

# Java 4 WEB

Lesson 10 - IO, NIO

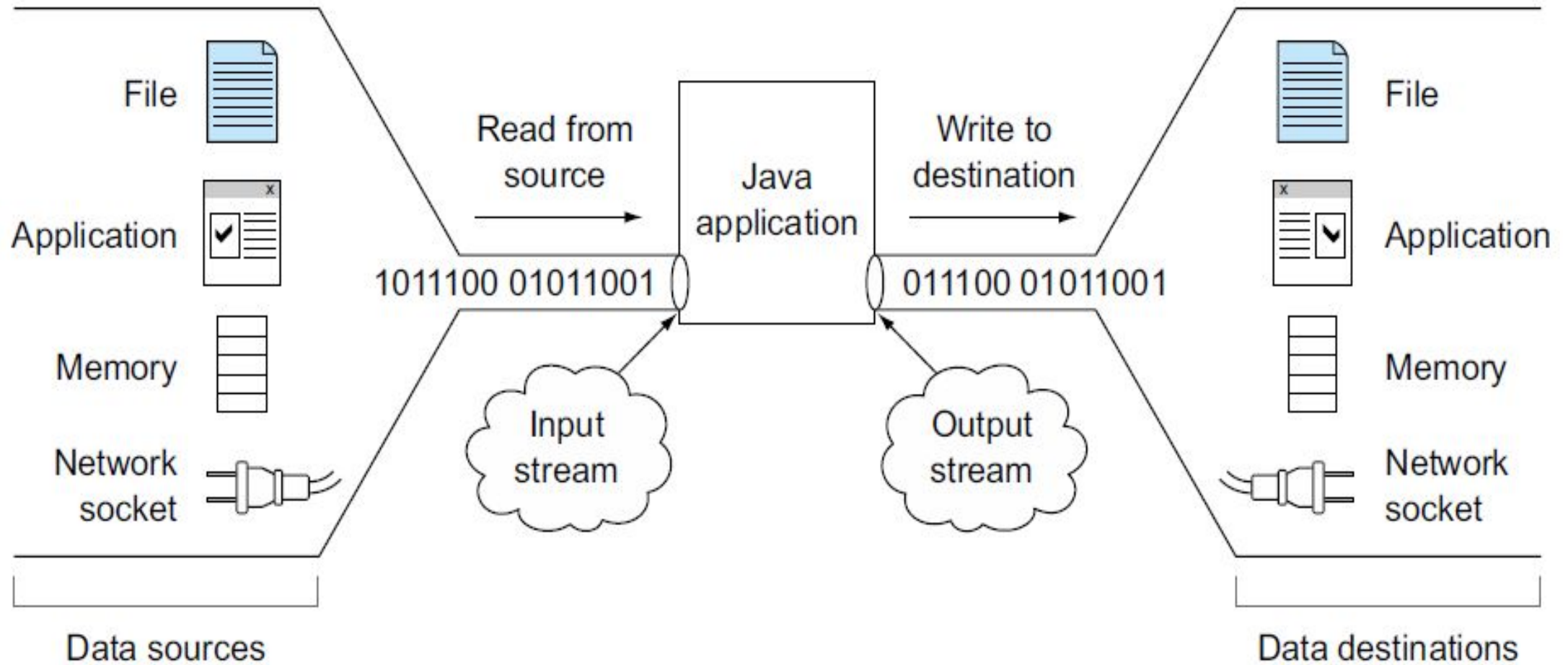
# Lesson goals

- Data exchange with external resources
- java.io – I/O:
- java.nio – NI/O: Non-blocking I/O 2,

# Glossary

- **File** – a record within a file system that stores user and system data (`java.exe`, `movie.mp4`...)
- **Directory** - a record within a file system that contains files as well as other directories (`Program Files`, `home`, ...)
- **Root directory** - the topmost directory in the file system (`c:\`, `/`, ...)
- **File system** - in charge of reading and writing data within a computer (`NTFS`, `FAT32`, ...)
- **Path** - String representation of a file or directory within a file system (`/user/home/zoo.txt`, `video/fun.3gp`, ...)

