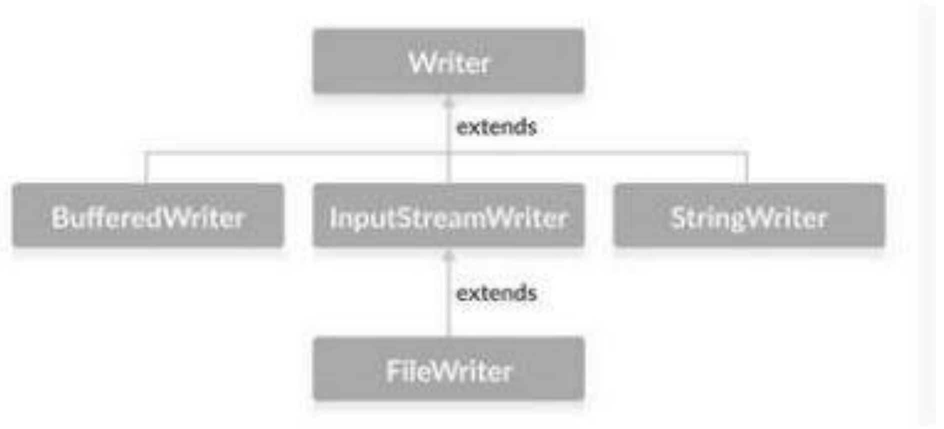


```
Java is very Interesting
```

first.txt file

Java Writer Class

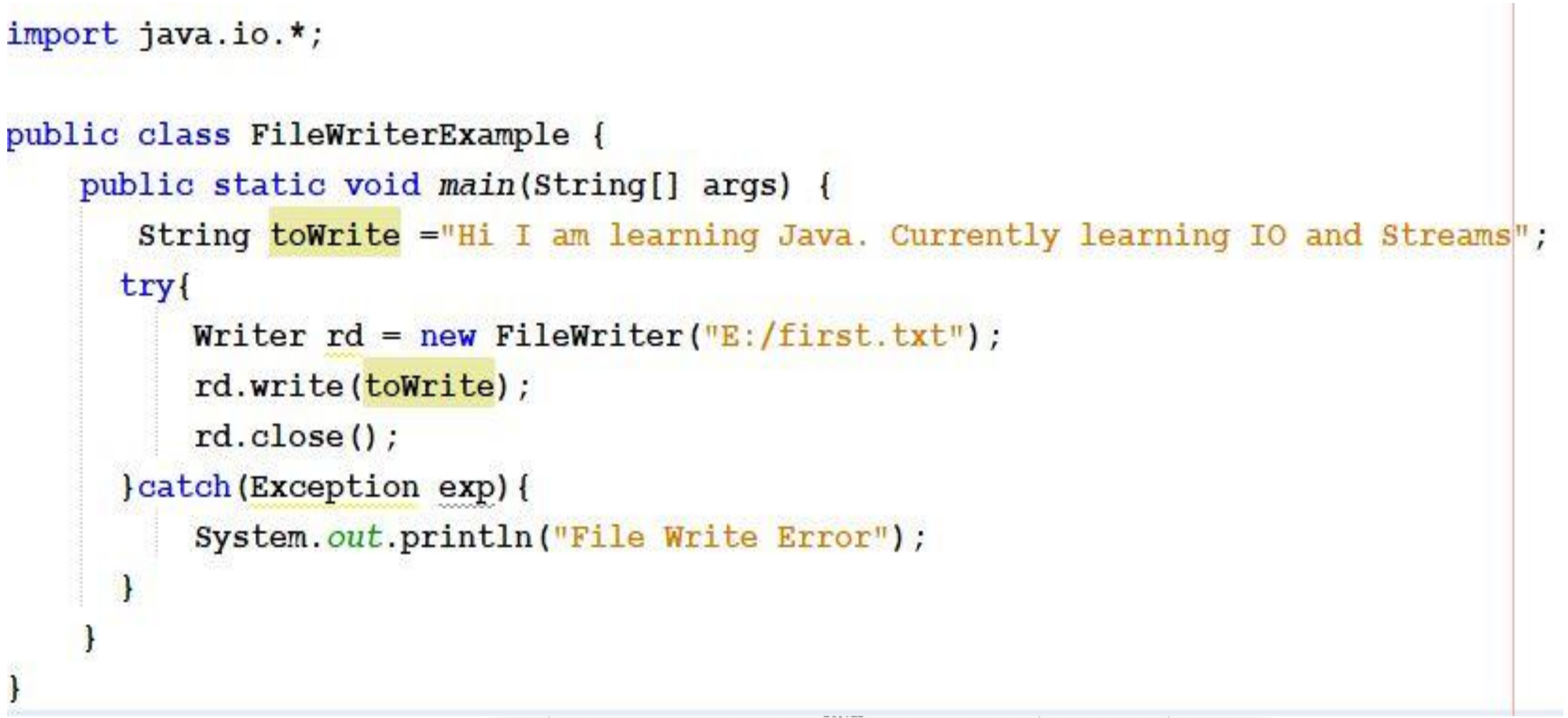
- ❖ The Writer class of the java.io package is an abstract superclass that represents a stream of characters.
- ❖ Since Writer is an abstract class, it is not useful by itself. However, its subclasses can be used to write data.



