### Terminologies

Stream is nothing but a flow of data, from river, small streams will be created, similarly from data source (files) we should always create a stream to read the data or to write the data

Data always moves in stream from source to destination (petrol from petrol pump to car tank also moves in fuel)

Ex:- to read the data from any file, we should create a stream

InputStream is to read data from data source (can be a file ..)

outputStream is to write the data to datasource (can be a file)

#### Serialization vs writing data to file

|  |  |
| --- | --- |
| serialization | writing as a plain text |
| Write the three serialized character objects to a file  Create a file and write three lines of text, one per character, separating the pieces of state with commas:  ..srGameCharacter  ..%gê8MÛIpowerLjava/lang/  String;[weaponst[Ljava/lang/  String;xp2tlfur[Ljava.lang.String;≠“VÁ  È{Gxptbowtswordtdustsq˜»tTrolluq˜tb  are handstbig axsq˜xtMagicianuq˜tspe  llstinvisibility  if u serialise humans cant read, but easy for JVM to understand | Write a plain-text file  Create a file and write three lines of text, one per character, separating the pieces of state with commas:  50,Elf,bow, sword,dust  200,Troll,bare hands,big ax  120,Magician,spells,invisibility |

### API

FileOutputStream ans objectOutputStream

FileOutputStreams write bytes to a file. ObjectOutputStreams turn objects into data that can be written to a stream.

If u want to write the objects directly then we have to choose ObjectOutputStream

|  |  |
| --- | --- |
| String fileName = "person.ser"; // Conventionally, .ser for serialized objects  // Create a Person object  Person person = new Person("Alice", 30, "New York");  try (FileOutputStream fileOut = new FileOutputStream(fileName);  ObjectOutputStream objectOut = new ObjectOutputStream(fileOut)) {  // Write the Person object to the file  objectOut.writeObject(person);  System.out.println("Person object has been serialized and saved to " + fileName); | only ObjectOutputStream lets you to write objects  but it cant directly write to a file it needs to be fed a helper  This is actually called chaining one stream to another stream  out.writeObject(emp1);: This is the magic! It takes the emp1 object, converts its state (all non-transient fields) into a byte stream, and writes it to the employee.ser file  serialization  is nothing but process of writing object data to a file  reading:- deserialization – is nothing but process of reading the data from file and injecting that data to object |

Connection streams (FileOutputStream, network socket, etc.), represent a connection to a source or destination while chain streams(ObjectOutputStream) can’t connect on their own and must be chained to a connection stream.

Often, it takes at least two streams hooked together to do something useful—one to represent the connection and another to call methods on

FileOutputStreams write bytes to a file. ObjectOutputStreams turn objects into data that can be written to a stream. So we make a FileOutputStream (a connection stream) that lets us write to a file, and we hook an ObjectOutputStream (a chain stream) on the end of it. When we call writeObject() on the ObjectOutputStream, the object gets pumped into the stream and then moves to the FileOutputStream where it ultimately gets written as bytes to a file.

Reading file from buildpath and from file system

|  |  |
| --- | --- |
| java 7 - Paths.get() is from java 7 | java 11- path.of() is from java 11 like List.of |
| Path path = Paths.get("folder", "file.txt");  Internally, Paths.get(...) just calls FileSystems.getDefault().getPath(...). | Path path = Path.of("folder", "file.txt");  Also delegates internally to FileSystems.getDefault().getPath(...) |

Both Return the same type (java.nio.file.Path)

 **Java 11+**: Prefer Path.of() — it’s shorter, clearer, and more modern.

 **Java 7–10**: Use Paths.get() — Path.of() didn’t exist yet.

|  |  |
| --- | --- |
| Reading from build path | reading from file system for debugging purposes |
| File file = Path.of("src/main/resources/book.xml").toFile();  ObjectMapper mapper = new XmlMapper();  JsonNode root = mapper.readTree(file); | File file = *Path*.*of*(System.*getProperty*("user.home"), "OneDrive","Documents", "mani.JSON").toFile(); JSONObject obj = (JSONObject) JSONValue.*parse*(new FileReader(file)); |

### How to load data from property file

Properties props = new Properties();

InputStream inputStream = new FileInputStream(AppConfigs.kafkaConfigFileLocation);

props.load(inputStream);

#### Reading line by line from file

// [**Fig. 11.3**](https://learning.oreilly.com/library/view/java-for-programmers/9780137574834/ch11.xhtml#ch11fig03): ReadTextFile.java

**2** // Reading a text file using a Scanner.

**3** import java.nio.file.Path;

**4** import java.util.Scanner;

**5**

**6** public class ReadTextFile {

**7** public static void main(String[] args) {

**8** // Path to clients.txt in user's Documents folder

**9** Path filePath = Path.of(System.getProperty("user.home"),

**10** "Documents", "clients.txt");

**11**

**12** // open clients.txt, read its contents and close the file

**13** try (var input = new Scanner(filePath)) {

**14** System.out.printf("%-10s%-12s%10s%n",

**15** "Account", "Name", "Balance");

**16**

**17** // read record from file

**18** while (input.hasNext()) { // while there is more to read

**19** // display record contents

**20** System.out.printf("%-10d%-12s%10.2f%n",

**21** input.nextInt(), input.next(), input.nextDouble());

**22** }

**23** }

**24** catch (Exception e) {

**25** System.err.printf(

**26** "Error reading file: %s%n", e.getMessage());

**27** }

**28** }

**29** }

### How to read all lines from csv file line by line and store in a string

Below will read all lines at once

Path path = Paths.*get*("data/NSE05NOV2018BHAV.csv");  
List<String> strings = Files.*readAllLines*(path);  
System.*out*.println("size of the list is--> "+strings.size());

 java.nio.file.Files: Contains static utility methods for operating on files and directories, such as copying, moving, deleting, reading, writing, and checking existence.

 java.nio.file.Path: Represents a path to a file or directory. It's the modern replacement for java.io.File for path representation.

 java.nio.file.Paths: A utility class that provides static methods to obtain Path instances.

 java.nio.file.StandardCopyOption: An enum that defines options for Files.copy() and Files.move() methods.

 java.nio.file.LinkOption: An enum that defines options for resolving symbolic links.

 java.nio.file.FileVisitResult, SimpleFileVisitor, FileVisitor, BasicFileAttributes: These are used in the Files.walkFileTree() method for traversing file trees (like when doing a recursive directory copy).

Copying files using java 7 way

*Java 7 introduced java.nio.file package – Files, paths (java.io.File is from java 1.0)- here nio- means new io (old api is blocking)*

|  |  |
| --- | --- |
| *copying using path* | copying using input stream |
| *Path* sourcePath = Paths.*get*("C:\\Users\\Manideep\\Downloads\\Lalitha Lavanya R.pdf"); *Path* destPath = Paths.*get*("C:\\Users\\Manideep\\Downloads\\Radhika-v1.pdf");  *Path* destPath1 = Files.*copy*(sourcePath, destPath); System.***out***.println("files has been copied"); | // we should always create a stream from file *InputStream* fileInputStream = new FileInputStream("C:\\Users\\Manideep\\Downloads\\Lalitha Lavanya R.pdf"); Files.*copy*(fileInputStream, destPath, *StandardCopyOption*.***REPLACE\_EXISTING***); System.***out***.println("files has been copied"); |