# IO



# File class

- `java.io.File.java` - represents the pathname of a particular file or directory on the file system
- File class can be used to represent **directories** as well as **files**
- Does **not require** file existence for creation of File object
- Types
  - **Relative** (`video/fun.mkv`) - concerning current directory
  - **Absolute** (`/home/video/fun.3gp`) - concerning root directory

# File instantiation

## File

```
File file = new File("/home/video/fun.3gp");
```

```
File parent = new File("/home");
```

```
File child = new File(parent, "video/fun.3gp");
```

## File separator

```
System.out.println(System.getProperty("file.separator"));
```

```
System.out.println(java.io.File.separator);
```

# File class methods

- `exists()` - is the file or directory exists.
- `getName()` - the name of the file or directory denoted by this path.
- `getAbsolutePath()` - the absolute pathname string of this path.
- `isDirectory()` - is the file denoted by this path is a directory.
- `isFile()` - is the file denoted by this path is a file.
- `length()` - the number of bytes in the file.
- `lastModified()` - the number of milliseconds since the epoch when the file was last modified.
- `delete()` - delete the file or directory. (if directory, then the directory must be empty).
- `renameTo(File)` - rename the file denoted by this path.
- `mkdir()` - create the directory named by this path.
- `mkdirs()` - create the directory named by this path including any nonexistent parent directories.
- `getParent()` - return the abstract pathname of this abstract pathname's parent or null
- `listFiles()` - return the files in the directory.

# Example

```
File file = new File("C:\\video\\fun.3gp");
System.out.println("File Exists: " + file.exists());

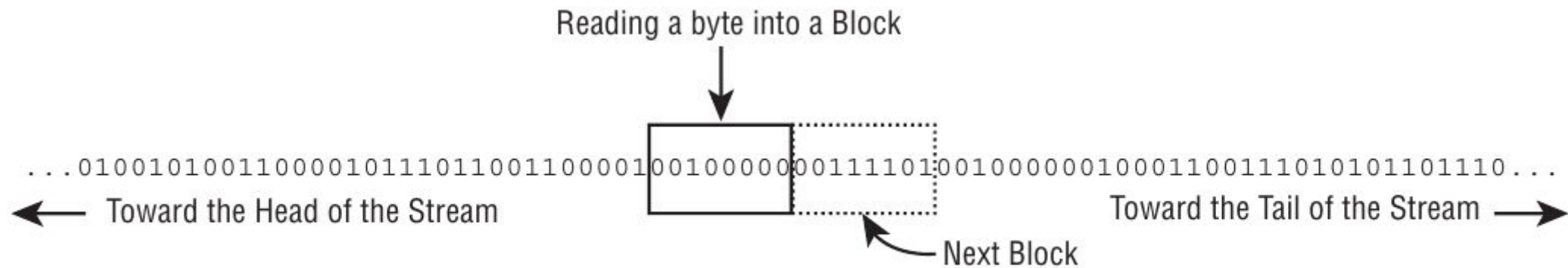
if (file.exists()) {
    System.out.println("Absolute Path: " + file.getAbsolutePath());
    System.out.println("Is Directory: " + file.isDirectory());
    System.out.println("Parent Path: " + file.getParent());

    if (file.isFile()) {
        System.out.println("File size: " + file.length());
        System.out.println("File LastModified: " + file.lastModified());
    } else {
        for (File subfile : file.listFiles()) {
            System.out.println("\t" + subfile.getName());
        }
    }
}
```



# Streams

A long, nearly never-ending “stream of water” with data presented one “wave” at a time.



# Built-in streams

- System.in
- System.err
- System.out

# Stream types by direction

- Input
- Output



# Stream types by content

- Byte Streams (InputStream/OutputStream)
- Character Streams (Reader/Writer)



# Stream types by abstraction level

- low-level stream
- high-level stream (wrapper)



```
try (FileInputStream fis = new FileInputStream("users.dat");  
    BufferedInputStream bfis = new BufferedInputStream(fis);  
    ObjectInputStream objectStream = new ObjectInputStream(bfis)  
) {  
    System.out.println(objectStream.readObject());  
}
```

