# Programming in Java

**Dr. Nyein Aye Maung Maung**
**Dr. Eng (Ritsumeikan University, Japan)**
**Lecturer**
**Computer Engineering and Information Technology Dept.**
**Yangon Technological University**

# Course Schedule

| Week | Topics |
|---|---|
| Week 1 | Overview of JAVA, Data Types, Variables and Arrays |
| Week 2 | Operators and Control Statements |
| Week 3 | Classes and A closer look at Methods and Classes |
| Week 4 | Inheritance, Polymorphism, Abstraction and Encapsulation |
| Week 5 | Packages and Interfaces |
| Week 6 | Exception Handling and Multi-threaded Programming |
| Week 7 | String Handling |
| Week 8 | Exploring java.lang and More utilities classes |
| Week 9 | Java Collections Framework |
| Week 10 | Java I/O |
| Week 11 | Basic Graphical User Interface |
| Week 12 | Event Handling |
| Week 13 | Database Programming |
| Week 14 | Applet and Networking |

# Lecture 10

# Java Input/Output

# Outline of Class

- java.io package
  - File
  - Streams
    - Character Streams
    - Byte Streams

# Lecture Objectives

- To discover how I/O is processed in Java.
- To distinguish between text I/O and binary I/O.
- To be able to read and write both binary and text files.

# File

- A primary source and destination for data within many programs
- Does not specify how information is retrieved from or store, it describes the properties of a file itself
- The File class can be used to obtain file and directory properties, to delete and rename files and directories and to create directories.
- Constructors
  - File(String directoryPath)
  - File(String directoryPath, String filename)
  - File(File dirObj, String filename)
  - File(URI uriObj)

Eg.
```
File f1 = new File("/");
File f2 = new File("/","autoexec.bat");
File f3 = new File(f1,"autoexec.bat");
```
File("c://book") *File object for the directory*
File("c://book//test.dat") *File object for the file*
*To avoid platform dependency, create file under* ***the current working directory***
File ("sources/test.dat")

# File Utilities

- Methods which return Boolean
  - exists( ) → return true if the file or directory represented by the File obj exists
  - canRead( ) → return true if the file can be read
  - canWrite( ) → return true if the file can be written
  - isDirectory( ) → return true if the File obj represents a directory
  - isAbsolute( ) → return true if the File obj is created using an absolute path
  - isFile( ) → return true if the File obj represents a file
- Methods which return String
  - getName( ) → return last name of complete directory and file name
  - getPath( ) → return complete directory and file name
- Methods which return Long
  - length( ) → return size of file ( 0 if no file exists)

# File Utilities (Cont')

- Other methods
  - delete( ) → delete the file and return true if deletion succeeds.
  - renameTo( dest: File) → rename file or directory and return true if operation succeeds
  - mkdir( ) → create directory represented in File obj and return true if created successfully
  - list ( ) → extract the list of other files and directories inside

Summary:

- A File object encapsulates the properties of a file or a path, but it does not contain the methods for creating a file or for writing/reading data to /from a file (referred to as data input and output , or I/O for short).
- In order to perform I/O, you need to create objects using appropriate Ja va I/O classes.
- The objects contain the methods for reading/writing data from/to a file.
- There are two types of files: text and binary. Text files are essentially str ings on disk.

Example 1- Write a program that creates a File object for 'test.docx' under subdirectory 'sources' in the working directory. Next, create a directory named 'documents' under the working directory and check whether the director is successfully created or not. Then, illustrates whether the file exists after creation or not, the length of file, the file can be read or written, whether it is a director or file or absolute file name, and finally displays absolute path of file and last modified date and time.

```
import java.io.*;
public class TestFileClass {
  public static void main(String[] args) {
//Create file object
    File file = new File("test.txt");
    System.out.println("Does it exist? " + file.exists() );
    System.out.println("The file has " + file.length() + " bytes");
    System.out.println("Can it be read? " + file.canRead() );
    System.out.println("Can it be written? " + file.canWrite() );
    System.out.println("Is it a directory? " + file.isDirectory() );
    System.out.println("Is it a file? " + file.isFile() );
    System.out.println("Is it absolute? " + file.isAbsolute() ));
    System.out.println("Is it hidden? " + file.isHidden() );
    System.out.println("Absolute path is " + file.getAbsolutePath() );
    System.out.println("Last modified on " + new java.util.Date( file.lastModified() ));
  }
}
```

## Output

```
New directory created ? true
Does file exist? true
The file has 21827 bytes
Can it be read? true
Can it be written? true
Is it a directory? false
Is it a file? true
Is it absolute? false
Absolute path is /Users/nyeinayemaungmaung/De
sktop/
Programming Language Class/week 8 java IO/sourc
es/test.docx
Complete path is sources/test.docx
Last modified on Fri Jul 17 13:41:27 MMT 2015
```

