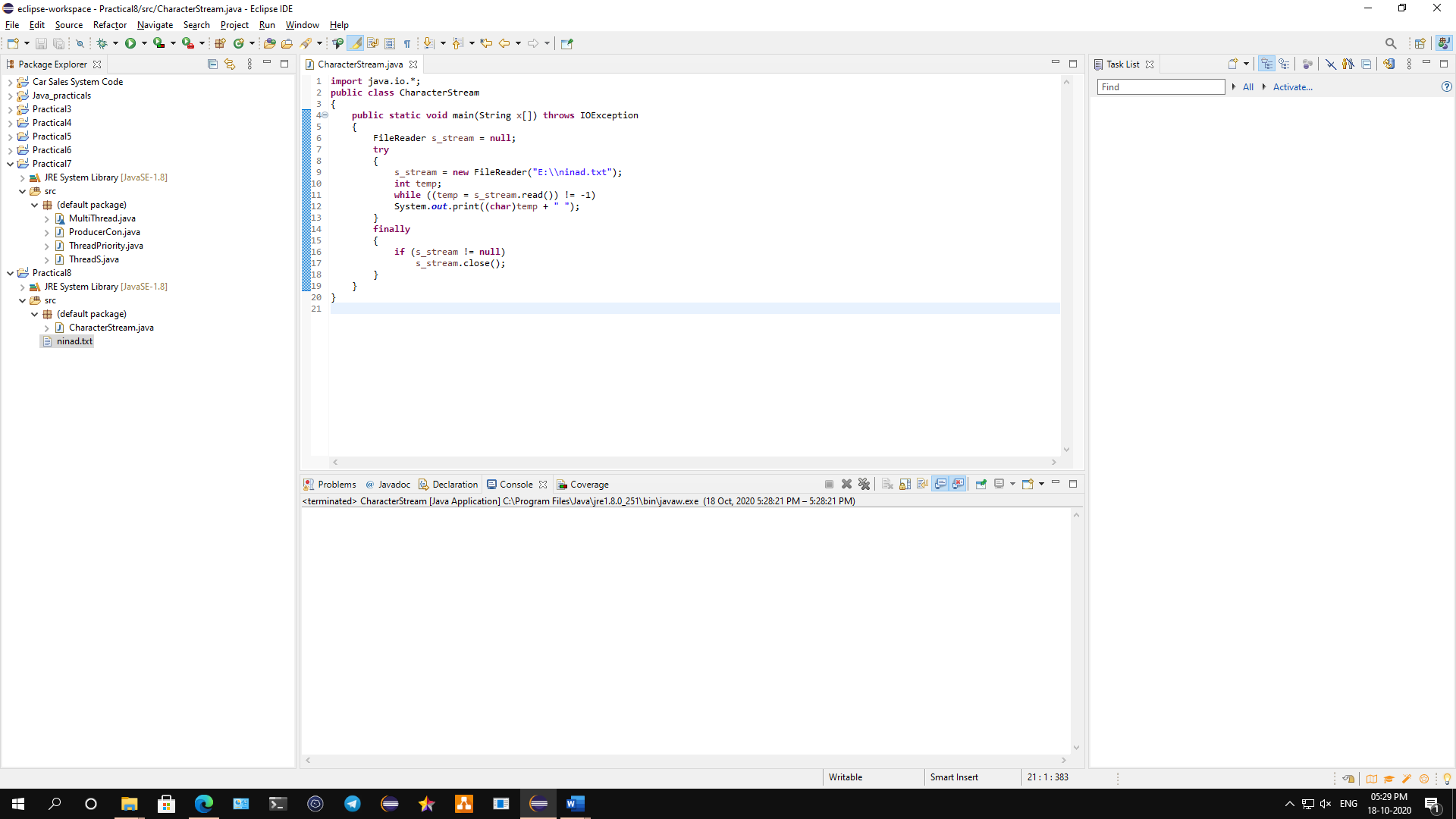
**Practical No 8**

**A) To implement program using console-based IO using character and byte stream.**

**Aim: Write a program to implement console-based IO using character and byte stream.**

**Description:**

A stream is a method to sequentially access a file. I/O Stream means an input source or output destination representing different types of sources e.g. disk files. The java.io package provides classes that allow you to convert between Unicode character streams and byte streams of non-Unicode text. In Java, characters are stored using Unicode conventions. Character stream automatically allows us to read/write data character by character. For example, FileReader and FileWriter are character streams used to read from source and write to destination. Byte streams process data byte by byte (8 bits). For example, FileInputStream is used to read from source and FileOutputStream to write to the destination. In the given program we have use CharacterStream and public class and then we defined main method with throws keyword, then we defined FileReader function to read the file which is defined in txt format with its dir. Then the finally keyword will display the output of text file. In another program we use ByteStream class followed by public class as fileoutputstream. We defined the main function follow by throws keyword and then we use try keyword to handle the exception, we also defined the file which is in txt format. The output will be displayed in byte format.