# Examples

Class Name	Low/High Level	Description
FileInputStream	Low	Reads file data as bytes
FileOutputStream	Low	Writes file data as bytes
FileReader	Low	Reads file data as characters
FileWriter	Low	Writes file data as characters
BufferedReader	High	Reads character data from an existing Reader in a buffered manner, which improves efficiency and performance
BufferedWriter	High	Writes character data to an existing Writer in a buffered manner, which improves efficiency and performance
ObjectInputStream	High	Deserializes primitive Java data types and graphs of Java objects from an existing InputStream
ObjectOutputStream	High	Serializes primitive Java data types and graphs of Java objects to an existing OutputStream
InputStreamReader	High	Reads character data from an existing InputStream
OutputStreamWriter	High	Writes character data to an existing OutputStream
PrintStream	High	Writes formatted representations of Java objects to a binary stream
PrintWriter	High	Writes formatted representations of Java objects to a text-based output stream

# Most common Stream Operations

- `read()` - `InputStream`
- `read(byte[] b)` - `InputStream`
- `read(byte[] b, int off, int len)` - `InputStream`
- `write(...)` - `OutputStream`
- `close()` - `InputStream`, `OutputStream` (use automatic resource management, I/O stream)
- `flush()` - `OutputStream`
- `boolean markSupported()` - `InputStream`
- `mark(readlimit)/reset()` - `InputStream` (use `markSupported()` - before)
- `skip(n)` - `InputStream`
- `int available()` - `InputStream`

# File Input/Output Stream

```
void copy(File source, File destination) throws IOException {
    try (InputStream in = new FileInputStream(source);
        OutputStream out = new FileOutputStream(destination)) {
        int b;
        while((b = in.read()) != -1) {
            out.write(b);
        }
    }
}
```



# Buffered Input/Output Stream

```
void copy(File source, File destination) throws IOException {
    try (InputStream in = new BufferedInputStream(new FileInputStream(source));
        OutputStream out = new BufferedOutputStream(new FileOutputStream(destination))
    ) {
        byte[] buffer = new byte[1024];
        int lengthRead;
        while ((lengthRead = in.read(buffer)) > 0) {
            out.write(buffer, 0, lengthRead);
            out.flush();
        }
    }
}
```

# Object serialization\deserialization

- Implement the `java.io.Serializable` interface (not all built-in classes are serializable)
- Add *`serialVersionUID`*
- `NotSerializableException`

```
public static void main(String[] args) throws Exception {  
    File usersStore = new File("users.dat");  
    writeUsers(Arrays.asList(new User("Simon", 12, 3)), usersStore);  
    System.out.println(getUsers(usersStore));  
}
```

# Object serialization\deserialization

```
class User implements Serializable {  
    private static final long serialVersionUID = 1864942621136349408L;  
  
    String name;  
    int age;  
    transient int badHabbits;  
  
    public User(String name, int age, int badHabbits) {  
        this.name = name;  
        this.age = age;  
        this.badHabbits = badHabbits;  
    }  
  
    @Override  
    public String toString() {  
        return "User{" + "name='" + name + '\'';  
    }  
}
```

# ObjectOutputStream (serialization)

```
void writeUsers(List<User> users, File dataFile) throws IOException {  
    try (FileOutputStream fos = new FileOutputStream(dataFile);  
        BufferedOutputStream bos = new BufferedOutputStream(fos);  
        ObjectOutputStream out = new ObjectOutputStream(bos)  
    ) {  
        for (User user : users) {  
            out.writeObject(user);  
        }  
    }  
}
```

# ObjectInputStream (deserialization)

```
List<User> getUsers(File dataFile) throws IOException, ClassNotFoundException {
    List<User> users = new ArrayList<User>();

    try (FileInputStream fis = new FileInputStream(dataFile);
        BufferedInputStream bis = new BufferedInputStream(fis);
        ObjectInputStream in = new ObjectInputStream(bis)
    ) {
        while (true) {
            Object object = in.readObject();
            if (object instanceof User) {
                users.add((User) object);
            }
        }
    } catch (EOFException e) {
        // File end reached
    }

    return users;
}
```

# File Reader/Writer

```
void copy(File source, File destination) throws IOException {
    try (Reader reader = new FileReader(source);
        Writer writer = new FileWriter(destination)) {
        int c;
        while ((c = reader.read()) > -1) {
            writer.write(c);
        }
    }
}
```