Example 2: Write a program that prompts the user to enter a directory name and creates a directory under the working directory. The program displays the message "Directory created successfully" if a directory is created or "Directory already exists" if the directory already exists.

```
import java.util.Scanner;
import java.io.*;

public class DirTest {
  public static void main(String[] args) {
    System.out.print("Enter a directory name: ");
    Scanner input = new Scanner(System.in);
    String directoryName = input.nextLine();
//Create file object for new directory
    File f=new File(directoryName);
 if (    f.mkdir()  ) {
      System.out.println("Directory " + directoryName + " created");
    }
    else {
      System.out.println("Directory " + directoryName + " already exists");
    }
  }
}
```

to create subdirectories, use / (eg. dir1/dir2/dir3)

**Example 3**- Write a program that displays a list of files or directories inside a directory.

```
// Using directories.
import java.io.File;
class DirList {
public static void main(String args[]) {
String dirname = "documents";
File f1 = new File(dirname);
if (f1.isDirectory()) {
      System.out.println("Directory of " + dirname);
      String s[] = f1.list();
      for (int i=0; i < s.length; i++) {
            File f = new File(dirname + "/" + s[i]);
            if (f.isDirectory()) {
                  System.out.println(s[i] + " is a directory");
            } else {
                  System.out.println(s[i] + " is a file");
            }
            }
} else {
      System.out.println(dirname + " is not a directory");
            }
      }
}
```

```
Directory of documents
subDirectory is a directory
Test 2 is a file
Test 3 is a file
Test 4 is a file
```

Quiz 1: What is wrong about creating a File object using the following statement?

```
new File("c:\book\test.dat");
```

Answer: The \\ is a special character. It should be written as \\ in Java

Quiz 2:

- How do you check whether a file already exists? exists( )
- How do you delete a file? delete( )
- How do you rename a file? renameTo( File name)
- Can you find the file size (the number of bytes) using the File class? Yes
- How do you create a directory? mkdir( )

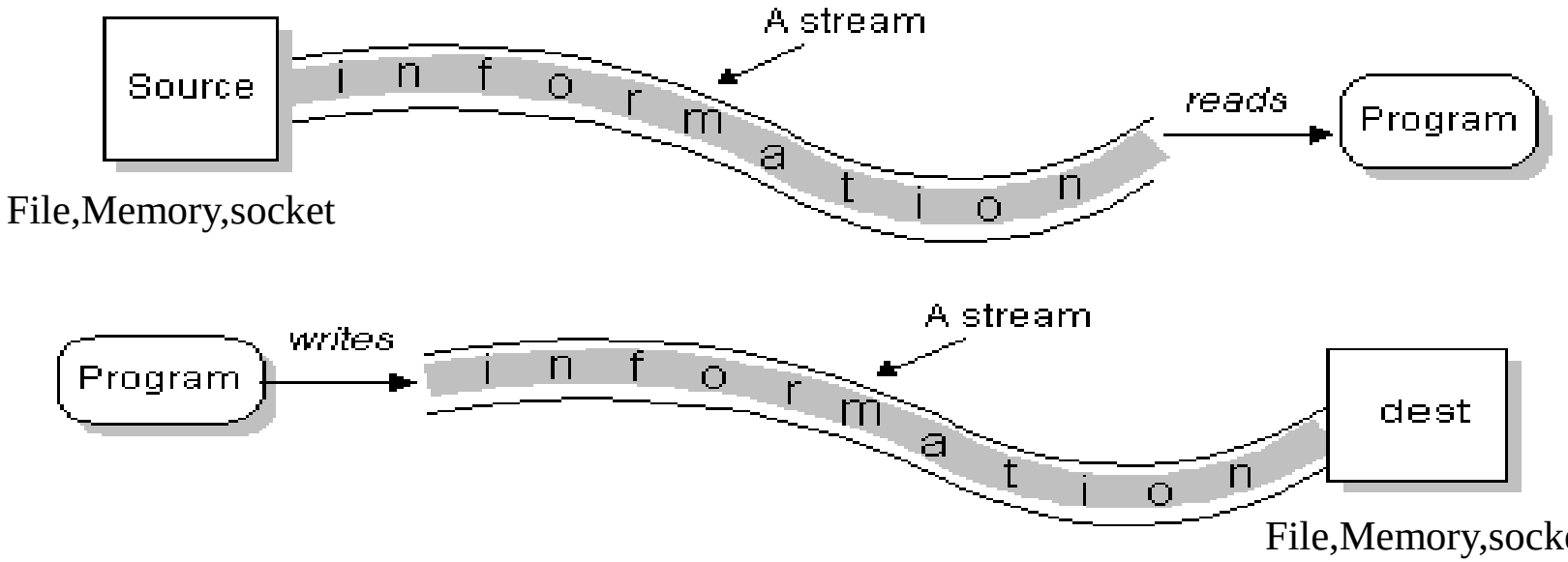
Quiz 3:

- Does creating a File object create a file on the disk? No

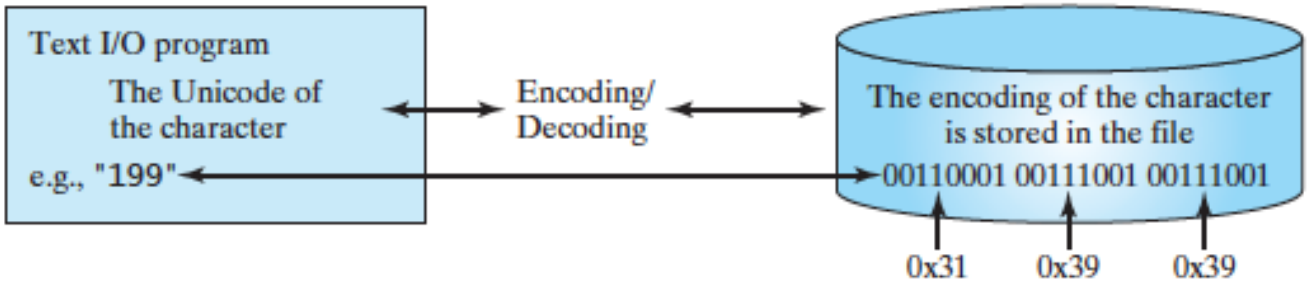
# Stream Overview

# Stream

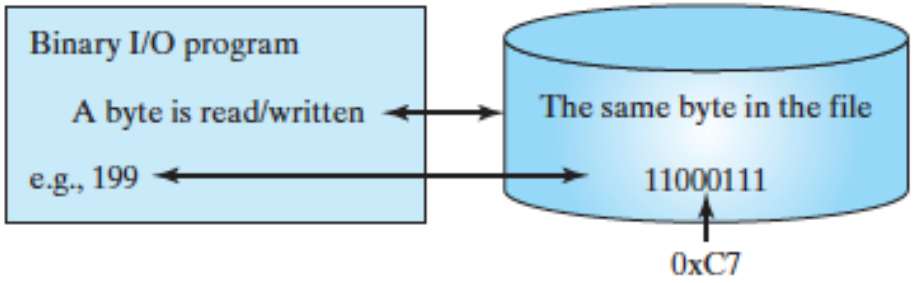
- Java programs perform I/O through streams
- A stream is a path of communication between a source of information and its destination
- A stream is linked to a physical device by Java I/O
  - **An input stream** can abstract many different kinds of input: from a disk file, a keyboard, or a network socket
  - **An output stream** may refer to the console, a disk file, or a network connection
- Same I/O classes and methods can be applied to any type of device (disk, keyboard, network socket)



# Text I/O and Binary I/O



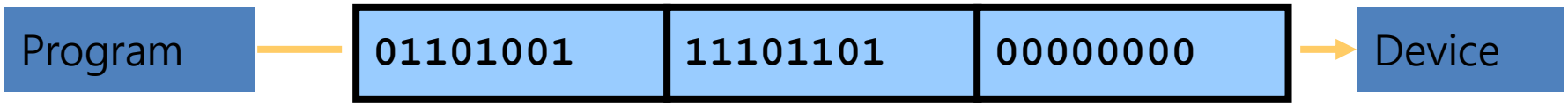

(a)



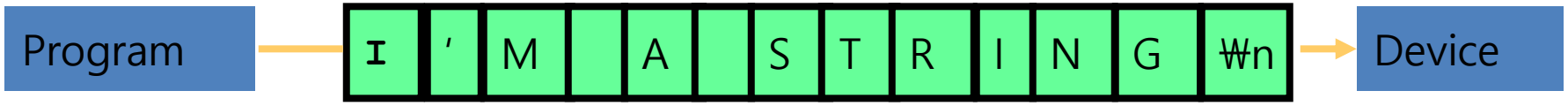

(b)

# Types of Stream

- Byte Stream (Byte – 8 bit stream)
  - for handling input and output of bytes
  - cannot work directly with Unicode characters




- Character Stream (16-bit UTF-16 characters stream)
  - for handling input and output of characters
  - write once, run anywhere