Methods of Writer Class

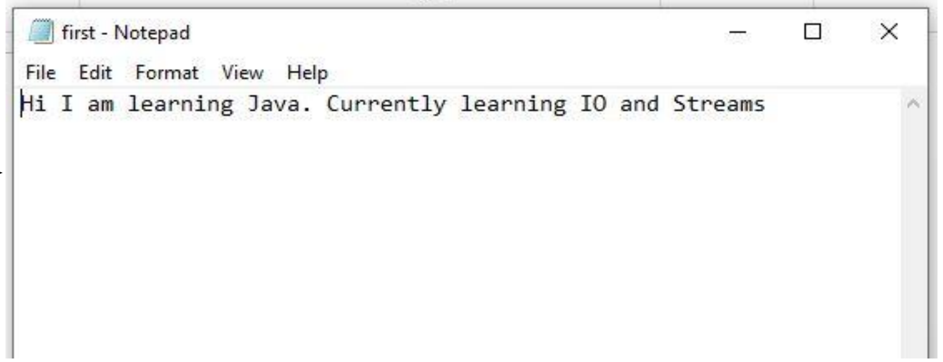
- ❖ The Writer class provides different methods that are implemented by its subclasses. Here are some of the methods:
    - write(char[] array) - writes the characters from the specified array to the output stream
    - write(String data) - writes the specified string to the writer
    - append(char c) - inserts the specified character to the current writer
    - flush() - forces to write all the data present in the writer to the corresponding destination
    - close() - closes the writer