# Buffered File Reader/Writer

```
void copy(File source, File destination) throws IOException {
    try (BufferedReader reader = new BufferedReader(new FileReader(source));
        BufferedWriter writer = new BufferedWriter(new FileWriter(destination))
    ) {
        String s;
        while ((s = reader.readLine()) != null) {
            writer.write(s);
            writer.newLine();
        }
    }
}
```

# Console

```
// Old school
try (
    InputStreamReader in = new InputStreamReader(System.in);
    BufferedReader reader = new BufferedReader(in)
) {
    String userInput = reader.readLine();
    System.out.println("You entered the following: " + userInput);
}

// New school
Console console = System.console();
if(console != null) {
    String userInput = console.readLine();
    console.writer().println ("You entered the following: "+userInput);
}
```



# NIO/NIO2

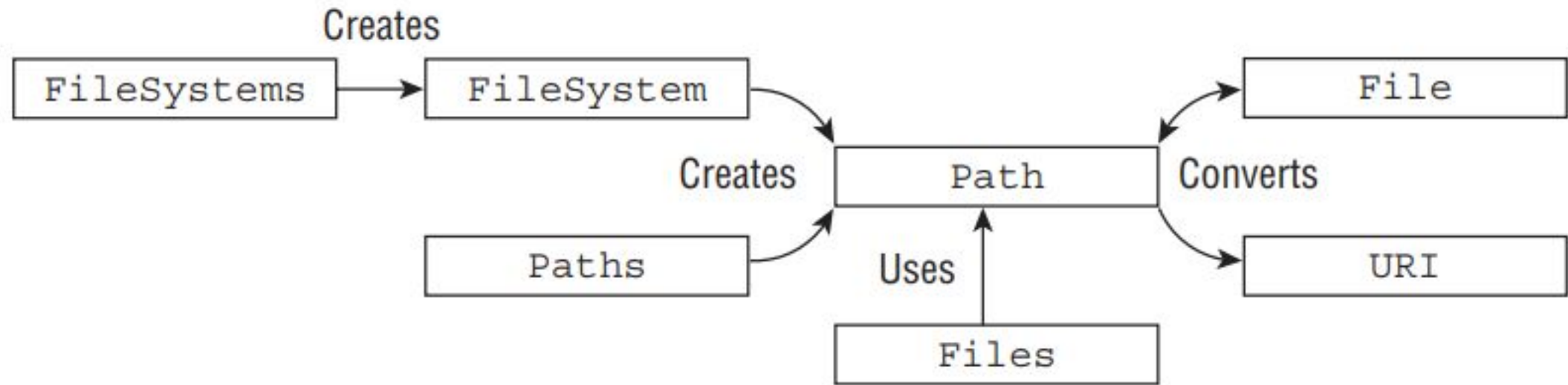
- `java.nio.file.Path` – **interface** represents a hierarchical path on the storage system to a file or directory.
- direct replacement for the legacy `java.io.File` class.
- Immutable (thread safe)
- Unlike the `File` class, the `Path` interface contains full support for symbolic links
- `java.nio.file.Paths` – factory for path instantiation
- Unlike the `File` class, the `Path` interface contains support for symbolic links
- Does not require file existence for creation of `Path` object
  - `'.'` - A reference to the current directory
  - `'..'` - A reference to the parent of the current directory

```
Path path1 = Paths.get("video/fun.3gp");
Path path2 = Paths.get("c:\\video\\fun.3gp");
Path path3 = Paths.get("/", "home", "video", "fun.3gp");

Path path4 = Paths.get(new URI("http://www.google.com"));
Path path5 = Paths.get(new URI("ftp://username:password@ftp.the-ftp-server.com"));

Path path1 = FileSystems.getDefault().getPath("video/3gp.png");
```

# Path overview



# Backward compatibility

```
File file = new File("video/fun.3gp");
```

```
Path path = file.toPath();
```

```
Path path = Paths.get("video/fun.3gp");
```

```
File file = path.toFile();
```