# How to do I/O

```
import java.io.*;
```

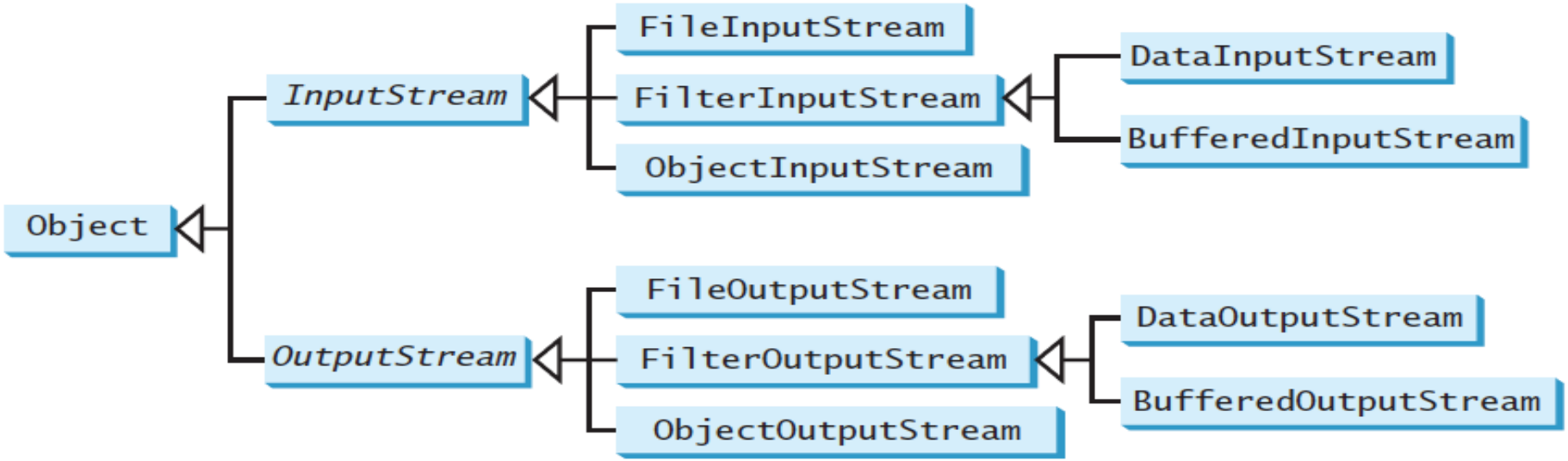
*1) Open* the stream

*2) Use* the stream (read, write, or both)

*3) Close* the stream

# Byte Stream (Binary I/O Classes)

- The abstract **InputStream** is the root class for reading binary data
- The abstract **OutputStream** is the root class for writing binary data




InputStream, OutputStream, and their subclasses are for performing binary I/O

- **Methods in InputSteam**
  - read(): int →Reads the next byte of data from the input stream
  - read(b: byte[]): int → Reads up to b.length bytes into array b from the input stream and returns the actual number of bytes read
  - close(): void → Closes this input stream and releases any system resources occupied by it

- **Methods in OutputStream Class**
  - write(int b): void → Writes the specified byte to this output stream
  - write(b: byte[]): void → Writes all the bytes in array b to the output stream
  - close(): void → Closes this output stream and releases any system resources occupied by it
  - flush(): void → Flushes this output stream and forces any buffered output bytes to be written out

# 1) FileInputStream/FileOutputStream

- Create an input/output stream for reading/writing bytes from/to files

FileInputStream(file: File) // Create a FileInputStream from a File object.

FileInputStream(filename: String) //Create from a file name

FileOutputStream(file: File)

FileOutputStream(filename: String)

FileOutputStream(file: File, append: boolean)

//If append is true, data are  appended to the existing file

FileOutputStream(filename: String, append: boolean)

Example 4 – A program that creates an output stream to the binary file and write values 1 to 10 to the file. Then, read the written values from the file and display back.

```
import java.io.*;
public class TestFileStream {
  public static void main(String[] args) throws IOException {
    // Create an output stream to the file
    FileOutputStream output = new FileOutputStream("temp.dat");
    // Output values to the file
    for (int i = 1; i <= 10; i++)
      output.write(i);    //same as invoking output.write((byte) i )
    // Close the output stream
    output.close();
    // Create an input stream for the file
    FileInputStream input = new FileInputStream("temp.dat");
    // Read values from the file
    int value;
    while ((value = input.read()) != -1) // return -1 when the end of the stream has been
                                                    reached
            System.out.print(value + " ");
    // Close the output stream
    input.close();
  }
}
```

## 2) DataInputStream/DataOutputStream

- **DataInputStream** reads bytes from the stream and converts them into appropriate primitive type values or strings.
  - extends FilterInputStream and implements the DataInput interface

**DataInputStream(InputStream InputStream)**

- **DataOutputStream** converts primitive type values or strings into bytes and outputs the bytes to the stream
  - extends FilterOutputStream and implements the DataOutput interface

