CSCI 2120:

Software Design & Development II

UNIT3: I/O management

io api

Externalization

Overview

1. Introduction
2. Externalizable interface
3. Externalizable interface methods
4. Externalizable examples
5. Difference between Serializable & Externalizable

Introduction

- **Externalization in Java** is a process that improves the performance of serialization and deserialization mechanisms by complete control over what data fields are serialized and deserialized.

- In other simple words, externalization is used to customize serialization and deserialization processes. It provides complete control over the serialization and deserialization tasks.

- Java supports the externalization process by the `Externalizable` interface.

- Implementing the `Externalizable` interface by a class provides more control in reading and writing objects from/to a stream.

Introduction

- The **main advantage** of externalization over the serialization process is that everything is taken care of by the programmer. JVM has no control over it. The complete serialization control goes to the application.

- Based on our requirements, we can serialize either the whole data field or a piece of the data field using the `Externalizable` interface which can help to improve the performance of the application.

- In the *serialization* process, JVM is taken care of everything. That is, JVM takes care of details of reading/writing Serializable objects from/to a stream.

- The programmer has no control over them. We can not store a piece of data that may create performance problems. To overcome this problem, Java introduced an externalization mechanism.

Externalizable Interface

`Externalizable` interface that extends `Serializable` interface.

The general syntax to declare an `Externalizable` interface in Java is as follows:

```
public interface Externalizable
       extends Serializable
```

A class must implement the `Externalizable` interface for complete control over the serialization process. To externalize objects, only the state of the object's class is automatically stored by the stream. The class that implements `Externalizable` interface is responsible for writing and reading its contents or data.

Externalizable Interface

The complete syntax of the `Externalizable` interface that extends `Serializable` interface is as follows:

```
public interface Externalizable extends Serializable {

   public void writeExternal(ObjectOutput out) throws IOException;
   public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException;

}
```

A `writeExternal()` method must be implemented to store the state (field) of objects in a class that implements `Externalizable` interface.

Similarly, `readExternal()` method must be implemented to read data written by `writeExternal()` method from the stream and restore the state of objects.

Externalizable interface methods

`Externalizable` interface is not a marker interface like `Serializable` interface. So, it provides two methods that are as follows:

Externalizable interface methods - `readExternal`

1. void readExternal(ObjectInput inStream):
The `readExternal()` method restores the calling object's contents by calling various methods on the specified `ObjectInput` in.

In other simple words, we call `readExternal()` method when we want to read an object's fields from a stream. We need to write logic to read an object's fields inside the `readExternal()` method.

The general syntax to declare `readExternal()` method with the specified object input stream is as follows:

```
public void readExternal(ObjectInput inStream) throws IOException, ClassNotFoundException {
   // Here, write the logic to read an object's fields from the stream.
}
```

Externalizable interface methods - `readExternal`

In this method, `inStream` is the byte stream from which the object is to be read. `ObjectInput` is a sub-interface of `DataInput` and is implemented by `ObjectInputStream.`

We can use `readBoolean(), readByte(), readInt(), readLong()` methods for primitive data types and `readObject()` method for String, Arrays, or any of the custom classes.

The `readExternal()` method throws `IOException` when an I/O error occurs. If the class of object being restored does not find, `ClassNotException` will be thrown.

Externalizable interface methods - `writeExternal`

2. void writeExternal(ObjectOutput outStream):
The `writeExternal()` method saves the calling object's contents by calling various methods on the specified ObjectOutput outStream.

In simple words, `writeExternal()` method is used when we want to write an object's fields to a stream. We need to write the logic to write data fields inside `writeExternal()` method.

The general syntax to declare `writeExternal()`method with the specified object output stream is as follows:

```
public void writeExternal(ObjectOutput outStream) throws IOException {
   // Here, write the logic to write object fields to a stream.
}
```

Externalizable interface methods - `writeExternal`

`outStream` is the byte stream to which the object is to be written. `ObjectOutput` is a sub-interface of `DataOutput` and is implemented by `ObjectOutputStream`.

We can use `writeBoolean(), writeByte(), writeInt(), writeLong()` methods for primitive data types and `writeObject()` method for strings, arrays, and custom classes.

This method can throw an `IOException` when an I/O error occurs.

Example 1: Externalizable - User class

1. Let’s take a simple example program where we will understand step by step how to serialize and deserialize `Externalizable` object fields. Look at the source code to understand better.

Example 1: Externalizable - User class

```java
import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;

// A User class that implements Externalizable interface.
public class User implements Externalizable {
   String name;
   int age;

   @Override
   public void writeExternal(ObjectOutput outStream) throws IOException {
       // Writing name and age to the stream.
       outStream.writeObject(name);
       outStream.writeInt(age);
   }
   @Override
   public void readExternal(ObjectInput inStream) throws ClassNotFoundException, IOException {
       // Reading name and age in the same order in which they were written.
       name = (String)inStream.readObject();
       age = inStream.readInt();
   }
}
```

Example 1: Externalizable - User class

In this source code, Java will pass the reference of the object output stream and object input stream to the `writeExternal()` and `readExternal()` methods of the `User` class respectively.

