Index

[Exercise 1: List the contents of a directory (Recursively) 5](#_Toc322469490)

[Task 1 - Create Directory Project 5](#_Toc322469491)

[Task 2 - Create ListDirectoryRecusive class 5](#_Toc322469492)

[Task 3 - Execute your program 6](#_Toc322469493)

[Exercise 2: FileOutputStream and FileInputStream 7](#_Toc322469494)

[Task 1 - Using notepad to Create the a file “source.txt” 7](#_Toc322469495)

[Task 2 - Use FileInputStream and FileOutputStream to copy file 7](#_Toc322469496)

[Task 3 - Execute and see the result (open file destination.txt) 7](#_Toc322469497)

[Exercise 3: DataInputStream and DataOutputStream 8](#_Toc322469498)

[Task 1 - Crate a Project named WriteInt 8](#_Toc322469499)

[Task 2 - Write code in main() 8](#_Toc322469500)

[Task 3 - Write code to read that “int number” 8](#_Toc322469501)

[Task 4 - Execute and see the result 8](#_Toc322469502)

[Exercise 4: InputStreamReader and OutputStreamWriter 8](#_Toc322469503)

[Task 1 - Create UnicodeFileIO Class 8](#_Toc322469504)

[Task 2 - Execute and see the result 9](#_Toc322469505)

[Exercise 5: BufferedReader and BufferedWriter 10](#_Toc322469506)

[Exercise 6: Copy File 10](#_Toc322469507)

[Task 1 - Create Project named CopyFile 10](#_Toc322469508)

[Task 2 - Copying a file without Buffering 10](#_Toc322469509)

[Task 3 - Execute and see the result 11](#_Toc322469510)

[Task 4 - Copying a file with a Programmer-Managed Buffer 12](#_Toc322469511)

[Task 5 - Execute and see the result 13](#_Toc322469512)

[Task 6 - Copying a file with Buffered Streams 13](#_Toc322469513)

[Task 7 - Execute and see the result 14](#_Toc322469514)

[Exercise 7: Best Copier 14](#_Toc322469515)

[Exercise 8: PipedReader and PipedWriter 15](#_Toc322469516)

[Task 1 - Create pReader extended Thread 15](#_Toc322469517)

[Task 2 - Create pWriter extended Thread 15](#_Toc322469518)

[Task 3 - Write test Program 16](#_Toc322469519)

[Task 4 - Execute and see the result 16](#_Toc322469520)

[Exercise 9: Serializable - ObjectOutputStream 17](#_Toc322469521)

[Task 1 - Create SerializedObject class 17](#_Toc322469522)

[Task 2 - Create a program to write and read Serializable object 17](#_Toc322469523)

[Task 3 - Execute your program 19](#_Toc322469524)

[Exercise 10: Count the number of lines 19](#_Toc322469525)

1. List the contents of a directory (Recursively)

## Create ListDirectoryRecusive class

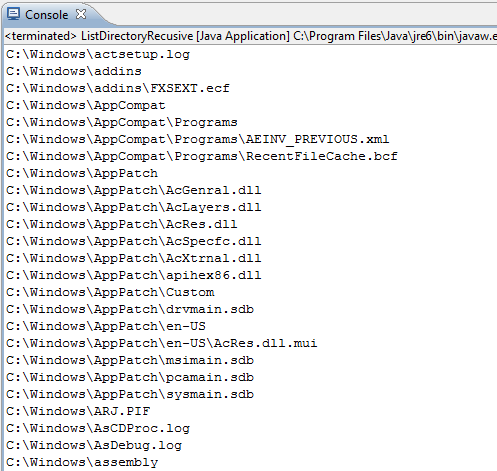
### Create listRecursive() method

* 1. **public** **static** **void** listRecursive(File dir) {
  2. **if** (dir.isDirectory()) {
  3. File[] items = dir.listFiles();
  4. **if** (items != **null**) {
  5. **for** (File item : items) {
  6. System.*out*.println(item.getAbsoluteFile());
  7. **if** (item.isDirectory())
  8. *listRecursive*(item);
  9. }
  10. }
  11. }
  12. }

### Write main() method to list the contents of a directory

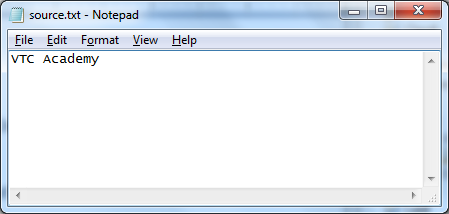
* 1. **public** **static** **void** main(String[] args) {
  2. File dir = **new** File("C:\\Windows");
  3. *listRecursive*(dir);
  4. }

