Java-Unit-5

20 March 2025 11:02

Q1. Differentiate between byte streams and character streams ?

In Java the streams are used for input and output operations by allowing data to be read from or written to a source or destination

Java offers two types of streams:

1. character streams
2. byte streams.

1. Character Streams:

Character streams are designed to address character based records, which includes textual records inclusive of letters, digits, symbols, and other characters

These streams are represented by way of training that quit with the phrase "Reader" or "Writer" of their names, inclusive of
FileReader,
BufferedReader,
FileWriter, and
BufferedWriter.

2. Byte Streams:

Byte streams are designed to deal with raw binary data, which includes all kinds of data, including characters, pictures, audio, and video.
These streams are represented through cclasses that cease with the word "InputStream" or "OutputStream" of their names, along with
FileInputStream,
BufferedInputStream,
FileOutputStream and
BufferedOutputStream.

Here are the some of the differences listed:

Aspect	Character Streams	Byte Streams
Data Handling	Handle character-based data	Handle raw binary data
Representation	Classes end with "Reader" or "Writer"	Classes end with "InputStream" or "OutputStream"
Suitable for	Textual data, strings, human-readable info	Non-textual data, binary files, multimedia
Character Encoding	Automatic encoding and decoding	No encoding or decoding
Text vs non-Text data	Text-based data, strings	Binary data, images, audio, video
Performance	Additional conversion may impact performance	Efficient for handling large binary data
Handle Large Text Files	May impact performance due to encoding	Efficient, no encoding overhead
String Operations	Convenient methods for string operations	Not specifically designed for string operations
Convenience Methods	Higher-level abstractions for text data	Low-level interface for byte data
Reading Line by Line	Convenient methods for reading lines	Byte-oriented, no built-in line-reading methods
File Handling	Read/write text files	Read/write binary files
Network Communication	Sending/receiving text data	Sending/receiving binary data
Handling Images/Audio/Video	Not designed for handling binary data directly	Suitable for handling binary multimedia data
Text Encoding	Supports various character encodings	No specific text encoding support

==

Q2. Write a program to copy the content of one file to another file using 1. Byte stream and character stream

```java
import java.io.CharArrayReader;
import java.io.IOException;

public class CharacterStreamExample
{
    public static void main(String[] args) {
        // Creates an array of characters
        char[] array = {'H','e','l','l','o'};
        try {
            CharArrayReader reader=new CharArrayReader(array);
            System.out.print("The characters read from the reader:");
            int charRead;
            while ((charRead=reader.read())!=-1) {
                System.out.print((char)charRead+",");
            }
            reader.close();
        } catch (IOException ex)
{
            ex.printStackTrace();
        }
    }
}
```

Output:

The characters read from the reader:H,e,l,l,o,

Example code for Byte Stream:

FileName: ByteStreamExample.java

```java
import java.io.ByteArrayInputStream;

public class ByteStreamExample
{
    public static void main(String[] args)
      {
        // Creates the array of bytes
        byte[] array = {10,20,30,40};
        try {
            ByteArrayInputStream input=new ByteArrayInputStream(array);
            System.out.println("The bytes read from the input stream:");
            for (int i=0;i<array.length;i++)
            {
                // Reads the bytes
                int data=input.read();
                System.out.print(data+",");
            }
            input.close();
        } catch (Exception ex)
{
            ex.getStackTrace();
        }
    }
}
```

Output:

The bytes read from the input stream:10,20,30,40.

==

Q3. What is file class? What is the purpose Java file class of it ?

Java File Class

The File class is an abstract representation of file and directory pathname. A pathname can be either absolute or relative.
The File class have several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory etc.

From <https://www.tpointtech.com/java-file-class>

Java File Example 1

```java
import java.io.*;
public class FileDemo {
    public static void main(String[] args) {

        try {
```

```
        File file = new File("javaFile123.txt");
        if (file.createNewFile()) {
            System.out.println("New File is created!");
        } else {
            System.out.println("File already exists.");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

  }
}
```

Output:
New File is created!

From <https://www.tpointtech.com/java-file-class>

===

Q4. What is Serialization and Deserialization . Explain Suitable example.

Serialization and Deserialization in Java

Serialization in Java is a mechanism for writing an object's state into a byte stream.
It is mainly used in Hibernate, RMI, JPA, EJB, and JMS technologies.