**Conclusion: We have written a program to implement console-based IO using character and byte stream.**

**Code:**

**Character Stream:**

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

**public** **class** CharacterStream

{

**public** **static** **void** main(String x[]) **throws** IOException

{

FileReader s\_stream = **null**;

**try**

{

s\_stream = **new** FileReader("E:\\ninad.txt");

**int** temp;

**while** ((temp = s\_stream.read()) != -1)

System.***out***.print((**char**)temp + " ");

}

**finally**

{

**if** (s\_stream != **null**)

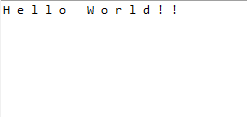
s\_stream.close();

}

}

}

**Output:**



**Byte Stream:**

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

**public** **class** ByteStream {

**private** **static** FileOutputStream *t\_stream*;

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

FileInputStream s\_stream = **null**;

*setT\_stream*(**null**);

**try** {

s\_stream = **new** FileInputStream("E:\\ninad.txt");

**int** temp;

**while** ((temp = s\_stream.read()) != -1)

System.***out***.print((**byte**)temp+ " ");

}

**finally** {

**if** (s\_stream != **null**)

s\_stream.close();

}

}

**public** **static** FileOutputStream getT\_stream() {

**return** *t\_stream*;

}

**public** **static** **void** setT\_stream(FileOutputStream t\_stream) {

ByteStream.*t\_stream* = t\_stream;