## Execute your program



1. FileOutputStream and FileInputStream

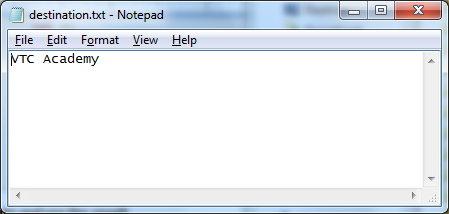
## Using notepad to Create the a file “source.txt”



## Use FileInputStream and FileOutputStream to copy file

* 1. FileInputStream in = **new** FileInputStream("D:\\source.txt");
  2. FileOutputStream out = **new** FileOutputStream("D:\\destination.txt");
  3. **int** c;
  4. **while** ((c = in.read()) != -1) {
  5. out.write(c);
  6. }
  7. **if** (in != **null**) {
  8. in.close();
  9. }
  10. **if** (out != **null**) {
  11. out.close();
  12. }

## Execute and see the result (open file destination.txt)



1. DataInputStream and DataOutputStream

## Crate a Project named WriteInt

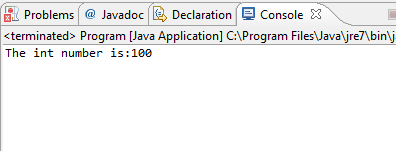
## Write code in main()

* 1. try {
  2. FileOutputStream fos = new FileOutputStream("D:\\WriteInt.txt");
  3. DataOutputStream dos = new DataOutputStream(fos);
  4. int i = 100;
  5. dos.writeInt(i);
  6. dos.close();
  7. } catch (IOException e) {
  8. System.*out*.println("IOException : " + e);
  9. }

## Write code to read that “int number”

* 1. //Your code

## Execute and see the result



1. InputStreamReader and OutputStreamWriter

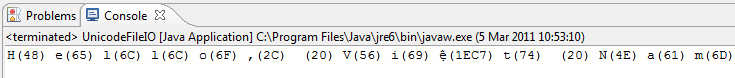
## Create UnicodeFileIO Class

### Write code in UnicodeFileIO.java

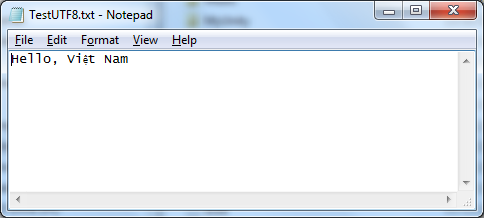
* 1. **public** **static** **void** main(String[] args) {
  2. **try** {
  3. FileOutputStream fos = **new** FileOutputStream("C:\\TestUTF8.txt");
  4. Writer out = **new** OutputStreamWriter(fos, "UTF8"); // Output file
  5. // UTF-8 encoded
  6. out.write("Hello, Việt Nam");
  7. out.close();
  8. // Character-based (or text-based) input
  9. FileInputStream fis = **new** FileInputStream("C:\\TestUTF8.txt");
  10. Reader in = **new** InputStreamReader(fis, "UTF8");
  11. **int** charRead;
  12. **int** charCount = 0;
  13. **while** ((charRead = in.read()) != -1) {
  14. charCount++;
  15. System.*out*.printf("%c(%02X) ", (**char**) charRead, charRead);
  16. }
  17. in.close();
  18. } **catch** (FileNotFoundException ex) {
  19. ex.printStackTrace();
  20. } **catch** (UnsupportedEncodingException ex) {
  21. ex.printStackTrace();
  22. } **catch** (IOException ex) {
  23. ex.printStackTrace();
  24. }
  25. }

## Execute and see the result

### The output appears on the "Console" panel

* 1. 

### Open file TestUTF8.txt and see the content inside

* 1. 

