Chapter 13

I/O, Try-with-Resources and other topics

- This chapter introduces one of Java's most important packages: **java.io**.
- ***java.io*** - this package supports Java's basic I/O system, including file I/O.

Streams

- Java programs perform I/O through streams.
- A stream is an **abstraction** that either produces or consumes information.
- An **input stream** can abstract many different kinds of input: from a disk file, a keyboard, or a network socket.
- An **output stream** may refer to the console, a disk file, or a network connection.
- Streams are a clean way to deal with I/O without having every part of your code understand the difference between a keyboard and a network.
- Java defines two types of streams: **byte** and **character**.
- To use any stream class, you must `import java.io` **package**.

i) Byte Stream

- They provide a convenient means for handling input and output of bytes.
- They can be used, for example, when **reading or writing binary data**.
- Using these you can store **characters**, **videos**, **audios**, **images** etc.
- Byte streams are defined by using two abstract classes: **InputStream** and **OutputStream**.
- These 2 classes have several concrete subclasses for handling the differences among various devices, such as disk files, network connections.
- The abstract classes InputStream and OutputStream define several key methods that the other stream classes implement.
- Two of the most important are **read()** and **write()**, which, respectively, read and write bytes of data.
- Table 13-1 lists down the byte stream classes. (The entire table need not be memorized)

Stream Class	Meaning
BufferedInputStream	Buffered input stream
BufferedOutputStream	Buffered output stream
ByteArrayInputStream	Input stream that reads from a byte array
ByteArrayOutputStream	Output stream that writes to a byte array
DataInputStream	An input stream that contains methods for reading the Java standard data types
DataOutputStream	An output stream that contains methods for writing the Java standard data types
FileInputStream	Input stream that reads from a file
FileOutputStream	Output stream that writes to a file
FilterInputStream	Implements **InputStream**
FilterOutputStream	Implements **OutputStream**
InputStream	Abstract class that describes stream input
ObjectInputStream	Input stream for objects
ObjectOutputStream	Output stream for objects
OutputStream	Abstract class that describes stream output
PipedInputStream	Input pipe
PipedOutputStream	Output pipe
PrintStream	Output stream that contains **print( )** and **println( )**
PushbackInputStream	Input stream that supports one-byte "unget," which returns a byte to the input stream
SequenceInputStream	Input stream that is a combination of two or more input streams that will be read sequentially, one after the other

Table 13-1 The Byte Stream Classes in **java.io**

ii) Character Stream

- They provide a convenient means for handling input and output of characters.
- These handle data in 16 bit Unicode.
- Using these you can read and write **text data** only.
- They are also defined using two abstract classes: **Reader** and **Writer**.
- Java has several concrete subclasses of each of these. Table 13-2 lists them down.
- The abstract classes Reader and Writer define several key methods like **read()** and **write()** that the other stream classes implement.

Stream Class	Meaning
BufferedReader	Buffered input character stream
BufferedWriter	Buffered output character stream
CharArrayReader	Input stream that reads from a character array
CharArrayWriter	Output stream that writes to a character array
FileReader	Input stream that reads from a file
FileWriter	Output stream that writes to a file
FilterReader	Filtered reader
FilterWriter	Filtered writer
InputStreamReader	Input stream that translates bytes to characters
LineNumberReader	Input stream that counts lines
OutputStreamWriter	Output stream that translates characters to bytes
PipedReader	Input pipe
PipedWriter	Output pipe
PrintWriter	Output stream that contains **print( )** and **println( )**
PushbackReader	Input stream that allows characters to be returned to the input stream
Reader	Abstract class that describes character stream input
StringReader	Input stream that reads from a string
StringWriter	Output stream that writes to a string
Writer	Abstract class that describes character stream output

Table 13-2 The Character Stream I/O Classes in **java.io**

Predefined Streams

- All Java programs automatically import the java.lang package.
- This package defines a class called **System**.
- System also contains three predefined stream variables: **in, out,** and **err**, which are declared as **public**, **static** and **final**.
- This means that they can be used by any other part of your program and without reference to a specific System object.
- **System.out** refers to the standard output stream. By default, this is the **console**.
- **System.in** refers to the standard input, which is the **keyboard** by default.
- **System.err** refers to the standard error stream, which also is the **console** by default.
- System.in is an object of type InputStream (byte stream)
- System.out and System.err are objects of type PrintStream (byte stream)

Reading Console Input

i) Using BufferedReader Class

- This method is used by wrapping the System.in (standard input stream) in an InputStreamReader which is wrapped in a BufferedReader.
- To read types other than string, we use functions like Integer.parseInt(), Double.parseDouble().

```java
// Java program to demonstrate BufferedReader
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class Main
{
    public static void main(String[] args) throws IOException
    {
        //Enter data using BufferReader
        BufferedReader reader =
                new BufferedReader(new InputStreamReader(System.in));

        // Reading data using readLine
        System.out.println("Enter your name: ");
        String name = reader.readLine();
        System.out.println("Enter your age: ");
        int age = Integer.parseInt(reader.readLine());
    }
}
```

ii) Using Scanner Class

```java
// Java program to demonstrate working of Scanner in Java
import java.util.Scanner;

class GetInputFromUser
{
    public static void main(String args[])
    {
        // Using Scanner for Getting Input from User
        Scanner in = new Scanner(System.in);

        String s = in.nextLine();
        System.out.println("You entered string "+s);

        int a = in.nextInt();
        System.out.println("You entered integer "+a);

        float b = in.nextFloat();
        System.out.println("You entered float "+b);
    }
}
```

iii) Using Console Class

```java
// Java program to demonstrate working of System.console()
// Note that this program does not work on IDEs as
// System.console() may require console
public class Main
{
    public static void main(String[] args)
    {
        // Using Console to input data from user
        String name = System.console().readLine();

        System.out.println(name);
    }
}
```

Note: Read examples on page 467- 469.

Writing Console Output

- Console output is most easily accomplished with print() and println().

Using PrintWriter Class

- PrintWriter defines several constructors. The one we will use is shown here:

  PrintWriter(OutputStream outputStream, boolean flushingOn)

- **outputStream** is an object of type OutputStream, and **flushingOn** controls whether Java flushes the output stream every time a println() method (among others) is called.
- If flushingOn is true, flushing automatically takes place. If false, flushing is not automatic.
- PrintWriter supports the print() and println() methods.

```
// Demonstrate PrintWriter
import java.io.*;

public class PrintWriterDemo {
  public static void main(String args[]) {
    PrintWriter pw = new PrintWriter(System.out, true);

    pw.println("This is a string");
    int i = -7;
    pw.println(i);
    double d = 4.5e-7;
    pw.println(d);
  }
}
```

The output from this program is shown here:

```
This is a string
-7
4.5E-7
```