

**(Established under the Presidency University Act, 2013 of the Karnataka Act 41 of 2013)**

## CSA2022 – Advanced JAVA Programming LAB SHEET - 8

**Module 2 – Input Output Operations in Java**

**Operations with Channel and Buffer**

Buffers work with the channel. Channels are the tube through which data is transferred and buffers are the source and target of those data transfers. In the case of a write, data we want to write is placed in a buffer, which is passed to a channel then the channel reads that data from the buffer and writes it into the file.

**Q4**: Write a Java program to perform read and write operation with a data file using FileChannel and

ByteBuffer.

**+++++++++++++++++++++++++++**

**Aim:** The aim of this program is to demonstrate read and write operations with a data file using FileChannel and ByteBuffer in Java. It showcases how data is written to a file and then read back from it using these classes.

**Algorithm:**

**Writing Data (write method):**

1. Create a method **write** that takes a filename and data to be written as parameters.
2. Open a RandomAccessFile with the provided filename in read-write mode.
3. Obtain a FileChannel from the RandomAccessFile.
4. Create a ByteBuffer with the capacity to hold the data length.
5. Loop through each character in the data and put its byte representation into the ByteBuffer.
6. Rewind the ByteBuffer to set its position to zero.
7. Write the data from the ByteBuffer to the channel.
8. Close the channel and the RandomAccessFile.

**Reading Data (read method):**

1. Create a method **read** that takes a filename as a parameter.
2. Open a RandomAccessFile with the provided filename in read-write mode.
3. Obtain a FileChannel from the RandomAccessFile.
4. Create a ByteBuffer with a capacity of 1024 to read data into.
5. Read data from the channel into the ByteBuffer and store the number of bytes read.
6. Loop through the bytes read from the ByteBuffer, converting them to characters and printing them.
7. Close the channel and the RandomAccessFile.

**Program Explanation:**

* The program defines two methods, **write** and **read**, for writing and reading data to/from a file, respectively.
* In the **write** method, a RandomAccessFile is opened in read-write mode, and a FileChannel is obtained from it. Data is written to the file using a ByteBuffer.
* In the **read** method, the program opens the same file and reads data into a ByteBuffer, then converts and prints the characters from the ByteBuffer.
* The main function demonstrates the usage of these methods by writing the string "iphone 6 50000" to a file named "tablet.store" and then reading and printing the content.

Top of Form

**Program:**

**import** java.io.\*;

**import** java.nio.\*;

**import** java.nio.channels.FileChannel;

**public** **class** FileChannelDemo {

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

String data="iphone 6 50000";

*write*("tablet.store", data);

*read*("tablet.store");

}

**public** **static** **void** write(String filename, String data) {

**try** {

RandomAccessFile store = **new** RandomAccessFile(filename, "rw");

FileChannel channel = store.getChannel();

ByteBuffer buffer = ByteBuffe r.*allocate*(data.length());

**for** (**int** i = 0; i < data.length(); i++) {

buffer.put((**byte**) (data.charAt(i)));

}

// Rewinds buffer, the position is set to zero

buffer.rewind();

channel.write(buffer);

channel.close();

store.close();

}**catch**(Exception e) {}

}

**public** **static** **void** read(String filename) {

**try** {

RandomAccessFile store = **new** RandomAccessFile(filename, "rw");

FileChannel channel = store.getChannel();

ByteBuffer buffer = ByteBuffer.*allocate*(1024);

**int** numOfBytesRead = channel.read(buffer);

System.***out***.println("Number of bytes read : " + numOfBytesRead);

**for**(**int** i=0;i<numOfBytesRead;i++) {

**char** c=(**char**)buffer.get(i);

System.***out***.print(c);

}

channel.close();

store.close();

}**catch**(Exception e) {}

}

}

**Output:**

Number of bytes read : 14

iphone 6 50000

**Operations with Serialization**

**Q5**: Write a java program to serialize and deserialize a student object under the file name file named student.ser.

**Aim:** The aim of this program is to demonstrate Java object serialization. It involves creating a **Student** object, serializing it, and saving it to a file.

**Algorithm:**

1. Import the necessary libraries, including **java.io.\*** and **java.io.Serializable**.
2. Create a **Student** class that implements the **Serializable** interface, which allows the object of this class to be serialized.
3. Define attributes for the **Student** class, such as **name**, **regdno**, and **cgpa**.
4. Create a **SerializeDemo** class with the **main** method.
5. Inside the **main** method: a. Create a **Student** object, **s**, and set its attributes.

b. Create a **FileOutputStream** to write data to a file. In this case, the file is named "student.ser" and is located in the "5BCA2" folder.

c. Create an **ObjectOutputStream** to write the **Student** object to the file.

d. Use the **writeObject** method to serialize and write the **Student** object to the file.

e. Close the **ObjectOutputStream** and **FileOutputStream**.

f. Print a message indicating that the serialized data is saved in the "5BCA2\student.ser" file.