Writer Class Example



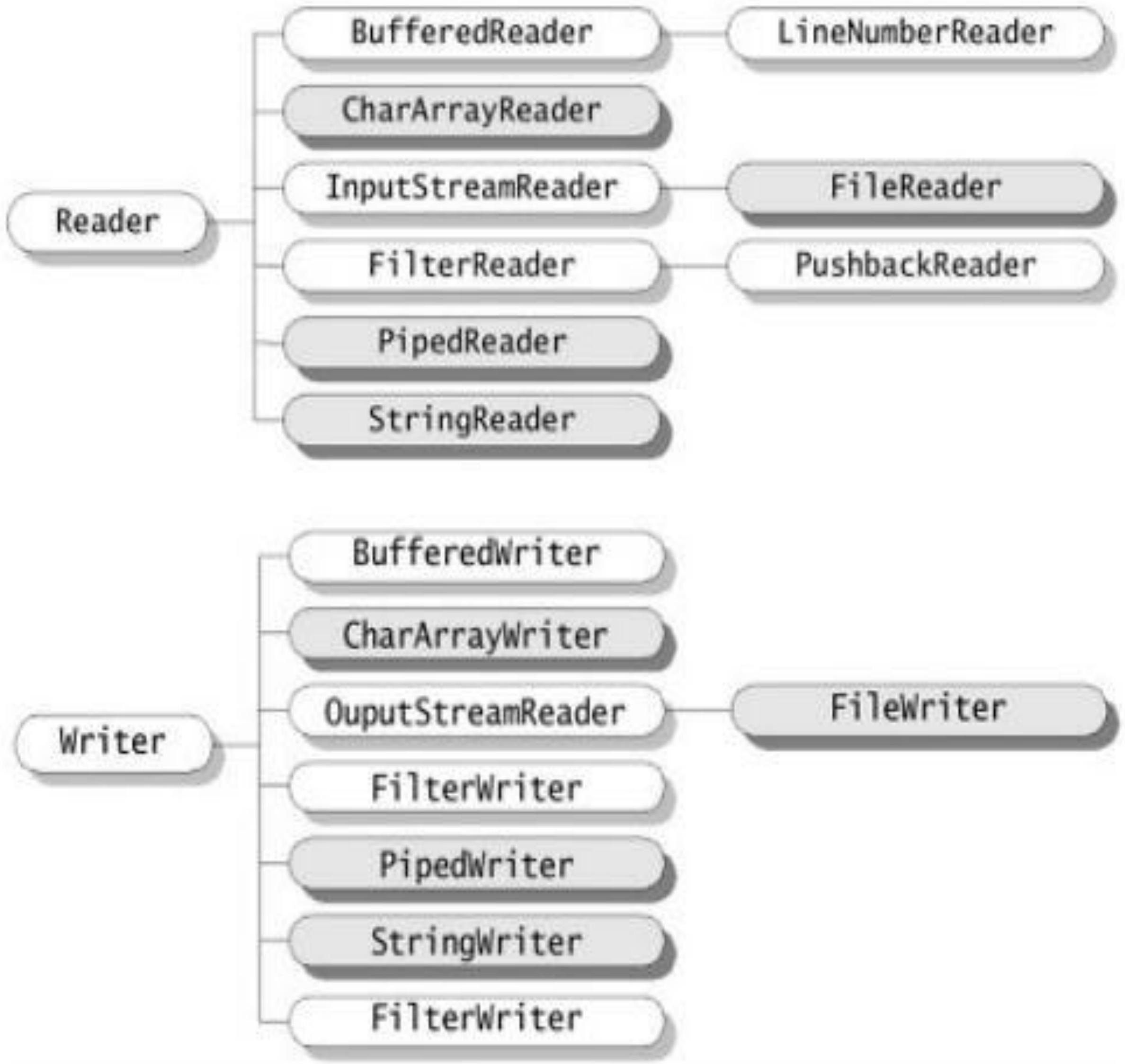
```
import java.io.*;

public class FileWriterExample {
    public static void main(String[] args) {
        String toWrite ="Hi I am learning Java. Currently learning IO and Streams";
       try{
            Writer rd = new FileWriter("E:/first.txt");
            rd.write(toWrite);
            rd.close();
       }catch(Exception exp){
            System.out.println("File Write Error");
       }
    }
}
```

first.txt file



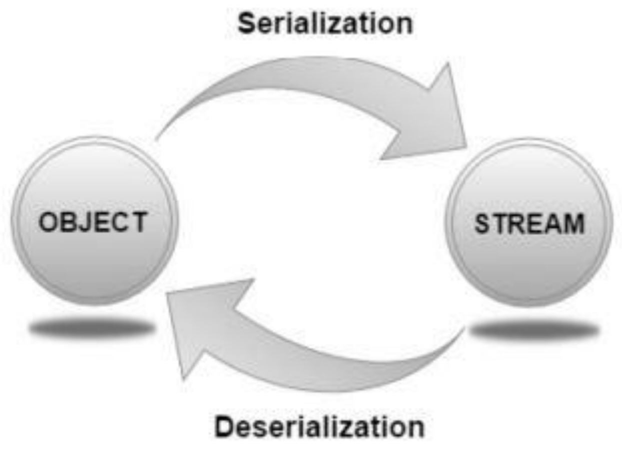
```
Hi I am learning Java. Currently learning IO and Streams
```

