Processing files in Java

Stanislav I. Protasov

27/07/2015

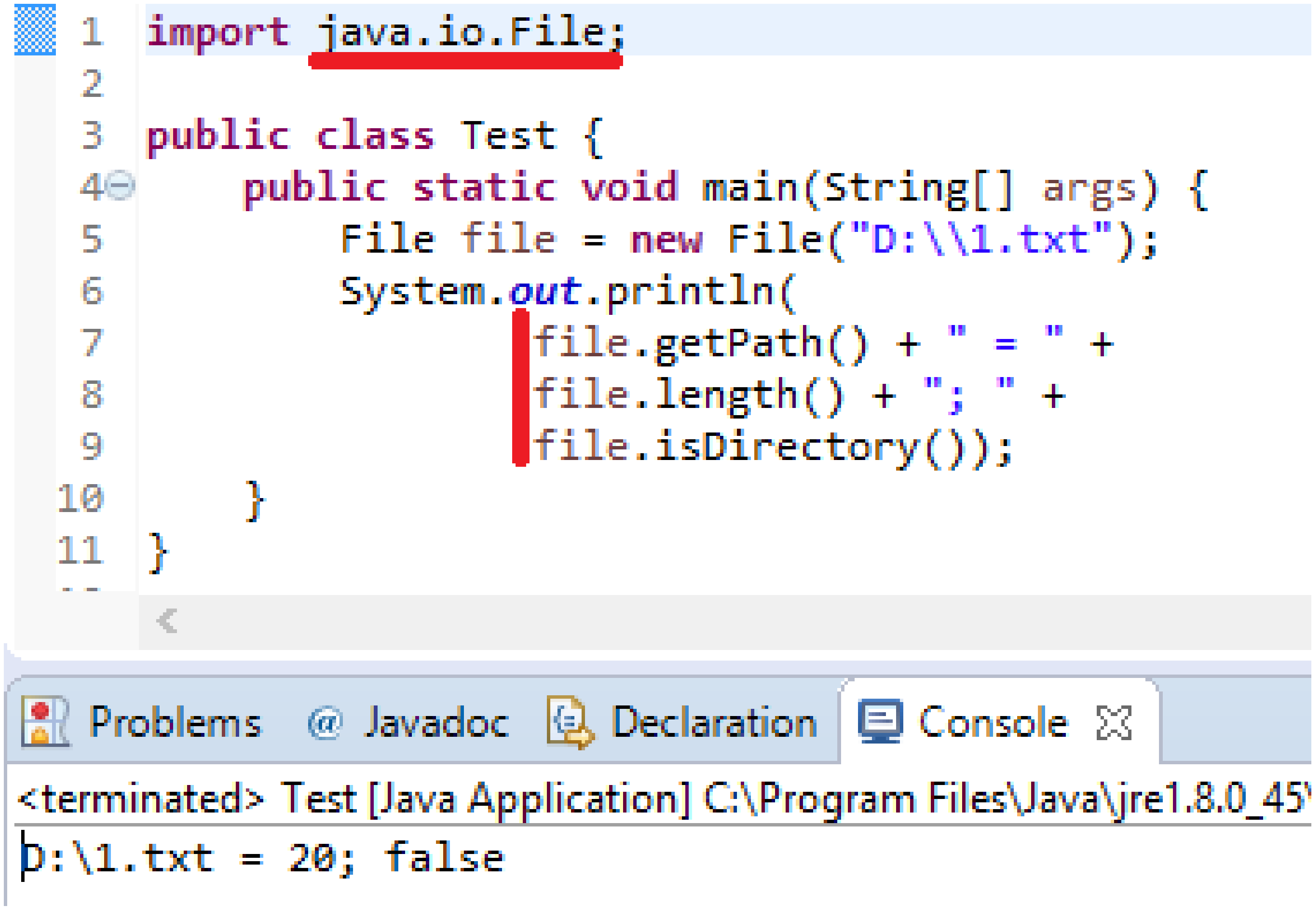
Agenda

- Files
  - What is file?
    - Content, size, attributes
  - What is the difference between byte and character?
    - Encodings
    - Binary files, text files
- Reading and writing files
  - How can you access the file?
  - Object serialization
  - Standard input, output, error

What is file?

