
LETS LEARN JAVA
FILES

Training Plan - Advanced

Topics to be covered
Java I/O and File handling FileInputStream/FileOutputStream Paths and Files BufferedStreams for better performance Pointed Exception Handling in file I/O. Try with resource

Files

Files are required to store data. Following are the key points to consider

1. Small Files >> Read all at once
2. Large Files >> User Buffered Reader based approach

Files can also be Character based as well as Object Based

1. Normal files (Store and retrieve Lines of data as strings)
2. Object based Files

Reading a small file using Java 8 File API

```
public class ReadingFiles {
    public static void main(String[] args) throws IOException {
        String directory = "C:\\Users\\HP\\Desktop\\JavaFiles";
        String fileName = "SampleCSVFile.csv";

        List<String> lines = Files.readAllLines(Paths.get(directory, fileName));

        for (String rec : lines) {
            System.out.println(rec);
        }

    }
}
```

Reading a large file using Buffering

```
public class ReadingLargeFiles {
    public static void main(String[] args) throws IOException {

        BufferedReader reader = Files
                .newBufferedReader(Paths.get("C:\\Users\\HP\\Desktop\\JavaFiles\\SampleCSVFile.csv"));

        String line = null;

        while ((line = reader.readLine()) != null) {
            System.out.println(line);
        }

    }
}
```

Making an Object Serializable to be stored

```java
public class Employee implements Serializable {

    private static final long serialVersionUID = 32278783228503671 30L;
    private Long id;
    private String firstName;
    private String lastName;
    private Boolean isMarried;
    private Character gender;
    private LocalDate dateOfBirth;
    private LocalDate dateOfJoining;
    private Integer salary;
    private String region;
    private Integer age;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
```

If you try to store an Object into a file without making it implement the Serializable interface, then Java will throw an Exception and will not create/read from the file.

The serialVersionUID has to be same while reading the file and while writing objects into it.

Storing data into Object File

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

        try (ObjectOutputStream outStream = new ObjectOutputStream(
                Files.newOutputStream(Paths.get("C:\\Users\\HP\\Desktop\\JavaFiles\\EmployeeFile.data")))) {
            List<String> lines = Files.readAllLines(Paths.get("C:\\Users\\HP\\Desktop\\JavaFiles\\SampleCSVFile.csv"));

            for (String rec : lines) {
                Employee emp = Employee.mapToEmployee(rec);
                outStream.writeObject(emp);
            }
            System.out.println("Object file created from the source file");
        } catch (IOException ex) {
            ex.printStackTrace();
        }

    }
}
```

Reading from Object File

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

        try (ObjectInputStream inStream = new ObjectInputStream(
            Files.newInputStream(Paths.get("C:\\Users\\HP\\Desktop\\JavaFiles\\EmployeeFile.data")))) {

            Employee emp = null;
            while ((emp = (Employee) inStream.readObject()) != null) {
                System.out.println(emp);
            }
        } catch (IOException | ClassNotFoundException ex) {
        }

    }
}
```

Try with resources (highlighted) will ensure the steam is gracefully closed after the execution is over, and even if there is an exception thrown

Reading from file and parsing into object

```java
public static void main(String[] args) throws IOException {

    List<String> allEmployeeRecords = Files
            .readAllLines(Paths.get("C:\\Users\\HP\\Desktop\\JavaFiles", "SampleCSVFile.csv"));

    List<Employee> employeeList = new ArrayList<>();
    for (String record : allEmployeeRecords) {
        employeeList.add(mapToEmployee(record));
    }
```

```java
private static Employee mapToEmployee(String recordString) {
    String[] recordElements = recordString.split(",");
    Employee emp = new Employee();
    emp.setId(Long.valueOf(recordElements[0]));
    emp.setFirstName(recordElements[1]);
    emp.setLastName(recordElements[2]);
    emp.setIsMarried(Boolean.valueOf(recordElements[3]));
    emp.setGender(recordElements[4].charAt(0));
    emp.setDateOfBirth(getDateFromString(recordElements[5]));
    emp.setDateOfJoining(getDateFromString(recordElements[6]));
    emp.setSalary(Integer.valueOf(recordElements[7]));
    emp.setRegion(recordElements[8]);
    return emp;
}

private static LocalDate getDateFromString(String inputString) {
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("M/d/yyyy");
    LocalDate dateValue = LocalDate.parse(inputString, formatter);
    return dateValue;
}
```

Lets program the below

Repeat the exercise on collections, but read the data from a file instead of hardcoding the same. Also, store output of question 3 into a file as well.

1. Find sum of salary of all unmarried people
2. Find out how many people younger than 40 are earning more than the average payout at the company
3. Create a summary of number of people, total experience and average salary by region
4. Find employees with names more than 5 characters, having an 'e' in them, married, and having less than average salary