# I/O

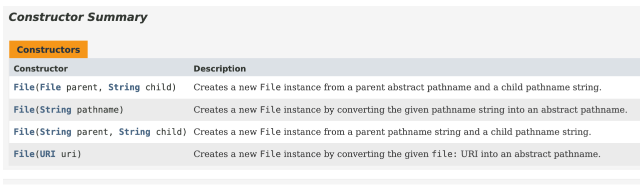
Files methods

## Methods of java.io.File

### delete

```
File notExisting = new File("notExistingFile");
//if the file does not exist, it does not throw an exception!!!
boolean isDeleted = notExisting.delete();
System.out.println(isDeleted); //false
```

## File constructors



*Constructor Summary*

Constructors

| Constructor | Description |
|---|---|
| `File(File parent, String child)` | Creates a new `File` instance from a parent abstract pathname and a child pathname string. |
| `File(String pathname)` | Creates a new `File` instance by converting the given pathname string into an abstract pathname. |
| `File(String parent, String child)` | Creates a new `File` instance from a parent pathname string and a child pathname string. |
| `File(URI uri)` | Creates a new `File` instance by converting the given `file:` URI into an abstract pathname. |

Constructor Summary

### File vs Path

```
java.io.File;
java.nio.file.Path;
```

## Path

Path is immutable. This line is ignored:

```
Path p1;
p1.normalize().relativize(Path.of("/lion"));
```

### Resolve

Calling resolve() with an absolute path as a parameter returns the absolute path.

```
var p1 = Path.of("/tmp")
var p2 = Path.of("enrico")
var p3 = Path.of("/users")

var resolve = p1.resolve(p2); // /tmp/enrico
resolve = p2.resolve(p1);   //  /tmp
resolve = p1.resolve(p3)  // /users
resolve = p3.resolve(p1)  // /tmp
```

resolve

## relativize

Both paths to be either absolute or relative.

```
Path path1 = Path.of("tmp/fish.txt"); //relative
Path path2 = Path.of("/user/friendly/birds.txt");  //absolute
// Exception in thread "main" java.lang.IllegalArgumentException: '
System.out.println(path1.relativize(path2));
```

relativize

## toRealPath

It does throw `IOException`

```
Path pom = Path.of("pom.xml");

///Users/enrico/github/ocp17/1Z0-829-preparation/pom.xml
System.out.println(pom.toRealPath());
```

`toRealPath()` throws `IOException` if the path does not exist.
toRealPath

## toAbsolutePath

It does **NOT** throw `IOException`

```
Path pom = Path.of("pom.xml").toAbsolutePath();
```

## System.in System.out

**Do not close!**
Because these are static objects, the System streams are shared by the entire application.
The JVM creates and opens them for us. They can be used in a try-with-resources statement or by calling close(), although closing them is not recommended.

## IO Classes

| Class Name | Level |
|---|---|
| FileInputStream | low |
| FileOutputStream | low |
| FileReader | low |
| FileWriter | low |
| BufferedInputStream | high |
| BufferedOutputStream | high |
| BufferedReader | high |
| BufferedWriter | high |
| ObjectInputStream | high |
| ObjectOutputStream | high |
| PrintStream | high |
| PrintWriter | high |

### Writer

`Writer` is an abstract class

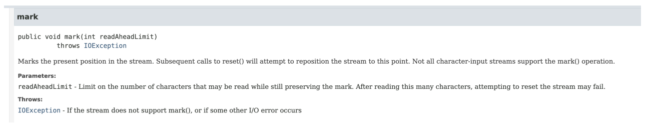
```
Writer writer = new PrintWriter(dest)
```

#### Writer in a try-with-resources

I need to declare `throws IOException` since `Writer` (abstract) implements `Closeable` which can throw IOE.

```
Writer w = c.writer();
  try (w) {...}
```

## Reader.mark()


**mark**

```
public void mark(int readAheadLimit)
          throws IOException
```

Marks the present position in the stream. Subsequent calls to reset() will attempt to reposition the stream to this point. Not all character-input streams support the mark() operation.

**Parameters:**

`readAheadLimit` - Limit on the number of characters that may be read while still preserving the mark. After reading this many characters, attempting to reset the stream may fail.

**Throws:**

`IOException` - If the stream does not support mark(), or if some other I/O error occurs

Reader.mark(limit)

### StringReader.mark()

*Marks the present position in the stream. Subsequent calls to reset() will reposition the stream to this point. Params: readAheadLimit – Limit on the number of characters that may be read while still preserving the mark. Because the stream's input comes from a string, there is no actual limit, so this argument must not be negative, but is otherwise ignored.*

## Serialization

### Serializable

If Object implemented `Serializable`, all objects would be serializable by default, defeating the purpose of having the Serializable interface. Serialization Example

### Deserialize Object

```
try {
    ....
//this is the proper way to read multiple items from a file
    while(true){
        var obj=ois.readObject();
        if(obj instanceof Person p){
          //do something
        }
    }
}catch(EOFException e){}
```

Deserialize

### Deserialize Object - extends

On deserialization, the JVM will call the no-arg constructor of the first non-serializable parent class it can find in the class hierarchy.

Serialization with Extends

## Console

Always checks if console is not null before using it.

```
Console console = System.console();
```

wrong way:

```
Console console = new Console(); //does not compile
```

### Console methods

```
String input = console.readLine("Type your name: ");
char[] pwd = console.readPassword("Type your pwd: ");
```

```
Reader reader = console.reader();
PrintWriter writer = console.writer()
```

Usage of Console

## Stream closed

`System.out` and `System.err` when declared within the try with resources can no longer log if used after they are closed.
System.err closed

## InputStream

### markSupported

```
if (inputStream.markSupported()){
    inputStream.mark(1);
    }
```

Usage of Mark