In the `writeExternal()` method, we write the name and age fields to the object output stream. The `writeObject()` and `writeInt()` methods are used to write `Externalizable` object fields such as name and age to the object output stream.

In the `readExternal()` method, we read the `Externalizable` object fields such as name and age from the stream and set them in the name and age instance variables.

Example 1: Externalizable - Serialize User

Let’s create a class named `SerializeUser` where we will serialize total object fields of the `User` class that implements `Externalizable` interface.

Example 1: Externalizable - Serialize User

```java
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class SerializeUser {
   public static void main(String[] args) throws IOException {
       // Create an object of Person class.
       User u = new User();
       u.name = "Ted";
       u.age = 99;

       // Create File object.
       File file = new File("./src/userfile.dat");
       FileOutputStream fos = new FileOutputStream(file);
       ObjectOutputStream oos = new ObjectOutputStream(fos);

       // Write or serialize objects to object output stream.
       oos.writeObject(u);
       oos.flush();
       System.out.println("Data has been written successfully...");
       // Print the output path of the file.
       System.out.println("Data are written to: " +file.getAbsolutePath());
   }
}
```

Example 1: Externalizable - Serialize User

Output:

```
Data has been written successfully...
Data are written to: /Users/ted/IdeaProjects/Lecture24-ObjectStream/./src/userfile.dat
```

Note:

a) In the serialization process, JVM first checks for `Externalizable` interface. If objects support `Externalizable` interface, JVM serializes objects using `writeExternal()` method.

If objects do not support `Externalizable` but implement `Serializable`, objects are stored using `ObjectOutputStream`.

b) For serializable objects, JVM serializes the only instance variables that are not declared with the `transient` keyword.

c) For externalizable objects, we have full control over what pieces of data have to serialize and what to not serialize.

Example 1: Externalizable - Deserialize User

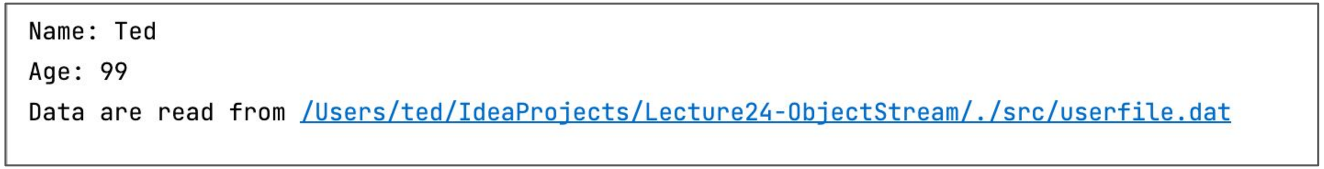
Let’s create a class `DeserializeUser` where we will deserialize total object fields of the `User` class that implements `Externalizable` interface.

Example 1: Externalizable - Deserialize User

```java
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class DeserializeUser {
   public static void main(String[] args) throws IOException, ClassNotFoundException {
       // Create a input file object.
       File file = new File("./src/userfile.dat");
       FileInputStream fis = new FileInputStream(file);
       ObjectInputStream ois = new ObjectInputStream(fis);

       // Deserialize or read objects of Person class.
       User u = (User)ois.readObject();

       // Displaying objects that are read from file.
       System.out.println("Name: " +u.name);
       System.out.println("Age: " +u.age);

       // Print the input path of the file.
       System.out.println("Data are read from " +file.getAbsolutePath());
   }
}
```

Example 1: Externalizable - Deserialize User

Output:



```
Name: Ted
Age: 99
Data are read from /Users/ted/IdeaProjects/Lecture24-ObjectStream/./src/userfile.dat
```

Note:

a) In the serialization process, JVM first checks for `Externalizable` interface. If objects support `Externalizable` interface, JVM serializes objects using `writeExternal()` method.

If objects do not support `Externalizable` but implement `Serializable`, objects are stored using `ObjectOutputStream`.

b) For serializable objects, JVM serializes the only instance variables that are not declared with the `transient` keyword.

c) For externalizable objects, we have full control over what pieces of data have to serialize and what to not serialize.

Example 2: writeExternal() & readExternal()

2. Let’s take another example program where we will serialize a piece of data fields in the `writeExternal()` method and deserialize a piece of data fields in the `readExternal()` method using an `Externalizable` interface.