}

}

**Output:**

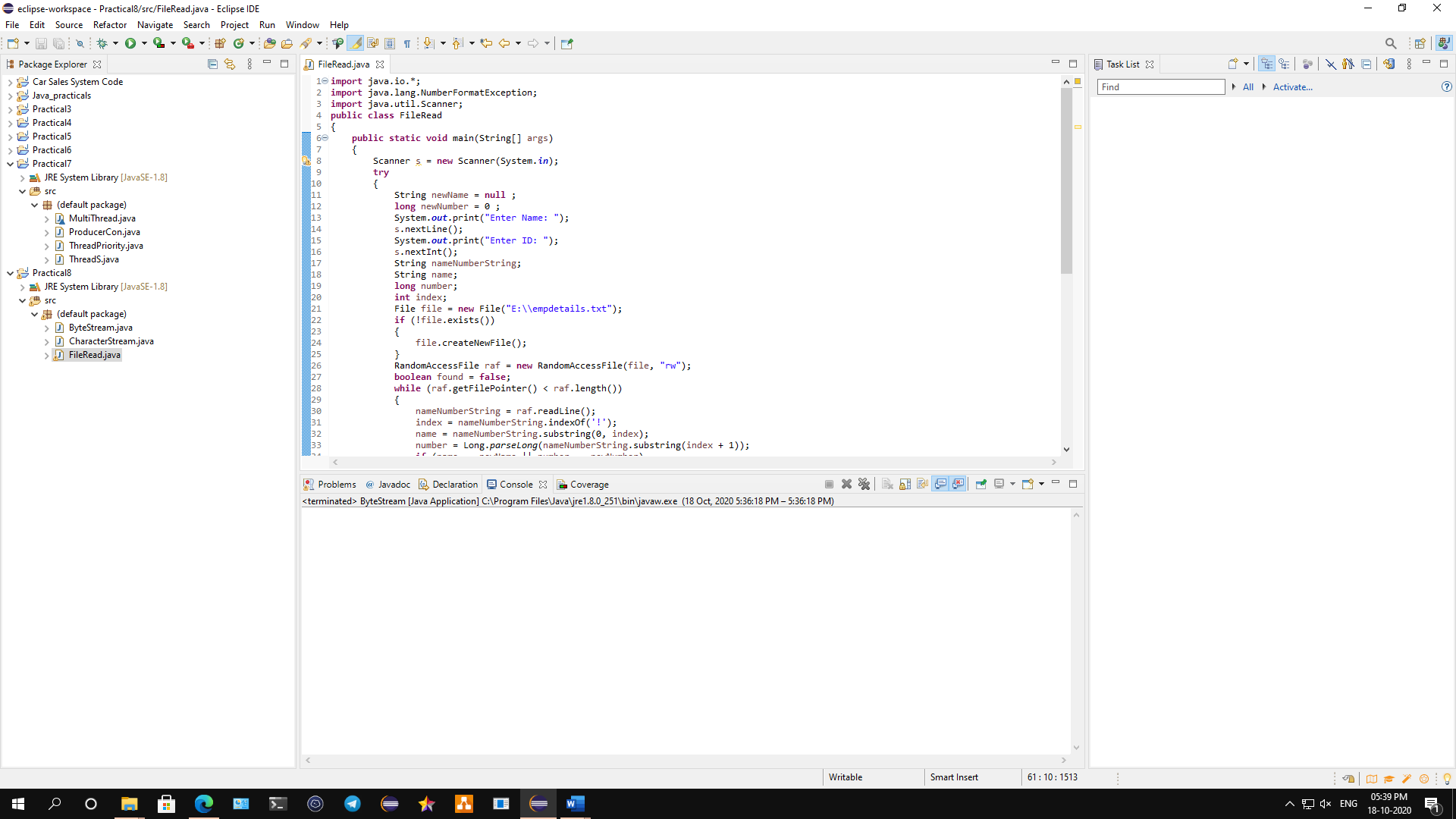


**B) To implement a program to store the detail of employee in file using file reader and writer class/File Input Stream and Output stream.**

**Aim: Write a java program to implement a program to store the detail of employee in file using file reader and writer class/File Input Stream and Output stream.**

**Description:**

I/O Stream means an input source or output destination representing different types of sources e.g. disk files. The java.io package provides classes that allow you to convert between Unicode character streams and byte streams of non-Unicode text. In Java, characters are stored using Unicode conventions. Character stream automatically allows us to read/write data character by character. For example, FileReader and FileWriter are character streams used to read from source and write to destination. Byte streams process data byte by byte (8 bits). For example, FileInputStream is used to read from source and FileOutputStream to write to the destination. In this program we have imported some java class such as io, lang.numberformatexception and scanner class. Then we defined fileread class followed by the main function, then we use scanner class followed by try key block which is used to handle then exception. We define long function for new number and then the file function is used to read the file. The output will be displayed showing employee details.



**Conclusion: We have implemented a program to store the detail of employee in file using file reader and writer class/File Input Stream and Output stream.**

**Code:**

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

**import** java.lang.NumberFormatException;

**import** java.util.Scanner;

**public** **class** FileRead

{

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

{

Scanner s = **new** Scanner(System.***in***);

**try**

{

String newName = **null** ;

**long** newNumber = 0 ;

System.***out***.print("Enter Name: ");

s.nextLine();

System.***out***.print("Enter ID: ");

s.nextInt();

String nameNumberString;

String name;

**long** number;

**int** index;

File file = **new** File("E:\\empdetails.txt");

**if** (!file.exists())

{

file.createNewFile();

}

RandomAccessFile raf = **new** RandomAccessFile(file, "rw");

**boolean** found = **false**;

**while** (raf.getFilePointer() < raf.length())

{

nameNumberString = raf.readLine();

index = nameNumberString.indexOf('!');

name = nameNumberString.substring(0, index);

number = Long.*parseLong*(nameNumberString.substring(index + 1));