For serializing the object, we call the **writeObject()** method of the ObjectOutputStream class,

Java's serialization feature transforms an object into a stream of bytes, making it easier to store or send over a network. Many different technologies, including Hibernate, RMI (Remote Method Invocation), JPA (Java Persistence API), EJB (Enterprise JavaBeans), and JMS (Java Message Service), heavily utilize this method.

We must implement the Serializable interface for serializing the object.

Advantages of Java Serialization

It is mainly used to travel object's state on the network (that is known as marshalling).

64. **Platform Independence:** There are no compatibility problems when transferring serialized objects between separate Java virtual machines (JVMs) running on different platforms. Java Serialization is an effective technique for inter-system communication because of its platform independence.
65. **Network Communication:** Java Serialization facilitates the transmission of object data over a network. This process, known as marshalling, allows objects to be serialized into a byte stream and sent across a network to be reconstructed on another machine. It is crucial for applications involving distributed systems, such as client-server architectures and web services.
66. **Object Persistence:** Serialization allows for indefinite storage of object state. Data can be kept between program executions by using serialized objects, which can be saved to a disc or a database and then retrieved.

Example of Java Serialization

In this example, we are going to serialize the object of ***Student*** class from above code. The writeObject() method of ObjectOutputStream class provides the functionality to serialize the object. We are saving the state of the object in the file named f.txt.

Persist.java

```
import java.io.*;
class Persist{
 public static void main(String args[]){
  try{
  //Creating the object
  Student s1 =new Student(211,"ravi");
  //Creating stream and writing the object
  FileOutputStream fout=new FileOutputStream("f.txt");
  ObjectOutputStream out=new ObjectOutputStream(fout);
  out.writeObject(s1);
  out.flush();
  //closing the stream
 out.close();
 System.out.println("success");
 }catch(Exception e){System.out.println(e);}
}
}
```

Output:

```
success
```

Explanation

The above Java code illustrates how to use the ObjectOutputStream class for object serialization. An instance of the Student class with the ID 211 and the name "ravi" is created within the main() method. Next, bytes are written to a file called "f.txt" using a FileOutputStream called fout that has been generated. The Student object s1 is then serialised and written to the file by creating an ObjectOutputStream with the name out and wrapping fout.
To relieve related system resources, the streams are terminated using close() after making sure that any buffered data has been sent to the file using flush(). The software prints "success" to the console if it is successful

Example of Java Deserialization

The reverse operation of serialization is called deserialization, where the byte stream is converted into an object.

for deserialization, we call the **readObject()** method of the ObjectInputStream class.

Deserialization is the process of reconstructing the object from the serialized state. It is the reverse operation of serialization. Let's see an example where we are reading the data from a deserialized object.
Deserialization is the process of reconstructing the object from the serialized state. It is the reverse operation of serialization. Let's see an example where we are reading the data from a deserialized object.

Depersist.java

```
import java.io.*;
class Depersist{
 public static void main(String args[]){
 try{
 //Creating stream to read the object
 ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt"));
 Student s=(Student)in.readObject();
 //printing the data of the serialized object
 System.out.println(s.id+" "+s.name);
 //closing the stream
 in.close();
 }catch(Exception e){System.out.println(e);}
}
}
```

Output:

```
211 ravi
```

Explanation

This Java code excerpt illustrates the use of the ObjectInputStream class for object deserialisation. A FileInputStream that reads data from the file "f.txt" is wrapped in an ObjectInputStream called in inside the main procedure. The serialized object that was previously written to the file is read using this stream. After that, the object is deserialised and cast to the Student class using the readObject() function.
The deserialised Student object's ID and name are then printed to the console along with other associated data. In order to free up related system resources, the stream is finally closed using the close() method. During deserialization, any exceptions are caught and their error messages are shown.

The reverse operation of serialization is called deserialization, where the byte stream is converted into an object. The serialization and deserialization process is platform-independent. It means we can serialize an object on one platform and deserialize it on a different platform.

==
Q5:
Explain transient and static variables with an example program ?

Transient variable in Java

A **transient** variable is a special type of variable which we create by using the **transient** keyword. It is a special type of variable which have a non-serialized value at the time of serialization. A variable that is initialized by its default value during de-serialization is known as a transient variable.

A transient variable plays an important role in preventing an object from being serialized. We can make any variable transient by using the **transient** keyword.

The transient keyword is mainly used at the time of serializing an object

In order to exclude the **instance variables** from the serialization process, we use the **transient** keyword.