# Names hierarchy

```
Path songPath = Paths.get("c:\\home\\video\\..\\audio\\song.flac");  
System.out.println(songPath); // c:\home\video\..\audio\song.flac  
System.out.println(songPath.getParent()); // c:\home\video\..\audio
```

```
Path normalized = songPath.normalize();  
System.out.println(normalized); // c:\home\audio\song.flac  
System.out.println(normalized.getParent()); // c:\home\audio
```

```
Path absolute = normalized.toAbsolutePath();  
System.out.println(absolute); // c:\home\audio\song.flac
```

```
System.out.println(absolute.getRoot()); // c:\  
System.out.println(absolute.getNameCount()); // 3  
System.out.println(absolute.getFileName()); // song.flac  
System.out.println(absolute.getName(0)); // home
```

```
System.out.println(absolute.subpath(1, 3)); // audio\song.flac
```

# Relativize, resolve

```
Path path1 = Paths.get("C:\\Windows");
Path path2 = Paths.get("C:\\Program Files\\JetBrains");
Path path3 = Paths.get("JetBrains");

println(path1.relativize(path2)); // ..\Program Files\JetBrains
println(path2.relativize(path1)); // ..\..\Windows
println(path2.relativize(path3)); // Exception!

Path path4 = Paths.get("IntelliJ IDEA");

println(path2.resolve(path4)); // C:\Program Files\JetBrains\IntelliJ IDEA
println(path1.resolve(path2)); // C:\Program Files\JetBrains
```

# File existence, print and search folder content

```
try {  
    println(Paths.get(".").toRealPath());  
  
    Path idea = Paths.get("C:\\Program Files\\JetBrains\\IntelliJ IDEA");  
    println(idea.toRealPath());  
    println(idea.toRealPath(LinkOption.NOFOLLOW_LINKS));  
  
    Files.list(idea).forEach(c::println);  
    Files.walk(idea, 1).forEach(c::println); // DFS  
    Files.find(idea, 1, (p,a) -> p.startsWith("bin")).forEach(c::println);  
} catch (IOException e) {  
    println("Can not read path: " + e.getMessage());  
}
```

# Interacting with Files

```
Path idea = Paths.get("c:\\Program Files\\JetBrains\\IntelliJ IDEA 2017.1");
Path currentDir = Paths.get("."); // c:\

Files.exists(idea); // true
Files.isDirectory(idea); // true
Files.isSameFile(idea, idea.resolve("bin\\..")); // true
Files.isSameFile(idea, currentDir); // false

Path newIdea = idea.resolveSibling(Paths.get("IntelliJ IDEA 2018.1"));
Files.exists(newIdea); // false
Files.createDirectory(newIdea); // c:\Program Files\JetBrains\IntelliJ IDEA 2018.1
Files.createDirectories(newIdea); // c:\Program Files\JetBrains\IntelliJ IDEA 2018.1
Files.exists(newIdea); // true
```

# Interacting with Files. Copy, move

```
Path fun = Paths.get("c:\\Video\\fun.3gp");
Path moreFun = Paths.get("c:\\Video\\moreFun.3gp");
Path moreMoreFun = Paths.get("c:\\Video\\moreMoreFun.3gp");
Files.exists(fun); // true
Files.exists(moreFun); // false
Files.exists(moreMoreFun); // false

Files.copy(fun, moreFun); // shallow copy
Files.exists(moreFun); // true
Files.exists(moreMoreFun); // false

Files.move(fun, moreMoreFun, REPLACE_EXISTING); // shallow move
Files.exists(moreMoreFun); // true
```



# Interacting with Files. Delete

```
Files.delete(fun); // Exception
```

```
Files.deleteIfExists(fun); // silent delete
```

```
Files.delete(moreFun); // deleted
```

```
Files.delete(moreMoreFun); // deleted
```

```
Files.exists(fun); // false
```

```
Files.exists(moreFun); // false
```

```
Files.exists(moreMoreFun); // false
```

# Encoding

- `java.nio.charset.Charset`
- `java.nio.charset.StandardCharsets`
- The character encoding determines how characters are encoded and stored in bytes and later read back or decoded as characters
  - Win-1251
  - UTF-8
  - ISO-8859-5
  - ...