**DataOutputStream(OutputStream outputStream)**

| java.io.DataInput Interface | java.io.DataInput Interface |
|---|---|
| +readBoolean(): boolean | +writeBoolean(b: boolean): void |
| +readByte(): byte | +writeByte(v: int): void |
| +readChar(): char | +writeBytes(s: String): void |
| +readFloat(): float | +writeChar(c: char): void |
| +readDouble(): double | +writeChars(s: String): void |
| +readInt(): int | +writeFloat(v: float): void |
| +readLong(): long | +writeDouble(v: double): void |
| +readShort(): short | +writeInt(v: int): void |
| +readLine(): String | +writeLong(v: long): void |
| +readUTF(): String | +writeShort(v: short): void |
| | +writeUTF(s: String): void |

Example 5 – A program that creates an output stream to the binary file to write students' test score (String name and Double score). Then, read the written values from the file and display back.

```
import java.io.*;
public class TestDataStream {
  public static void main(String[] args) throws IOException {
    // Create an output stream for file temp.dat
    DataOutputStream output = new DataOutputStream(new FileOutputStream("out.dat"));
    // Write student test scores to the file
    output.writeUTF("John");
    output.writeDouble(85.5);
    output.writeUTF("Jim");
    output.writeDouble(185.5);
    output.writeUTF("George");
    output.writeDouble(105.25);
    // Close output stream
    output.close();
    // Create an input stream for file temp.dat
    DataInputStream input = new DataInputStream(new FileInputStream("out.dat"));
    // Read student test scores from the file
    System.out.println( input.readUTF() + " " + input.readDouble() );
    System.out.println( input.readUTF() + " " + input.readDouble() );
    System.out.println( input.readUTF() + " " + input.readDouble() );
}
}
```

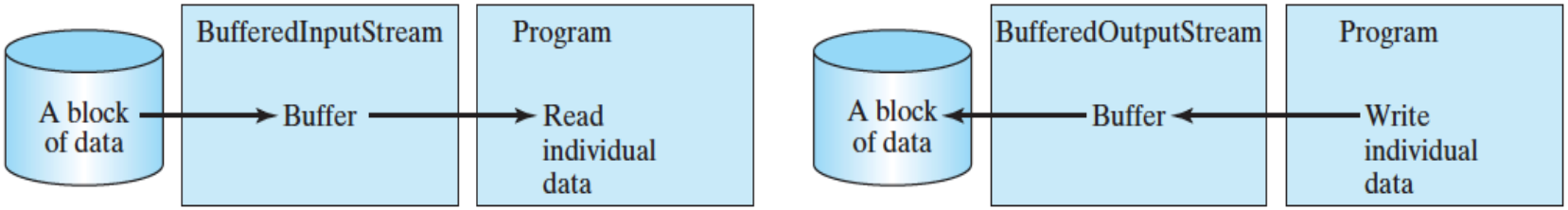
Example 6- Write a program that writes double values input from keyboard to a binary file until user enters the value 0.0. Then, read all the data from the file.

```
import java.io.*;
import java.util.*;
public class DetectEndofFile {
  public static void main(String[] args) throws IOException {
    try {
      DataOutputStream output = new DataOutputStream(new FileOutputStream("test.dat"));
      Scanner s=new Scanner(System.in);
      System.out.println("Enter double values:");
      double value=s.nextDouble();
      while(    value!=0.0     )
      {
                output.writeDouble(value);
                value=s.nextDouble();
      }
      output.close();
      DataInputStream input = new DataInputStream(new FileInputStream("test.dat"));
      while (true) {
        System.out.println(input.readDouble());
      }
    }
    catch (EOFException ex) {
              System.out.println("All data were read");
    } } }
```

```
Enter double values:
12.3
15.6
18.9
0.0
12.3
15.6
18.9
All data were read
```

# 3) BufferedInputStream/BufferedOutputStream

- Can be used to speed up input and output by reducing the number of disk reads and writes
- The whole block of data on the disk is read into the buffer in the memory once. The individual data are then delivered to your program from the buffer
- Using BufferedOutputStream , the individual data are first written to the buffer in the memory. When the buffer is full, all data in the buffer is written to the disk once




```
BufferedInputStream(InputStream inputStream)
BufferedInputStream(InputStream inputStream, int bufSize)
```

```
BufferedOutputStream(OutputStream outputStream)
BufferedOutputStream(OutputStream outputStream, int bufSize)
```

Example 7 - Write a program that copies the binary source file to the target binary file and displays the number of bytes in the file. The program should alert the user if the source file does not exist or if the target file already exists

```
import java.io.*;

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

    String sourceFileName="source.java";
    String targetFileName="target.java";
    // Check if source file exists
    File sourceFile = new File(sourceFileName);
    if ( !sourceFile.exists()) {
       System.out.println("Source file does not exist");
       System.exit(1);
    }
    // Check if target file exists
    File targetFile = new File(targetFileName);
    if ( targetFile.exists( ) ) {
      System.out.println("Target file already exists");
      System.exit(2);
    }
```

```
// Create an input stream
    BufferedInputStream input =
      new BufferedInputStream(new FileInputStream(sourceFile));

    // Create an output stream
    BufferedOutputStream output =
      new BufferedOutputStream(new FileOutputStream(targetFile));

    // Continuously read a byte from input and write it to output
    int  r, numberOfBytesCopied = 0;
    while ((r=input.read()) != -1) {
      output.write((byte) r);
      numberOfBytesCopied++;
    }

    // Close streams
    input.close();
    output.close();

    // Display the file size
    System.out.println(numberOfBytesCopied + " bytes copied");
  }
}
```