1. BufferedReader and BufferedWriter

Edit code in Exercise 4 using **BufferedReader** and **BufferedWriter**

1. Copy File

In this exercise you will write a program to copy a file by using 2 methods

* Copying a file without Buffering
* Copying a file with a Programmer-Managed Buffer
* Copying a file with Buffered Streams

## Create Project named CopyFile

## Copying a file without Buffering

### Create **FileCopyNoBuffer** Class

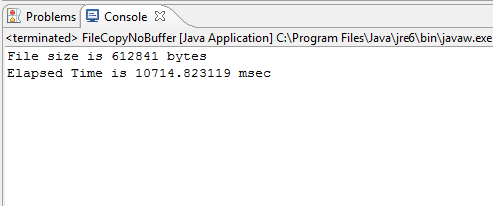
### Write code in **FileCopyNoBuffer.java**

* 1. **import** java.io.\*;
  2. **public** **class** FileCopyNoBuffer {
  3. **public** **static** **void** main(String[] args) {
  4. File fileIn;
  5. FileInputStream in = **null**;
  6. FileOutputStream out = **null**;
  7. **long** startTime, elapsedTime; // for speed benchmarking
  9. **try** {
  10. fileIn = **new** File("D:\\a.jpg");
  11. System.*out*.println("File size is " + fileIn.length() + " bytes");
  12. in = **new** FileInputStream(fileIn);
  13. out = **new** FileOutputStream("D:\\b.jpg");
  15. startTime = System.*nanoTime*();
  16. **int** byteRead;
  17. // Read a unsigned byte (0-255) and padded to 32-bit int
  18. **while** ((byteRead = in.read()) != -1) {
  19. // Write the least significant byte, drop the upper 3 bytes
  20. out.write(byteRead);
  21. }
  22. elapsedTime = System.*nanoTime*() - startTime;
  23. System.*out*.println("Elapsed Time is " + (elapsedTime / 1000000.0) + " msec");
  24. } **catch** (IOException ex) {
  25. ex.printStackTrace();
  26. } **finally** { // always close the streams
  27. **try** {
  28. **if** (in != **null**) in.close();
  29. **if** (out != **null**) out.close();
  30. } **catch** (IOException ex) { ex.printStackTrace(); }
  31. }
  32. }
  33. }

## Execute and see the result

### To run the program, right-click anywhere on the source file " FileCopyNoBuffer.java" (or from the "Run" menu) ⇒ Choose "Run As" ⇒ "Java Application".

### The output appears on the "Console" panel:

* 1. 

### Open copied file in your system and compare with the original file

## Copying a file with a Programmer-Managed Buffer

### Create **FileCopyUserBuffer** Class

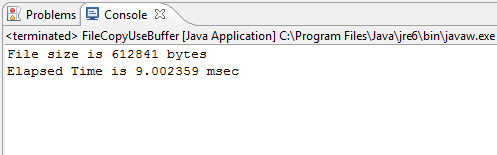
### Write code in **FileCopyUserBuffer.java**

* 1. **import** java.io.\*;
  2. **public** **class** FileCopyUserBuffer {
  3. **public** **static** **void** main(String[] args) {
  4. File fileIn;
  5. FileInputStream in = **null**;
  6. FileOutputStream out = **null**;
  7. **long** startTime, elapsedTime; // for speed benchmarking
  9. **try** {
  10. fileIn = **new** File("D:\\b.jpg");
  11. System.*out*.println("File size is " + fileIn.length() + " bytes");
  12. in = **new** FileInputStream(fileIn);
  13. out = **new** FileOutputStream("D:\\c.jpg");
  14. startTime = System.*nanoTime*();
  15. **byte**[] byteBuf = **new** **byte**[4096]; // 4K buffer
  16. **int** numBytesRead;
  17. **while** ((numBytesRead = in.read(byteBuf)) != -1) {
  18. out.write(byteBuf, 0, numBytesRead);
  19. }
  20. elapsedTime = System.*nanoTime*() - startTime;
  21. System.*out*.println("Elapsed Time is " + (elapsedTime / 1000000.0) + " msec");
  22. } **catch** (IOException ex) {
  23. ex.printStackTrace();
  24. } **finally** { // always close the streams
  25. **try** {
  26. **if** (in != **null**) in.close();
  27. **if** (out != **null**) out.close();
  28. } **catch** (IOException ex) { ex.printStackTrace(); }
  29. }
  30. }
  31. }

## Execute and see the result

### To run the program, right-click anywhere on the source file "FileCopyUserBuffer.java" (or from the "Run" menu) ⇒ Choose "Run As" ⇒ "Java Application".

### The output appears on the "Console" panel

* 1. 