Example 2: writeExternal() & readExternal() - Worker

```java
import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;

public class Worker implements Externalizable {
   String name;
   int id;
   double salary;

   Worker() {}

   Worker(String name, int id, double salary) {
       this.name = name;
       this.id = id;
       this.salary = salary;
   }
   @Override
   public void writeExternal(ObjectOutput outStream) throws IOException {
       // Serializing only id and salary.
       outStream.writeInt(id);
       outStream.writeDouble(salary);
   }
   @Override
   public void readExternal(ObjectInput inStream) throws ClassNotFoundException, IOException {
       // Order of reads must be the same as the order of writes.
       id = inStream.readInt();
       salary = inStream.readDouble();
   }
}
```

Example 2: writeExternal() & readExternal() - Serialize

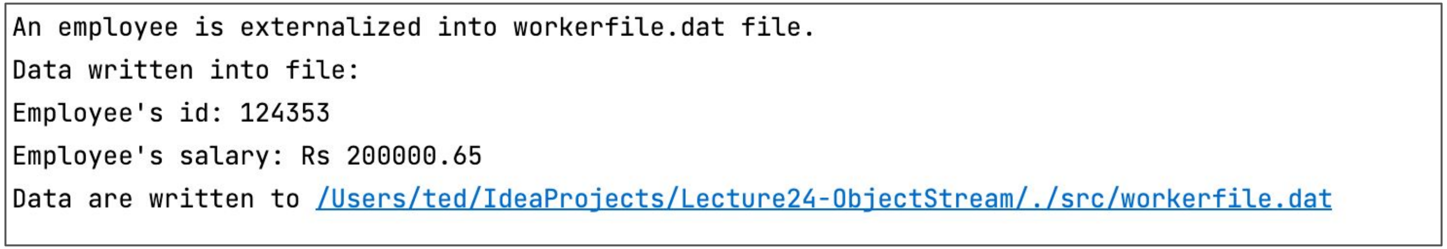
Let’s create a program where we will serialize an `Worker` object that implements an `Externalizable` interface.

Example 2: writeExternal() & readExternal() - Serialize

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class SerializeWorker {
   public static void main(String[ ] args) throws IOException {
       // Create an object of class Employee.
       Worker emp = new Worker("John", 124353, 200000.65 );

       File file = new File("./src/workerfile.dat");
       FileOutputStream fos = new FileOutputStream(file);
       ObjectOutputStream oos = new ObjectOutputStream(fos);

       emp.writeExternal(oos); // Externalizing employee.
       System.out.println("An employee is externalized into workerfile.dat file.");

       // Display the serialized data on the Standard Output.
       System.out.println("Data written into file:");
       System.out.println("Employee's id: " +emp.id);
       System.out.println("Employee's salary: "+"Rs " +emp.salary);

       // Print output path.
       System.out.println("Data are written to " +file.getAbsolutePath());
   }
}
```

Example 2: writeExternal() & readExternal() - Serialize

Output:



```
An employee is externalized into workerfile.dat file.
Data written into file:
Employee's id: 124353
Employee's salary: Rs 200000.65
Data are written to /Users/ted/IdeaProjects/Lecture24-ObjectStream/./src/workerfile.dat
```

Explanation:

As you can observe in this program, we have serialized only two data fields `id` and `salary`. Thus, we have control over the serialization

Example 2: writeExternal() & readExternal() - Deserialize

Let’s create a program where we will deserialize an `Worker` object that implements an `Externalizable` interface.

Example 2: writeExternal() & readExternal() - Deserialize

```java
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;

public class DeserializeWorker {
   public static void main(String[ ] args) throws IOException, ClassNotFoundException {
       File file = new File("./src/workerfile.dat");
       FileInputStream fis = new FileInputStream(file);
       ObjectInputStream ois = new ObjectInputStream(fis);

       // Deserializing or reading one object.
       Worker emp = new Worker();
       emp.readExternal(ois);

       // Displaying data (objects) that are read.
       System.out.println("Employee's id: " +emp.id);
       System.out.println("Employee's salary: " +emp.salary);

       // Print input path.
       System.out.println("Data read from " +file.getAbsolutePath());
       ois.close();
   }
}
```

Example 2: writeExternal() & readExternal() - Deserialize

Output:

```
Employee's id: 124353
Employee's salary: Rs 200000.65
Data read from ./src/workerfile.dat
```

Difference between Serializable and Externalizable

1. `Serializable` is a marker interface whereas, `Externalizable` is not a marker interface. It provides two methods: `writeExternal()` and `readExternal()`.

2. In the case of `Serializable,` default serialization process is used. While, in the case of Externalization, custom serialization process is used that is implemented by the application.

3. In the serialization process, everything is taken care of by JVM and the programmer doesn't have any control. While in the case of externalization process, everything is taken care of by the programmer and JVM doesn't have any control.

4. `Externalizable` interface is generally used when we want to store the output data in custom format that is different from java default serialization format like csv, database, flat file, XML, etc.

5. In externalization process, `readExternal()` and `writeExternal()` methods are used to perform serialization.

6. `Externalizable` interface can improve the performance of serialization process if used correctly.

END