Quiz 1: After the program is finished, how many bytes are in the file t.dat ? Show the contents of each byte.

```
import java.io.*;
public class Test {
public static void main(String[] args) throws IOException {
DataOutputStream output = new DataOutputStream(
new FileOutputStream("t.dat"));
output.writeInt(1234);
output.writeInt(5678);
output.close();
}
}
```

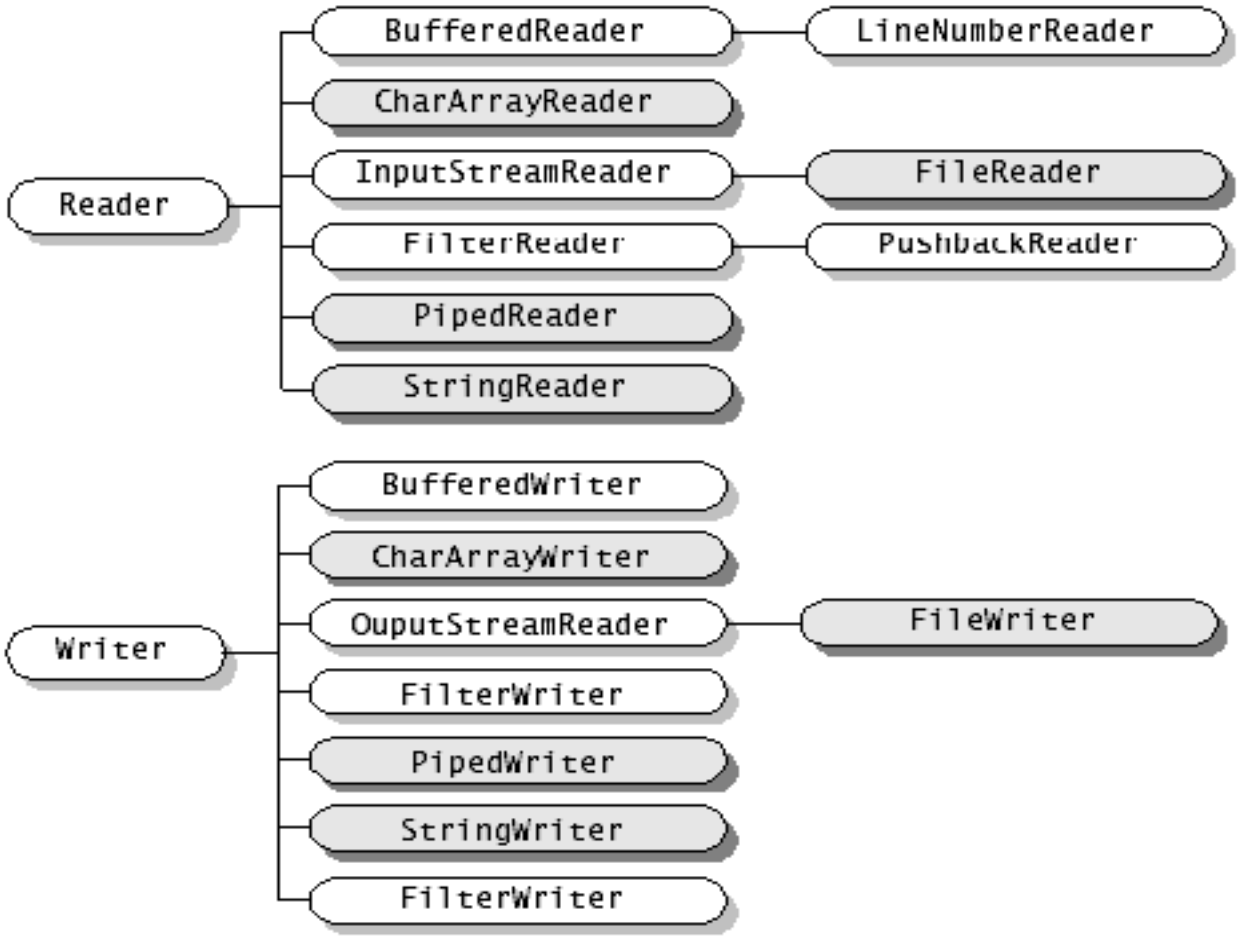
Answer: 8 byte (One Int value → 32 bits)
00 00 04 D2 00 00 16 2E