### Open copied file in your system and compare with the original file

## Copying a file with Buffered Streams

### Create FileCopyBuffered Class

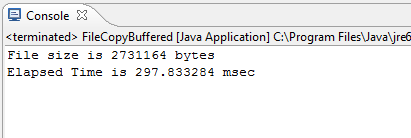
### Write code in FileCopyBuffered.java

* 1. **import** java.io.\*;
  2. **public** **class** FileCopyBuffered {
  3. **public** **static** **void** main(String[] args) {
  4. File fileIn;
  5. BufferedInputStream in = **null**;
  6. BufferedOutputStream out = **null**;
  7. **long** startTime, elapsedTime; // for speed benchmarking
  9. **try** {
  10. fileIn = **new** File("D:\\a.jpg");
  11. System.*out*.println("File size is " + fileIn.length() + " bytes");
  12. in = **new** BufferedInputStream(**new** FileInputStream(fileIn));
  13. out = **new** BufferedOutputStream(**new** FileOutputStream("D:\\e.jpg"));
  14. startTime = System.*nanoTime*();
  15. **int** byteRead;
  16. **while** ((byteRead = in.read()) != -1) {
  17. out.write(byteRead);
  18. }
  19. elapsedTime = System.*nanoTime*() - startTime;
  20. System.*out*.println("Elapsed Time is " + (elapsedTime / 1000000.0) + " msec");
  21. } **catch** (IOException ex) {
  22. ex.printStackTrace();
  23. } **finally** { // always close the streams
  24. **try** {
  25. **if** (in != **null**) in.close();
  26. **if** (out != **null**) out.close();
  27. } **catch** (IOException ex) { ex.printStackTrace(); }
  28. }
  29. }
  30. }

## Execute and see the result

### To run the program, right-click anywhere on the source file "FileCopyBuffered.java" (or from the "Run" menu) ⇒ Choose "Run As" ⇒ "Java Application".

### The output appears on the "Console" panel

* 1. 

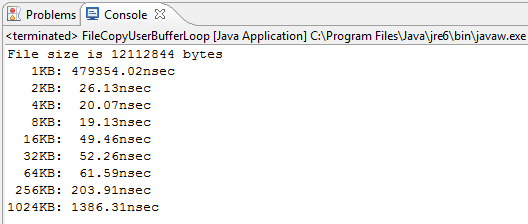
### Open copied file in your system and compare with the original file

1. Best Copier

Rewrite **FileCopyUserBuffer.java** to find the fastest method of copy by choosing different buffersize

* 1. int[] bufSizeKB = {1, 2, 4, 8, 16, 32, 64, 256, 1024}; // in KB
  2. for (int run = 0; run < bufSizeKB.length; run++)
  3. {
  4. bufSize = bufSizeKB[run] \* 1024;
  5. byte[] byteBuf = new byte[bufSize]; // 4K buffer
  6. //your code
  7. }

Output of your program will be:



1. PipedReader and PipedWriter

## Create pReader extended Thread

* 1. **import** java.io.IOException;
  2. **import** java.io.PipedReader;
  3. **public** **class** pReader **extends** Thread {
  4. PipedReader reader;
  5. pReader(PipedReader reader) {
  6. **this**.reader = reader;
  7. }
  8. **public** **void** run() {
  9. **try** {
  10. **while** (**true**) {
  11. *sleep*(1000);
  12. **if** (reader.ready())
  13. System.*out*.print((**char**) reader.read());
  14. }
  15. } **catch** (IOException e) {
  16. // **TODO** Auto-generated catch block
  17. e.printStackTrace();
  18. } **catch** (InterruptedException e) {
  19. e.printStackTrace();
  20. }
  21. }
  22. }

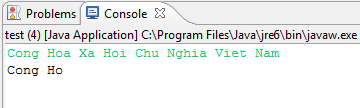
## Create pWriter extended Thread

* 1. **import** java.io.IOException;
  2. **import** java.io.PipedWriter;
  3. **import** java.util.Scanner;
  4. **public** **class** pWriter **extends** Thread {
  5. PipedWriter writer;
  6. pWriter(PipedWriter writer) {
  7. **this**.writer = writer;
  8. }
  9. **public** **void** run() {
  10. String str;
  11. **while** (**true**) {
  12. Scanner in = **new** Scanner(System.*in*);
  13. str = in.nextLine();
  14. **try** {
  15. writer.write(str);
  16. } **catch** (IOException e) {
  17. // **TODO** Auto-generated catch block
  18. e.printStackTrace();
  19. }
  20. }
  21. }
  22. }

## Write test Program

* 1. **import** java.io.PipedReader;
  2. **import** java.io.PipedWriter;
  3. **public** **class** test {
  4. **public** **static** **void** main(String args[]) **throws** Exception {
  5. // Create a new instance of a PipedReader object.
  6. PipedReader reader = **new** PipedReader();
  7. // Create a new instance of a PipedWriter object.
  8. PipedWriter writer = **new** PipedWriter();
  9. // Connect the PipedReader to a PipedWriter object.
  10. reader.connect(writer);
  11. pReader treader = **new** pReader(reader);
  12. pWriter twriter = **new** pWriter(writer);
  13. treader.start();
  14. twriter.start();
  15. }
  16. }

## Execute and see the result



1. Serializable - ObjectOutputStream

Object serialization is the process of representing a particular state of an object in a serialized bit-stream, so that the bit stream can be written out to an external device (such as a disk file or network).