**if** (name == newName || number == newNumber)

{

found = **true**;

**break**;

}

}

**if** (found == **false**)

{

nameNumberString = newName+"!"+ String.*valueOf*(newNumber);

raf.writeBytes(nameNumberString);

raf.writeBytes(System.*lineSeparator*());

System.***out***.println(" Employee added. ");

raf.close();

}

**else**

{

raf.close();

System.***out***.println(" The entered employee" + " already exists. ");

}

}

**catch** (IOException ioe)

{

System.***out***.println(ioe);

}

**catch** (NumberFormatException nef)

{

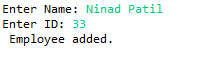
System.***out***.println(nef);

}

}

}

**Output:**

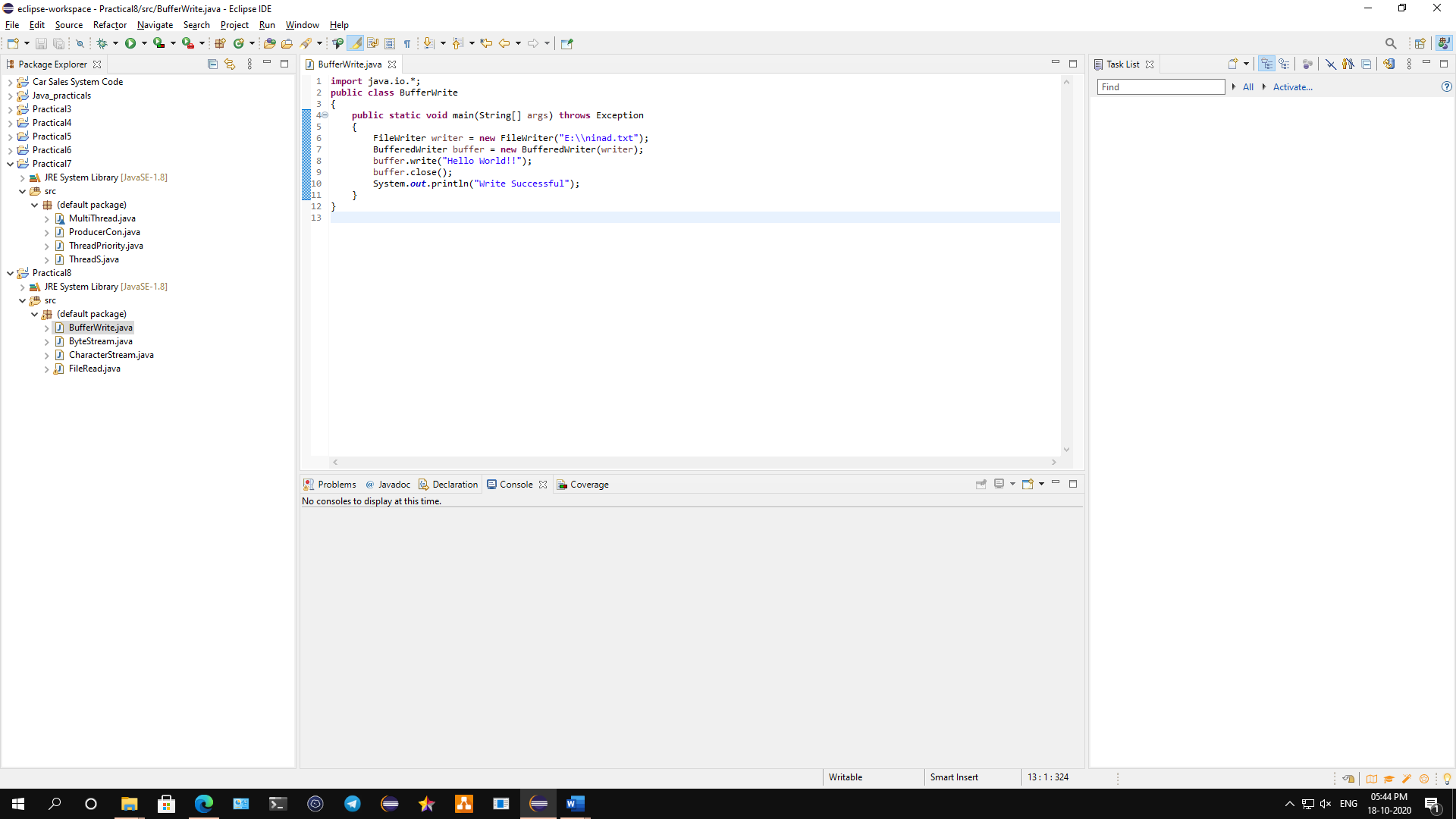


**C) To implement a program on buffered Reader/Writer.**

**Aim: Write a java program to implement buffered reader/writer.**

**Description:**

Java BufferedWriter class is used to provide buffering for Writer instances. It makes the performance fast. It inherits [Writer](https://www.javatpoint.com/java-writer-class) class. The buffering characters are used for providing the efficient writing of single [arrays](https://www.javatpoint.com/array-in-java), characters, and [strings](https://www.javatpoint.com/java-string). Java BufferedReader class is used to read the text from a character-based input stream. It can be used to read data line by line by readLine() method. It makes the performance fast. It inherits [Reader](https://www.javatpoint.com/java-reader-class) [class](https://www.javatpoint.com/object-and-class-in-java). In the first program we defined the bufferwrite class followed by the main function and the throws keyword which throws an exception. Then we use file write function to write the character in the output. In the second program we defined the bufferread class followed by the main functions which also define the throws keyword to throws an exception. Then the file read function is used to read the file and it will print the output i which will read the character in the files and display it.



**Conclusion: We have implemented a program to read/write buffer.**

**Code:**

**Buffered Write:**

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

**public** **class** BufferWrite

{

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

{

FileWriter writer = **new** FileWriter("E:\\ninad.txt");

BufferedWriter buffer = **new** BufferedWriter(writer);

buffer.write("Hello World!!");

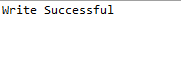
buffer.close();

System.***out***.println("Write Successful");

}

}

**Output:**



**Buffered Read:**

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

**public** **class** BufferRead

{

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

{

FileReader fr=**new** FileReader("E:\\ninad.txt");

BufferedReader br=**new** BufferedReader(fr);

**int** i;

**while**((i=br.read())!=-1)

{

System.***out***.print((**char**)i);

}

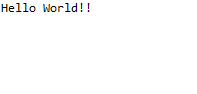
br.close();

fr.close();