Quiz 2: For each of the following statements on a DataOutputStream output , how many bytes are sent to the output?

output.writeChar('A'); → 2 bytes
output.writeChars("BC"); → 4 bytes
output.writeUTF("DEF"); → 2 + 3 bytes (the first two bytes store the number of characters in the string.
Each ASCII character takes one byte in UTF)

# Character Stream (Text I/O Classes)

- The abstract **Reader** is the root class for reading text-based data and the abstract **Writer** is the root class for writing text-based data




Reader, Writer and their subclasses for performing character-based I/O

- **Methods in Reader Class**

| Method | Description |
|---|---|
| read() : int | Reads an integer representation of the next available char . -1 for EOF |
| read(char buffer[]): int | Attempts to read up to buffer.length characters into buffer and returns the actual number of char that were successfully read |
| read( char buffer[ ], int offset, int numChars) | Attempts to read up to numChars characters into buffer starting at buffer[offset], returning the number of characters successfully read. |

- **Methods in Writer Class**

| Method | Description |
|---|---|
| write() : int | Write a single character |
| Writer append(char ch) | Appends ch to the end of the invoking output stream. |
| Writer append(CharSequence chars) | Appends chars to the end of the invoking output stream. |
| Writer append(CharSequence chars, int begin, int end) | Appends the sub-range of chars specified by begin and end–1 to the end of the invoking ouput stream. |
| void write(char buffer[ ]) | Writes a complete array of characters to the invoking output stream. |
| abstract void write(char buffer[ ], int offset, int numChars) | Writes a subrange of numChars characters from the array buffer, beginning at buffer[offset] to the invoking output stream. |
| void write(String str) | Writes str to the invoking output stream |
| void write(String str, int offset, int numChars) | Writes a subrange of numChars characters from the string str, beginning at the specified offset. |

1) FileReader/FileWriter
   - Create a reader/writer to read/write the contents from/to a file

```
FileReader(String filePath)
FileReader(File fileObj)
```

```
FileWriter(String filePath)
FileWriter(String filePath, boolean append)
FileWriter(File fileObj)
FileWriter(File fileObj, boolean append)
```

2) BufferReader/BufferWriter
   - Improve performance by buffering input.
   - BufferReader has method to read a line of text, **`return null for EOF`**
     - String readLine()

```
BufferedReader(Reader inputStream)
BufferedReader(Reader inputStream, int bufSize)
```

```
BufferedWriter(Writer outputStream)
BufferedWriter(Writer outputStream, int bufSize)
```

## Example 8: FileWriter Demo

```
import java.io.*;
class FileWriterDemo {
        public static void main(String args[]) throws IOException {
            String source = "Now is the time for all good men\n"
                + " to come to the aid of their country\n"
                + " and pay their due taxes.";
                //create char array from String
                char buffer[] = source.toCharArray();
                FileWriter f0 = new FileWriter("file1.txt");
                for (int i=0; i < buffer.length; i += 2) {
                f0.write(buffer[i]);
                }
                f0.close();
                FileWriter f1 = new FileWriter("file2.txt");
                f1.write(buffer);
                f1.close();
                FileWriter f2 = new FileWriter("file3.txt");
                f2.write(buffer,buffer.length-buffer.length/4,buffer.length/4);
                f2.close();
                }
}
```

write(char buffer[ ], int offset, int numChars)

file1.txt

```
Nwi h iefralgo e
t oet h i ftercuty n a hi u ae.
```

file2.txt

```
Now is the time for all good men
 to come to the aid of their country
 and pay their due taxes.
```

file3.txt

```
nd pay their due taxes.
```

Example 9 – Write a program that prompts user to enter the name of file to be opened and then read and display the contents of the file.

```
import java.io.*;
import java.util.Scanner;
class FileInput{
public static void main(String[] args)
   {
      String fileName = null;
      try
      {
            Scanner keyboard = new Scanner(System.in);
            System.out.println("Enter file name:");
            fileName = keyboard.next();
            BufferedReader inputStream =
             new BufferedReader(new FileReader(fileName));
            String line = null;
     while((line = inputStream.readLine()) != null) {

         System.out.println("The first line in " + fileName + " is:");
         System.out.println(line);
     }
         inputStream.close();

      }
```

```
        catch(FileNotFoundException e)
        {
       System.out.println("File " + fileName + " not found.");
        }
        catch(IOException e)
        {
           System.out.println("Error reading from file " + fileName);
        }

    }
}
```

# Handling Text I/O using Scanner and PrintWriter

- **PrintWriter**
  - java.io.PrintWriter class can be used to create a file and write data to a text file

**+PrintWriter(file: File)**
**+PrintWriter(filename: String)**
+print(s: String): void
+print(c: char): void
+print(cArray: char[]): void
+print(i: int): void
+print(l: long): void
+print(f: float): void
+print(d: double): void
+print(b: boolean): void
Also contains the overloaded **println** methods.