Whole Reader and Writer Class



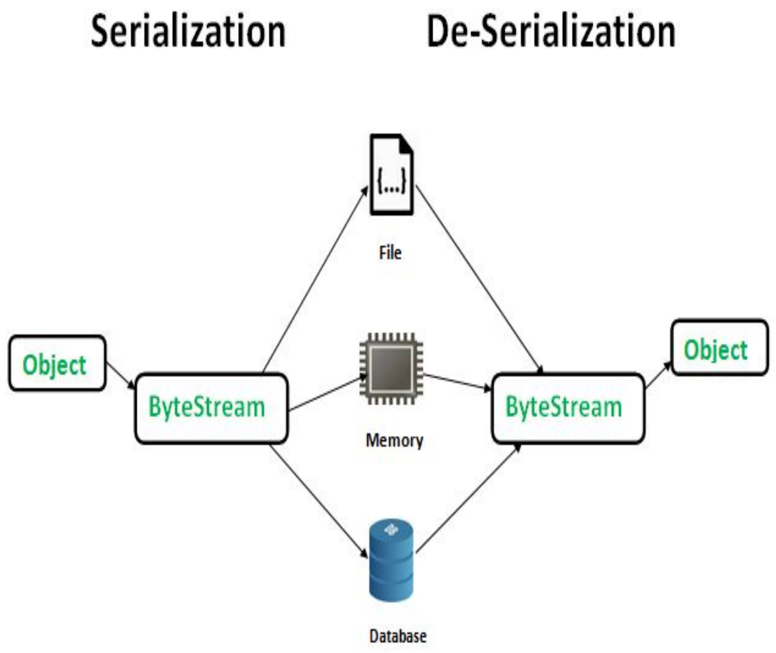
Serialization and Deserialization

- ❖ Serialization is a mechanism of converting the state of an object into a byte stream.
- ❖ Serialization is done to save/persist state of an object.
- ❖ Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory.

