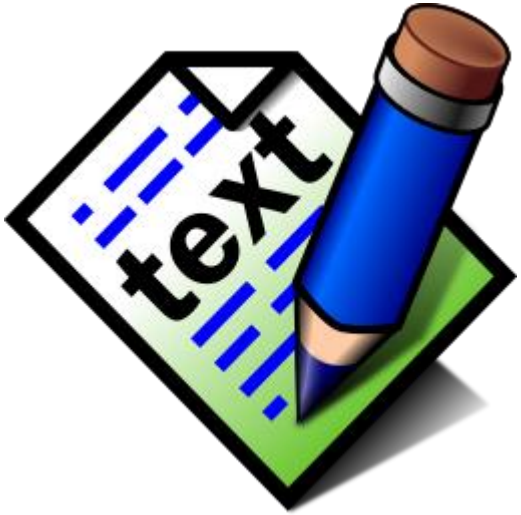


# File IO in Java

ITCS 209

Assistant Prof. Dr. Suppawong Tuarob

Faculty of Information and Communication Technology

Faculty of ICT
Mahidol University

# File Basics

- Recall that a file is **block** structured.  What does this mean?
- What happens when an application **opens** or **closes** a file?
- Every OS has its own EOF character and, for text files, its own EOL character(s).

# Streams

- Java file I/O involves **streams**. You write and read data to streams.
- The purpose of the stream abstraction is to keep program code independent from physical devices.
- Three stream objects are automatically created for every application: `System.in`, `System.out`, and `System.err`.

# Types of Streams

- There are 2 kinds of streams
  - byte streams
  - character streams

# Character Streams

- Character streams create **text** files.
- These are files designed to be read with a text editor.
- Java automatically converts its internal unicode characters to the local machine representation (ASCII in our case).

# Byte Streams

- Byte streams create **binary** files.
- A binary file essentially contains the memory image of the data. That is, it stores bits as they are in memory.
- Binary files are faster to read and write because no translation need take place.
- Binary files, however, cannot be read with a text editor.

# Classes

- Java has 6 classes to support stream I/O
- **File**: An object of this class is either a file or a directory.
- **OutputStream**: base class for byte output streams
- **InputStream**: base class for byte input streams
- **Writer**: base class for character output streams.
- **Reader**: base class for character input streams.
- **RandomAccessFile**: provides support for random access to a file.

- Note that the classes **InputStream**, **OutputStream**, **Reader**, and **Writer** are *abstract* classes.

# File class

```
File myDir = new File("C:\\CS311");

File myFile = new File("C:\\CS311\\junk.java");

File myFile = new File("C:\\CS311", "junk.java");

File myFile = new File(myDir, "junk.java").
```

# File methods

- `exists()`
- `isDirectory()`
- `isFile()`
- `canRead()`
- `canWrite()`
- `isHidden()`
- `getName()`

# File methods (cont)

- `getPath()`
- `getAbsolutePath()`
- `getParent()`
- `list()`
- `length()`
- `renameTo( newPath )`
- `delete()`
- `mkdir()`
- `createNewFile()`

# Writing TextFile using FileWriter Class

- The `FileWriter` class is a convenience class for writing character files.
- One version of the constructor take a `string` for a file name, another version takes an object of the `File` class.
- See FileWriterDemo.java

```java
public class FileWriterDemo {

    public static void main(String[] args)
    {
        BufferedWriter writer = null;
        File file = new File("log.txt");
        try {
            writer = new BufferedWriter(new FileWriter(file));
            writer.write("Hello World.\n"); //to write
            writer.append("Hello Mars.\n"); //to append

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        catch (IOException e) {
            e.printStackTrace();
        }finally
        {
                try {
                    if(writer != null)
                        writer.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
        }
    }
}
```

# Reading One Char at a Time

- See **StreamReaderDemo.java**
- The `read()` method returns an integer
- This integer should be cast to a `char`
- A value of -1 indicates the end of the stream has been reached.

```java
public class StreamReaderDemo {
    public static void main(String[] args)
    {
        BufferedReader reader = null;
        File file = new File("log.txt");
        try {
            reader = new BufferedReader
                (new InputStreamReader
                (new FileInputStream(file)));

            int c = -1;

            while((c = reader.read()) != -1)
            {
                char character = (char) c;
                System.out.print(character+"|");
            }

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        catch (IOException e) {
            e.printStackTrace();
        }finally
        {
                try {
                    if(reader != null)
                    reader.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
        }
    }
```

# Reading One Line at a Time

- See **LineReaderDemo.java**
- Use a `BufferedReader`
- The `readLine()` method returns a `String`.
- If the `String` is null, then the end of the stream has been reached.

```java
public class LineReaderDemo {
    public static void main(String[] args)
    {
        BufferedReader reader = null;
        File file = new File("log.txt");
        try {
            reader = new BufferedReader
                (new InputStreamReader
                (new FileInputStream(file)));

            String line = null;

            while((line = reader.readLine()) != null)
            {   line = line.trim();
                if(line.isEmpty()) continue;

                 System.out.println(line);
            }

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        catch (IOException e) {
            e.printStackTrace();
        }finally
        {
            try {
                if(reader != null)
                reader.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
```

# Reading One Word (Token) at a Time

- See **TokenReaderDemo.java**
- A word is a sequence of non-whitespace character.
- Use a Scanner
- hasNext() = true if there is one more word to read. False otherwise.
- next() returns the next word.

```java
public class TokenReaderDemo {
    public static void main(String[] args)
    {
        Scanner reader = null;
        File file = new File("log.txt");
        try {
            reader = new Scanner(file);

            String token = null;

            while(reader.hasNext())
            {   token = reader.next();
                System.out.print(token+"|");
            }

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        finally
        {   if(reader != null) reader.close();
        }
    }
}
```

# Shortcuts: FileUtils from Apache Commons IO

Apache Commons™
http://commons.apache.org/

```
File file = new File("log.txt");
//To write
try {
        FileUtils.write(file, "Hello World.\n", "UTF-8");
} catch (IOException e) {e.printStackTrace();}

//To append
try {
        FileUtils.write(file, "Hello Mars.\n", "UTF-8", true);
} catch (IOException e) {e.printStackTrace();}

//o read
try {
        List<String> lines = FileUtils.readLines(file, "UTF-8");
        for(String line: lines)
        {       System.out.println(line);
        }
} catch (IOException e) {e.printStackTrace();}
```



Note: Apache Commons IO 2.5+ requires character encoding as an input.