  - **To append text to file**:

```
PrintWriter pw = new PrintWriter(new FileWriter(new
File("persons.txt"),true));
```

- **Scanner**
  - To read from a file,
    - Scanner input = new Scanner(**new File(filename)**);

Example 10: Write a program that writes lines to the file scores.txt. Each line consists of a first name (string), middle name initial (character),a last name(String) and a score (integer)

```
public class WriteData {
  public static void main(String[] args) throws Exception {
    java.io.File file = new java.io.File("scores.txt");   ///create file object
    if (file.exists()) {
      System.out.println("File already exists");
      System.exit(0);
    }

    // Create an output stream using PrintWriter
    java.io.PrintWriter output = new java.io.PrintWriter(file);

    // Write formatted output to the file
    output.print("John T Smith ");
    output.println(90);
    output.print("Eric K Jones ");
    output.println(85);

    // Close the file
    output.close();
  }
}
```

Example 11 : Write a program that read scores data from the file scores.txt. Each line consists of a first name (string), middle name initial (character),a last name(String) and a score (integer)

```
public class ReadData {
  public static void main(String[] args) throws Exception {
    // Create a File instance
    java.io.File file = new java.io.File("scores.txt");

    // Create a Scanner for the file
    Scanner input = new Scanner(file);

    // Read data from a file
    while (input.hasNext()) {
      String firstName = input.next();
      String mi = input.next();
      String lastName = input.next();
      int score = input.nextInt();
      System.out.println(
        firstName + " " + mi + " " + lastName + " " + score);
    }
    // Close the file
    input.close();
  }
}
```

Example 12 : Write a program that prompts the user to choose a file using a dialog box and display its contents on the console.

```
import java.util.Scanner;
import javax.swing.JFileChooser;
public class FileDialog {
public static void main(String[] args) throws Exception {
     JFileChooser fileChooser = new JFileChooser();
     if (fileChooser.showOpenDialog(null)==JFileChooser.APPROVE_OPTION){
 // Get the selected file
 java.io.File file = fileChooser.getSelectedFile();
 // Create a Scanner for the file
 Scanner input = new Scanner(file);
 // Read text from the file
 while (input.hasNext()) {
      System.out.println(input.nextLine());
 }
 // Close the file
 input.close();
 }
 else {
System.out.println("No file selected");
}
}
}
```

# BufferedReader && Scanner

- BufferedReader EOF detection
  - readLine() returns null
  - read() returns -1
- Scanner EOF detection
  - nextLine() throws exception
  - needs hasNextLine() to check first
  - nextInt(), hasNextInt(), …
- Scanner
  - nextInt(), nextFloat(), … for parsing types
- BufferedReader
  - read(), readLine(), … none for parsing types
  - needs StringTokenizer then wrapper class methods like Integer.parseInt(token)

# Reading Data from the Web

- Access data from a file that is on the Web using the file's URL
  - www.google.com/index.html
- to read data from a URL
  - create a URL object using the java.net.URL

```
try {
  URL url = new URL("http://www.google.com/index.html");
}
catch (MalformedURLException ex) {
  ex.printStackTrace();
}
```

Throws if url has syntax error

  - use the openStream() method defined in the URL class to open an input stream and use this stream to create a Scanner object

**Scanner input = new Scanner(url.openStream());**

Example 11: A program that reads file from the web.

```
public class ReadFileFromURL {
  public static void main(String[] args) {
    System.out.print("Enter a URL: ");
    String URLString = new Scanner(System.in).next();

    try {
      java.net.URL url = new java.net.URL(URLString);
      int count = 0;
      Scanner input = new Scanner(url.openStream());
      while (input.hasNext()) {
        String line = input.nextLine();
        count += line.length();
      }

      System.out.println("The file size is " + count + " bytes");
    }
    catch (java.net.MalformedURLException ex) {
      System.out.println("Invalid URL");
    }
    catch (java.io.IOException ex) {
      System.out.println("I/O Errors: no such file");
    }
  }
}
```

# ASSIGNMENTS

1. Write a program to create a file named Assignement1.txt if it does not exist. Append new data to it if it already exists. Write 100 integers created randomly into the file using text I/O. Integers are separated by a space.
2. Write a program to create a file named Assignment2.dat if it does not exist. Append new data to it if it already exists. Write 100 integers created randomly into the file using binary I/O.
3. Write a program that writes three lines inputted by the user from keyboard to the file.
4. Write a program that copies text from one file to another file.
5. Write a class named replaceText that creates a file by replacing a string in another text file with a new string. Strings should be inputted from user via keyboard.
6. Write a program that reads lines of characters from a text file and writes each line as a UTF-8 string into a binary file. Display the sizes of the text file and the binary file.

# Next Week Lecture

- Java Graphical User Interface