# Read\Write files with NIO2

```
Path src = Paths.get("c:\\users.txt");
Path dest = Paths.get("c:\\adminUsers.txt");
try (
    BufferedReader reader = Files.newBufferedReader(src, Charset.forName("US-ASCII"));
    BufferedWriter writer = Files.newBufferedWriter(dest, Charset.forName("UTF-16"))
) {
    String currentLine;
    while ((currentLine = reader.readLine()) != null) {
        writer.write(currentLine);
    }
} catch (IOException e) {
    // Handle file I/O exception...
}
```

# Read\Write files with NIO2

```
Path src = Paths.get("c:\\users.txt");  
try {  
    List<String> users = Files.readAllLines(src);  
    Stream<String> usersStream = Files.lines(src);  
} catch (IOException e) {  
    // Handle file I/O exception...  
}
```

# File Attributes

```
Path file = Paths.get("c:\\video\\fun.3gp");  
Files.isDirectory(file); // false  
Files.isHidden(file); // false  
Files.isReadable(file); // true  
Files.isExecutable(file); // true  
Files.isSymbolicLink(file); // false  
Files.getLastModifiedTime(file); // 2017-12-03T13:46:24.818731Z  
Files.size(file); // 13  
Files.getOwner(file); // BUILTIN\Admin (Alias)
```

# IO vs NIO2

<b>Legacy Method</b>	<b>NIO.2 Method</b>
<code>file.exists()</code>	<code>Files.exists(path)</code>
<code>file.getName()</code>	<code>path.getFileName()</code>
<code>file.getAbsolutePath()</code>	<code>path.toAbsolutePath()</code>
<code>file.isDirectory()</code>	<code>Files.isDirectory(path)</code>
<code>file.isFile()</code>	<code>Files.isRegularFile(path)</code>
<code>file.isHidden()</code>	<code>Files.isHidden(path)</code>
<code>file.length()</code>	<code>Files.size(path)</code>
<code>file.lastModified()</code>	<code>Files.getLastModifiedTime(path)</code>
<code>file.setLastModified(time)</code>	<code>Files.setLastModifiedTime(path, fileTime)</code>
<code>file.delete()</code>	<code>Files.delete(path)</code>
<code>file.renameTo(otherFile)</code>	<code>Files.move(path, otherPath)</code>
<code>file.mkdir()</code>	<code>Files.createDirectory(path)</code>
<code>file.mkdirs()</code>	<code>Files.createDirectories(path)</code>
<code>file.listFiles()</code>	<code>Files.list(path)</code>

# Literature

- [Basic I/O](#)
- [Class Charset](#)
- [Package java.io](#)

# Homework Task 1

1. Download ZIP **by java program** [users.zip](#) (direct link for download can be retrieved in browser)
2. Unzip it by program and find file named 'users.dat' (it can be in a child folder)
3. Try to deserialize 'users.dat' file into list of user objects (modify `User` class if needed)
4. Fix serialization problems in `User` class
5. Deserialize 'users.dat' file into list of user objects
6. `int allUsersCount = Count all deserialized users`
7. `String mostFavoriteEncodingAmongAllUsers = Find the most favorite encoding among users`
8. Between users who has higher access level than average, find '`int targetUserId`' = user id with biggest name length
9. Write ALL deserialized users names into file1. Every name in separate line (use next encoding '`{mostFavoriteEncodingAmongAllUsers}-{targetUserId}`'). Use system temp folder (files should be removed after usage)
10. Write ALL deserialized users names into file2. Every name in separate line (use next encoding '`{mostFavoriteEncodingAmongAllUsers}-{allUsersCount}`'). Use system temp folder (files should be removed after usage)
11. Answer – found files encodings and the difference of two files sizes in bytes
12. Cover by unit tests



# Homework Task 1

```
public class User {  
    private final int id;  
    private final String name;  
    private final int accessLevel;  
    private final String favoriteEncoding;  
  
    public User(int id, String name, int accessLevel, String favoriteEncoding) {  
        this.id = id;  
        this.name = name;  
        this.accessLevel = accessLevel;  
        this.favoriteEncoding = favoriteEncoding;  
    }  
  
    /* getters/setter and other are omitted */  
}
```

# Homework Task 2

Create a command line application that will create / delete / rename the folder with the specified name. The folder name and the number (or code) of the operation must be passed as incoming parameters in runtime. If folder with such name already exists – prompt user for confirmation of replacement.

# Homework Task 3

Implement Copying files. Create an application that will copy the file to the specified folder. In this case, the user should be able to select the type of stream: with and without buffering. Also, the application must display the time for which the operation was performed.

Testing the application by taking a large file (2-4GB for example) and copy it using buffering and without. Compare the execution time.