In Java, object that requires to be serialized must implement java.io.Serializable or java.io.Externalizable interface. Serializable interface is an empty interface (or tagged interface) with nothing declared. Its purpose is simply to declare that particular object is serializable.

## Create SerializedObject class

### In the "Package Explorer" (left panel) ⇒ Right-click on your Project (or use the "File" menu) ⇒ New ⇒ Class.

### The "New Java Class" dialog pops up.

* In "Name" field, enter “SerializedObject ".
* In Interface field, press “Add” button,
* Enter Serializable and press Add
* Click "Finish".

### Write code in **SerializedObject.java**

* 1. **import** java.io.Serializable;
  2. **class** SerializedObject **implements** Serializable {
  3. **private** **int** number;
  5. **public** SerializedObject(**int** number) {
  6. **this**.number = number;
  7. }
  9. **public** **int** getNumber() {
  10. **return** number;
  11. }
  12. }

## Create a program to write and read Serializable object

### In the "Package Explorer" (left panel) ⇒ Right-click on your Project (or use the "File" menu) ⇒ New ⇒ Class.

### The "New Java Class" dialog pops up.

* In "Name" field, enter “ObjectSerializationTest".
* Check "public static void main(String[] args)" box.
* Click "Finish".

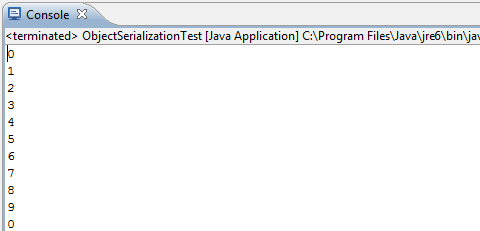
### Write code for main() method

* 1. **public** **static** **void** main(String[] args) {
  2. ObjectInputStream in = **null**;
  3. ObjectOutputStream out = **null**;
  4. **try** {
  5. out = **new** ObjectOutputStream(**new** BufferedOutputStream(
  6. **new** FileOutputStream("C:\\object.dat")));
  8. // Create an array of 10 SerializedObjects with ascending numbers
  9. SerializedObject[] objs = **new** SerializedObject[10];
  10. **for** (**int** i = 0; i < objs.length; i++) {
  11. objs[i] = **new** SerializedObject(i);
  12. }
  13. // Write the 10 objects to file, one by one.
  14. **for** (**int** i = 0; i < objs.length; i++) {
  15. out.writeObject(objs[i]);
  16. }
  17. // Write the entire array in one go.
  18. out.writeObject(objs);
  19. out.close();
  21. in = **new** ObjectInputStream(**new** BufferedInputStream(
  22. **new** FileInputStream("C:\\object.dat")));
  23. // Read back the objects, cast back to its original type.
  24. SerializedObject objIn;
  25. **for** (**int** i = 0; i < objs.length; i++) {
  26. objIn = (SerializedObject)in.readObject();
  27. System.*out*.println(objIn.getNumber());
  28. }
  29. SerializedObject[] objArrayIn;
  30. objArrayIn = (SerializedObject[])in.readObject();
  31. **for** (SerializedObject o : objArrayIn) {
  32. System.*out*.println(o.getNumber());
  33. }
  34. in.close();
  35. } **catch** (Exception ex) {
  36. ex.printStackTrace();
  37. }
  38. }

## Execute your program

### To run the program, right-click anywhere on the source file " ObjectSerializationTest.java" (or from the "Run" menu) ⇒ Choose "Run As" ⇒ "Java Application".

### The output appears on the "Console" panel



1. Count the number of lines

Write a program that will count the number of lines in each file that is specified on the command line. Assume that the files are text files. Note that multiple files can be specified, as in "java LineCounts file1.txt file2.txt file3.txt". Write each file name, along with the number of lines in that file, to standard output. If an error occurs while trying to read from one of the files, you should print an error message for that file, but you should still process all the remaining files.