}

}

**Output:**

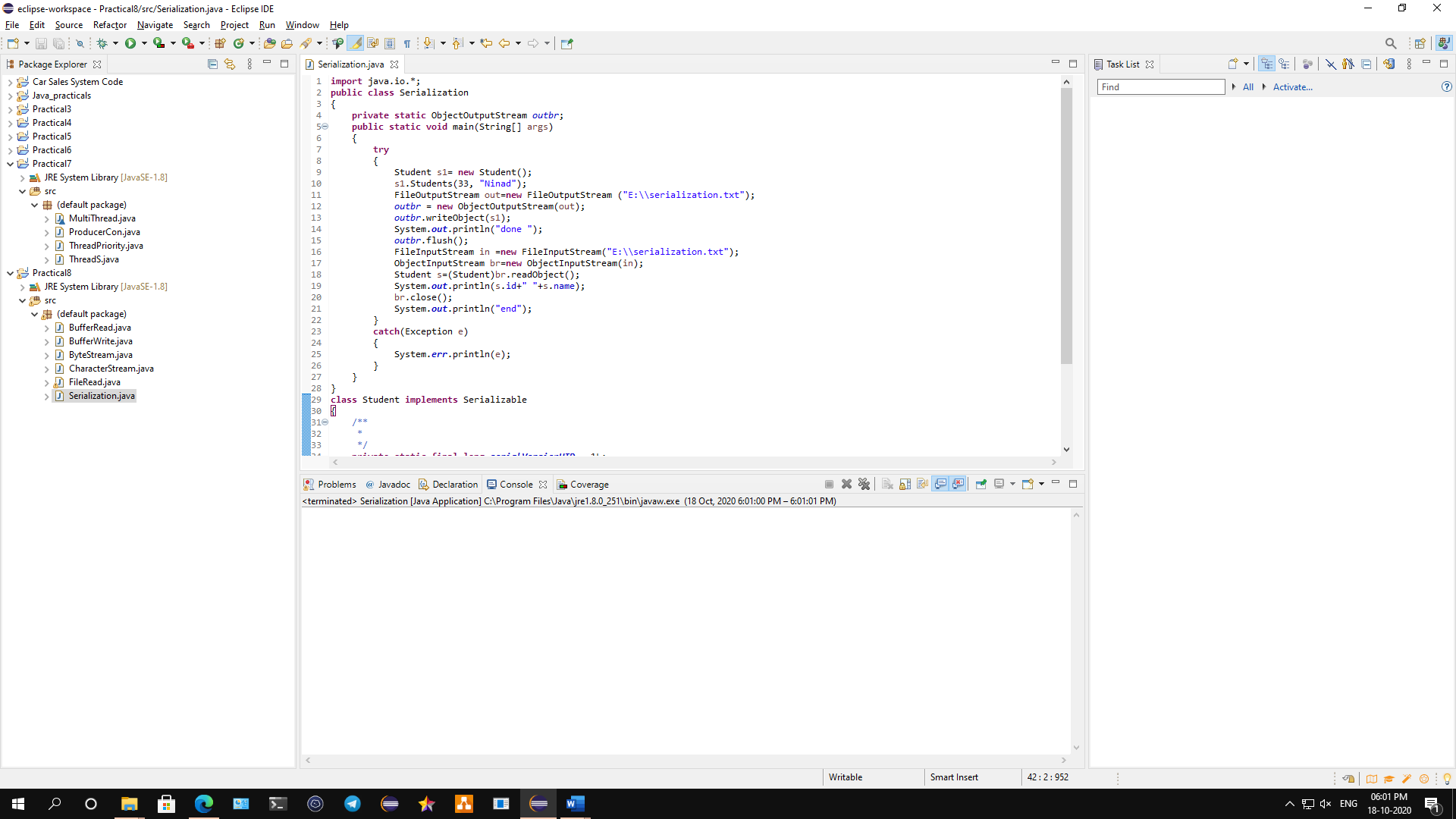


**D) To demonstrate Object serialization and De serialization.**

**Aim: Write a java program to demonstrate Object serialization and De serialization.**

**Description:**

**Serialization in Java** is a mechanism of writing the state of an object into a byte-stream. It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies. The reverse operation of serialization is called deserialization where byte-stream is converted into an object. The serialization and deserialization process are platform-independent, it means you can serialize an object in a platform and deserialize in different platform. For serializing the object, we call the **writeObject()** method ObjectOutputStream, and for deserialization we call the **readObject()** method of ObjectInputStream class. In the given program we defined the serialization class followed by the object outputstream and main function. Then we defined try keyword to handle the exception, then we define the output stream function to read the output and save it in a file name serialization.txt once it save it will proceed to inputstream function which helps to read and display the output of the content which is in serialization.txt file.



**Conclusion: We have demonstrated a program on Object serialization and De serialization.**

**Code:**

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

**public** **class** Serialization

{

**private** **static** ObjectOutputStream *outbr*;

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

{

**try**

{

Student s1= **new** Student();

s1.Students(33, "Ninad");

FileOutputStream out=**new** FileOutputStream ("E:\\serialization.txt");

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

*outbr*.writeObject(s1);

System.***out***.println("done ");

*outbr*.flush();

FileInputStream in =**new** FileInputStream("E:\\serialization.txt");

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

Student s=(Student)br.readObject();

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

br.close();

System.***out***.println("end");

}

**catch**(Exception e)

{

System.***err***.println(e);

}

}

}

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

{

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**int** id;

String name;

**public** **void** Students(**int** id, String name)

{

**this**.id = id;

**this**.name = name;

}

}

**Output:**