Example of Java Transient Keyword

Let's take an example, we have declared a class as ***Student***, it has three data members ***id, name*** and ***age***. If you serialize the object, all the values will be serialized but we don't want to serialize one value, e.g. ***age*** then we can declare the age data member as transient.
In this example, we have created two classes ***Student*** and ***PersistExample***. The ***age*** data member of the ***Student*** class is declared as transient, its value will not be serialized.
If you deserialize the object, you will get the default value for transient variable.
Let's create a class with transient variable.
PersistExample.java

```java
import java.io.*;
public class Student implements Serializable{
 int id;
 String name;
 transient int age;//Now it will not be serialized
 public Student(int id, String name,int age) {
  this.id = id;
  this.name = name;
  this.age=age;
 }
}
class PersistExample{
 public static void main(String args[])throws Exception{
  Student s1 =new Student(211,"ravi",22);//creating object
  //writing object into file
  FileOutputStream f=new FileOutputStream("f.txt");
  ObjectOutputStream out=new ObjectOutputStream(f);
  out.writeObject(s1);
  out.flush();
  out.close();
  f.close();
  System.out.println("success");
 }
}
```

Output:

```
success
```

Now write the code for deserialization.
DePersist.java

```java
import java.io.*;
class DePersist{
 public static void main(String args[])throws Exception{
  ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt"));
  Student s=(Student)in.readObject();
  System.out.println(s.id+" "+s.name+" "+s.age);
  in.close();
 }
}
```

Output:

```
211 ravi 0
```

From <https://www.tpointtech.com/transient-keyword>

3) Static variable

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

Example Static variable

Example

```java
class Student{
   //static variable
  static int age;
}
public class Main{
  public static void main(String args[]){
     Student s1 = new Student();
     Student s2 = new Student();
     s1.age = 24;
     s2.age = 21;
     Student.age = 23;
     System.out.println("S1\'s age is: " + s1.age);
     System.out.println("S2\'s age is: " + s2.age);
  }
}
```

Compile and Run

Output:

```
S1's age is: 23
S2's age is: 23
```

From <https://www.tpointtech.com/java-variables>

==

Java Console Class

The Java Console class is be used to get input from console. It provides methods to read texts and passwords.

If you read password using Console class, it will not be displayed to the user.

The java.io.Console class is attached with system console internally. The Console class is introduced since 1.5.

Let's see a simple example to read text from console.

```java
String text=System.console().readLine();
System.out.println("Text is: "+text);
```

From <https://www.tpointtech.com/java-console-class>

Java Console class declaration

Let's see the declaration for Java.io.Console class:

```java
public final class Console extends Object implements Flushable
```

From <https://www.tpointtech.com/java-console-class>

Java Console class methods

Method	Description
Reader reader()	It is used to retrieve the reader object associated with the console
String readLine()	It is used to read a single line of text from the console.
String readLine(String fmt, Object... args)	It provides a formatted prompt then reads the single line of text from the console.
char[] readPassword()	It is used to read password that is not being displayed on the console.
char[] readPassword(String fmt, Object... args)	It provides a formatted prompt then reads the password that is not being displayed on the console.
Console format(String fmt, Object... args)	It is used to write a formatted string to the console output stream.
Console printf(String format, Object... args)	It is used to write a string to the console output stream.

PrintWriter writer()	It is used to retrieve the PrintWriter object associated with the console.
void flush()	It is used to flushes the console.

How to get the object of Console

System class provides a static method console() that returns the singleton instance of Console class.

```java
public static Console console(){}
```

Let's see the code to get the instance of Console class.

```java
Console c=System.console();
```

Java Console Example

```java
import java.io.Console;
class ReadStringTest{
public static void main(String args[]){
Console c=System.console();
System.out.println("Enter your name: ");
String n=c.readLine();
System.out.println("Welcome "+n);
}
}
```

Output

```
Enter your name: Nakul Jain
Welcome Nakul Jain
```

Java Console Example to read password

```java
import java.io.Console;
class ReadPasswordTest{
public static void main(String args[]){
Console c=System.console();
System.out.println("Enter password: ");
char[] ch=c.readPassword();
String pass=String.valueOf(ch);//converting char array into string
System.out.println("Password is: "+pass);
}
}
```

Output

```
Enter password:
Password is: 123
```

From <https://www.tpointtech.com/java-console-class>