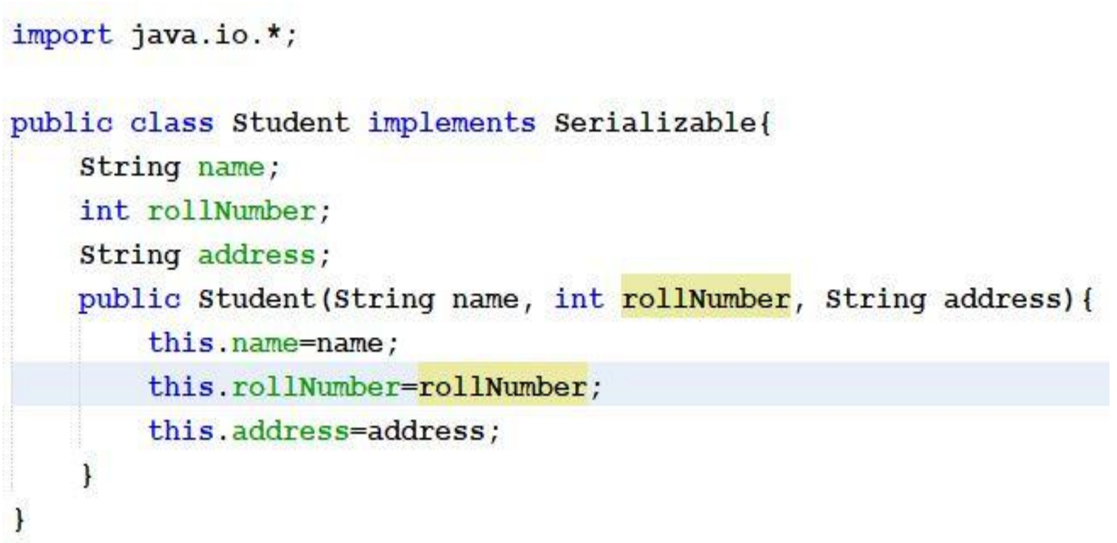


Serialization and Deserialization

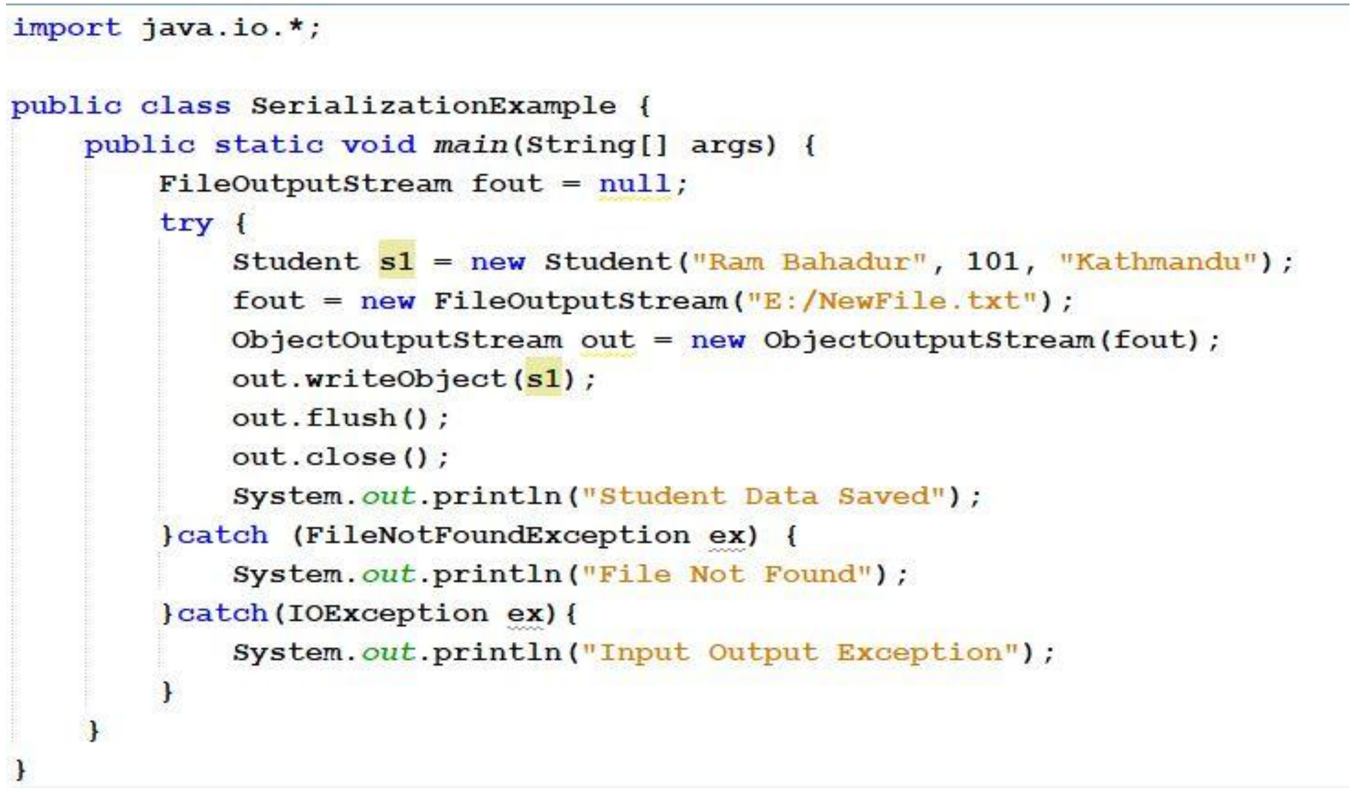
- ❖ The serialization and deserialization process is platform-independent, it means we can serialize an object in a platform and deserialize in different platform
- ❖ For serializing the object, we call the **writeObject() method of ObjectOutputStream**, and for deserialization we call the **readObject() method of ObjectInputStream class**
- ❖ **We must have to implement the Serializable interface for serializing the object**



