Binary Files in Java

1. Overview

In this tutorial, we're going to show how to read and write Java objects from/to binary files. To learn how to read and write in text files, we have the article Text Files in Java.

2. What is a Binary File?

Binary files are files that contain binary data. They usually have a .dat extension. An advantage of using binary files is that we can store entire objects (even the whole collection) and later we can just as easily read them from a file.

Additionally, we don't need to add any external dependency to perform actions on binary files. The classes we're going to use come with Java API.

2. Serialization

We should be aware of one rule when working with binary files: objects we intend to store in binary files should be serializable. The term serialization represents a mechanism that allows us to transform an object into a byte stream. The reverse process is deserialization.

Furthermore, we can make our object serializable by simply implementing the java.io.Serializable interface. This interface doesn't contain any methods we need to implement but rather it serves as a marker.

Let's create a class named Item that implements a Serializable interface:

```
class Item implements Serializable {

    private String code;
    private String name;
    private BigDecimal price;

    public Item(String code, String name, BigDecimal price) {
        this.code = code;
        this.name = name;
        this.price = price;
    }

    public String getCode() {
        return code;
    }

    public void setCode(String code) {
        this.code = code;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public BigDecimal getPrice() {
        return price;
    }

    public void setPrice(BigDecimal price) {
        this.price = price;
    }

    /**
     * Returns <code>String</code> representation of an object
     */
```

```
    @Override
    public String toString() {
        return "Item{" +
                "code='" + code + '\'' +
                ", name='" + name + '\'' +
                ", price=" + price +
                '}';
    }
}
```

3. Writing in Binary File

Now, to write in binary files, we are going to use classes from Java API – java.io.FileOutputStream and java.io.ObjectOutputStream.

A FileOutputStream represents an output stream for writing data to a specific file. We can store raw bytes (such as image data). For writing in text files, we should consider using FileWriter instead.

An ObjectOutputStream writes Java objects to an OutputStream. Once written, the objects can be read using ObjectInputStream.

However, we're going to use the writeObject() method to write an object to the stream.

Let's take a look at an example of writing an object in a binary file:

```
class WriteExample {

    public static void main(String[] args) {

        String filePath = "files/item.dat";
        Item item = new Item("123456", "Apple", BigDecimal.valueOf(15.8));

        writeToFile(item, filePath);
    }

    public static void writeToFile(Item item, String filePath) {
        try (FileOutputStream fileOut = new FileOutputStream(filePath);
             ObjectOutputStream objectOut = new ObjectOutputStream(fileOut)) {
            objectOut.writeObject(item);
        } catch (Exception e) {
            System.err.println("Error - " + e);
        }
    }

}
```

The code snippet above will create a binary file item.dat (if the file doesn't exist) and will write an object into that file.

The data inside binary files can't be easily interpreted by humans. We can understand some parts of the data, but not all of them. If we need to create a file that is readable for humans, we should consider using text files instead.

The data stored in the file might look like the following:

•• sr com.peterlic.Item •\•y•? L codet Ljava/lang/String;L nameq ~ L pricet Ljava/math/BigDecimal;xpt 123456t

••• xp sr java.math.BigInteger••••;• I bitCountI bitLengthI firstNonzeroByteNumI lowestSetBitI signum[magnitud

4. Reading from Binary File

We'll use java.io.FileInputStream and java.io.ObjectInputStream classes to read objects from a file. Both classes come with Java API.

Objects can be read from binary files by calling the readObject() method. This method will return an object of the type Object. We'd need to perform a cast operator to convert a java.lang.Object to an Item type.

Let's see how we can read the Item we stored from the item.dat file:

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

        String filePath = "files/item.dat";

        Item item = readFromFile(filePath);
        System.out.println(item);

    }

    public static Item readFromFile(String filePath) {
        try (FileInputStream fis = new FileInputStream(filePath);
             ObjectInputStream ois = new ObjectInputStream(fis)) {
            Object obj = ois.readObject();
            return (Item) obj;
        } catch (Exception e) {
            System.err.println("Error - " + e);
            return null;
        }
    }
}
```

Finally, we'll get the item and write its attributes in a console.

Item{code='123456', name='Apple', price=15.8}

5. Conclusion

In this short tutorial, we explained how objects can be stored and read from binary files.

As always, the entire code can be found over on GitHub.

Read More