**Program Explanation:**

* The program begins by defining a **Student** class that implements the **Serializable** interface. This allows instances of the **Student** class to be converted into a byte stream and saved to a file.
* In the **SerializeDemo** class, the **main** method demonstrates object serialization:
  + A **Student** object, **s**, is created and populated with values for **name**, **regdno**, and **cgpa**.
  + A **FileOutputStream** is created to specify the file where the serialized data will be saved. In this case, it's "student.ser" within the "5BCA2" folder.
  + An **ObjectOutputStream** is created to write the object to the file.
  + The **writeObject** method is used to serialize and write the **Student** object to the file.
  + Both the **ObjectOutputStream** and **FileOutputStream** are closed.
  + A message is printed to indicate that the serialized data is saved in the "5BCA2\student.ser" file.

This program demonstrates how to use Java's object serialization to save an object's state to a file, allowing it to be deserialized and reconstructed later. In this case, a **Student** object is serialized and saved to a file named "student.ser."

Top of Form

**Program:**

**import** java.io.\*;

**import** java.io.Serializable;

**class** Student **implements** Serializable {

String name;

String regdno;

**double** cgpa;

}

**public** **class** SerializeDemo {

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

Student s = **new** Student();

s.name = "sanjay";

s.regdno = "20213cse0123";

s.cgpa = 7.5;

FileOutputStream fileOut =**new** FileOutputStream("5BCA2\\student.ser");

ObjectOutputStream out = **new** ObjectOutputStream(fileOut);

out.writeObject(s);

out.close();

fileOut.close();

System.***out***.printf("Serialized data is saved in 5BCA2\\student.ser");

}

}

**Output:**

Serialized data is saved in 5BCA2\\student.ser

**Aim:** The aim of this program is to demonstrate Java object deserialization. It involves reading a serialized **Student** object from a file, reconstructing it, and displaying its attributes.

**Algorithm:**

1. Import the necessary libraries, including **java.io.\***.
2. Create a **DeserializeDemo** class with the **main** method.
3. Inside the **main** method: a. Initialize a **Student** object, **s**, to **null**. This object will be used to hold the deserialized data.

b. Create a **FileInputStream** to read data from the serialized file. The file "student.ser" is located in the "5BCA2" folder.

c. Create an **ObjectInputStream** to read the serialized **Student** object from the file.

d. Use the **readObject** method to deserialize and read the **Student** object from the file. Cast the result to a **Student** object.

e. Close the **ObjectInputStream** and **FileInputStream**.

f. Print the deserialized student's details, including their name, registration number, and CGPA.

**Program Explanation:**

* The program begins by creating a **DeserializeDemo** class with a **main** method.
* Inside the **main** method, the following steps are executed:
  + A **Student** object, **s**, is initialized to **null**. This object will be used to hold the deserialized data.
  + A **FileInputStream** is created to read data from the serialized file. The file "student.ser" is located in the "5BCA2" folder.
  + An **ObjectInputStream** is created to read the serialized **Student** object from the file.
  + The **readObject** method is used to deserialize and read the **Student** object from the file. The result is cast to a **Student** object.
  + Both the **ObjectInputStream** and **FileInputStream** are closed.
  + The program prints the details of the deserialized student, including their name, registration number, and CGPA.

This program demonstrates how to use Java's object deserialization to read and reconstruct a serialized object from a file. In this case, a **Student** object is deserialized from the "student.ser" file, and its attributes are printed to the console.

Top of Form

**Program:**

**import** java .io.\*;

**public** **class** DeserializeDemo {

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

Student s = **null**;

FileInputStream fileIn = **new** FileInputStream("5BCA2\\student.ser");

ObjectInputStream in = **new** ObjectInputStream(fileIn);

s = (Student) in.readObject();

in.close();

fileIn.close();

System.***out***.println("Deserialized Student...");

System.***out***.println("Name: " + s.name);

System.***out***.println("Regd no: " + s.regdno);

System.***out***.println("CGPA: " + s.cgpa);

}

}

**Output:**

Deserialized Student...

Name: sanjay

Regd no: 20213cse0123

CGPA: 7.5