Serialization Example

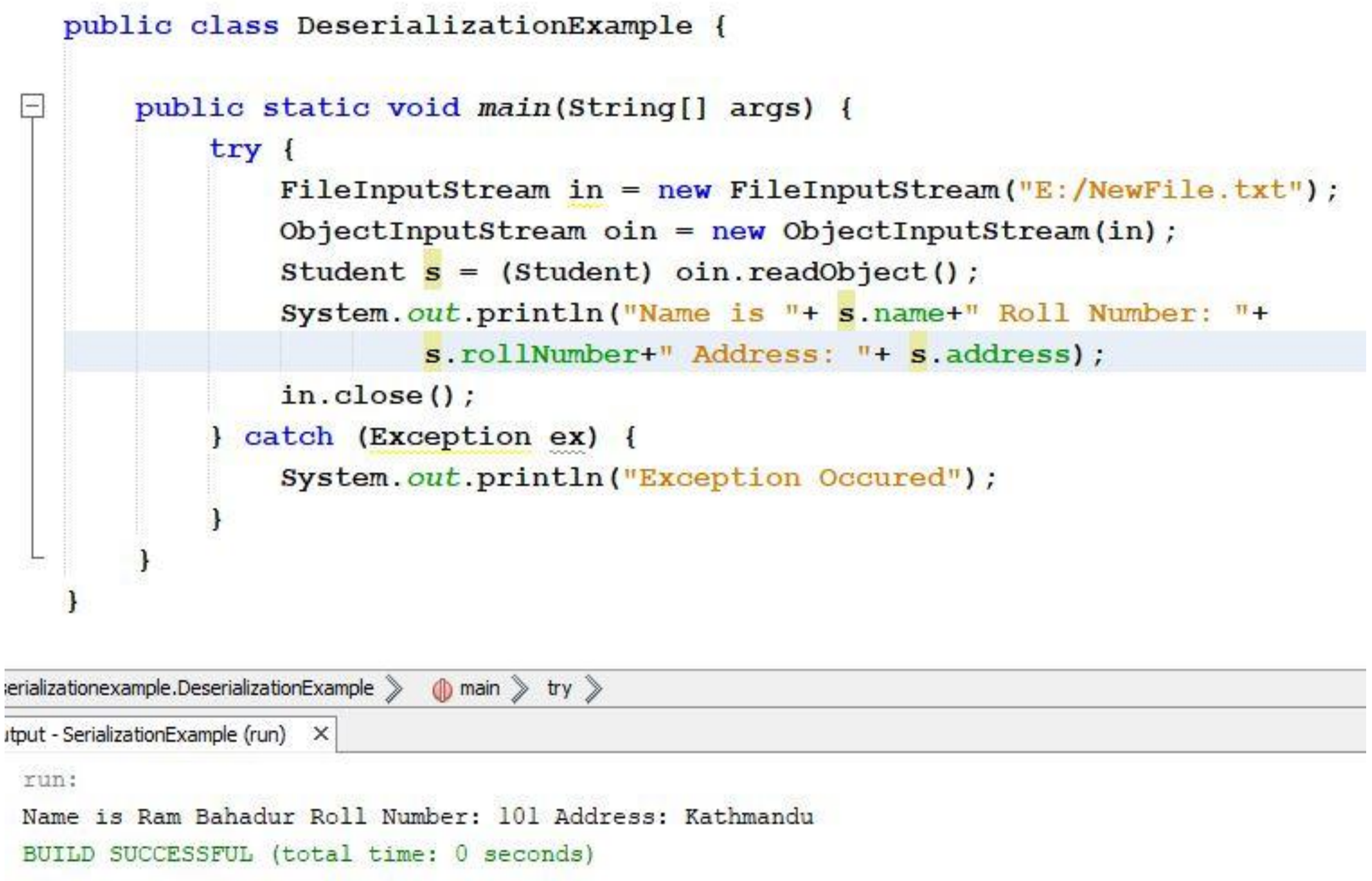


```java
import java.io.*;

public class Student implements Serializable{
    String name;
    int rollNumber;
    String address;
    public Student(String name, int rollNumber, String address){
        this.name=name;
        this.rollNumber=rollNumber;
        this.address=address;
    }
}
```



```java
import java.io.*;

public class SerializationExample {
    public static void main(String[] args) {
        FileOutputStream fout = null;
        try {
            Student s1 = new Student("Ram Bahadur", 101, "Kathmandu");
            fout = new FileOutputStream("E:/NewFile.txt");
            ObjectOutputStream out = new ObjectOutputStream(fout);
            out.writeObject(s1);
            out.flush();
            out.close();
            System.out.println("Student Data Saved");
        }catch (FileNotFoundException ex) {
            System.out.println("File Not Found");
        }catch(IOException ex){
            System.out.println("Input Output Exception");
        }
    }
}
```

Deserialization Example



```java
public class DeserializationExample {

    public static void main(String[] args) {
        try {
            FileInputStream in = new FileInputStream("E:/NewFile.txt");
            ObjectInputStream oin = new ObjectInputStream(in);
            Student s = (Student) oin.readObject();
            System.out.println("Name is "+ s.name+" Roll Number: "+
                    s.rollNumber+" Address: "+ s.address);
            in.close();
        } catch (Exception ex) {
            System.out.println("Exception Occured");
        }
    }
}
```

erializationexample.DeserializationExample
main
try
itput - SerializationExample (run)

```
run:
Name is Ram Bahadur Roll Number: 101 Address: Kathmandu
BUILD SUCCESSFUL (total time: 0 seconds)
```