- File systems
    - What is file
    - What is file system
    - File size
    - Data. What else?

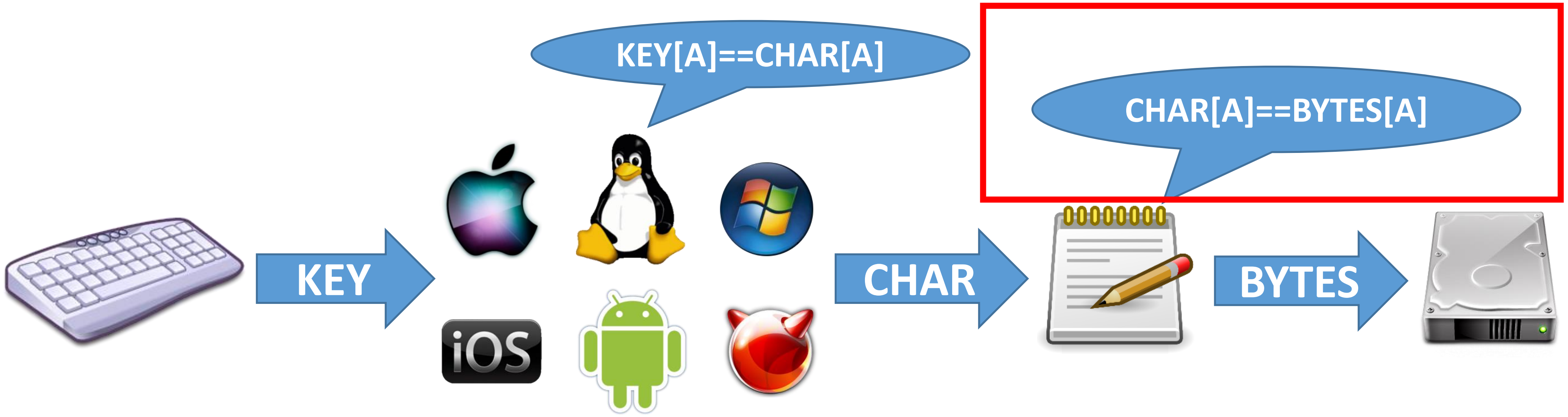
File attributes in Java



```java
import java.io.File;

public class Test {
    public static void main(String[] args) {
        File file = new File("D:\\1.txt");
        System.out.println(
                file.getPath() + " = " +
                file.length() + "; " +
                file.isDirectory());
    }
}
```

```
<terminated> Test [Java Application] C:\Program Files\Java\jre1.8.0_45
D:\1.txt = 20; false
```

Encodings



Encodings (Charsets)

- Ages ago:
  - 1 char = 7 bit (ASCII) $\longrightarrow$ Я = ?
  - We have codepages!
    - KOI8-R $\longrightarrow$ Я = 0xF1
    - Windows-1251 $\longrightarrow$ Я = 0xDF
    - Windows-1252 $\longrightarrow$ Я => ß
    - …
- Today
  - 1 character = 1, 2, 4 bytes. Stop, what?
  - Unicode – let's get all languages together!
    - Codepoints (unified representation) Я= U+042F
    - Unicode-16 LE, BE $\longrightarrow$ Я = 0x2F04; 0x042F
    - Unicode-32 LE, BE
    - UTF-8, … $\longrightarrow$ ЯQ = D0 AF 51
    - BOM $\longrightarrow$ FE FF, FF FE, FE BB BF

Binary and text files

Binary file – file of **bytes**

    this is about ***storing data***

Text file – file of **chars**

    this is about ***representing text***

This is a_____text
File

54 68 69 73 20 69 73 20 61 09 74 65 78 74 0D 0A

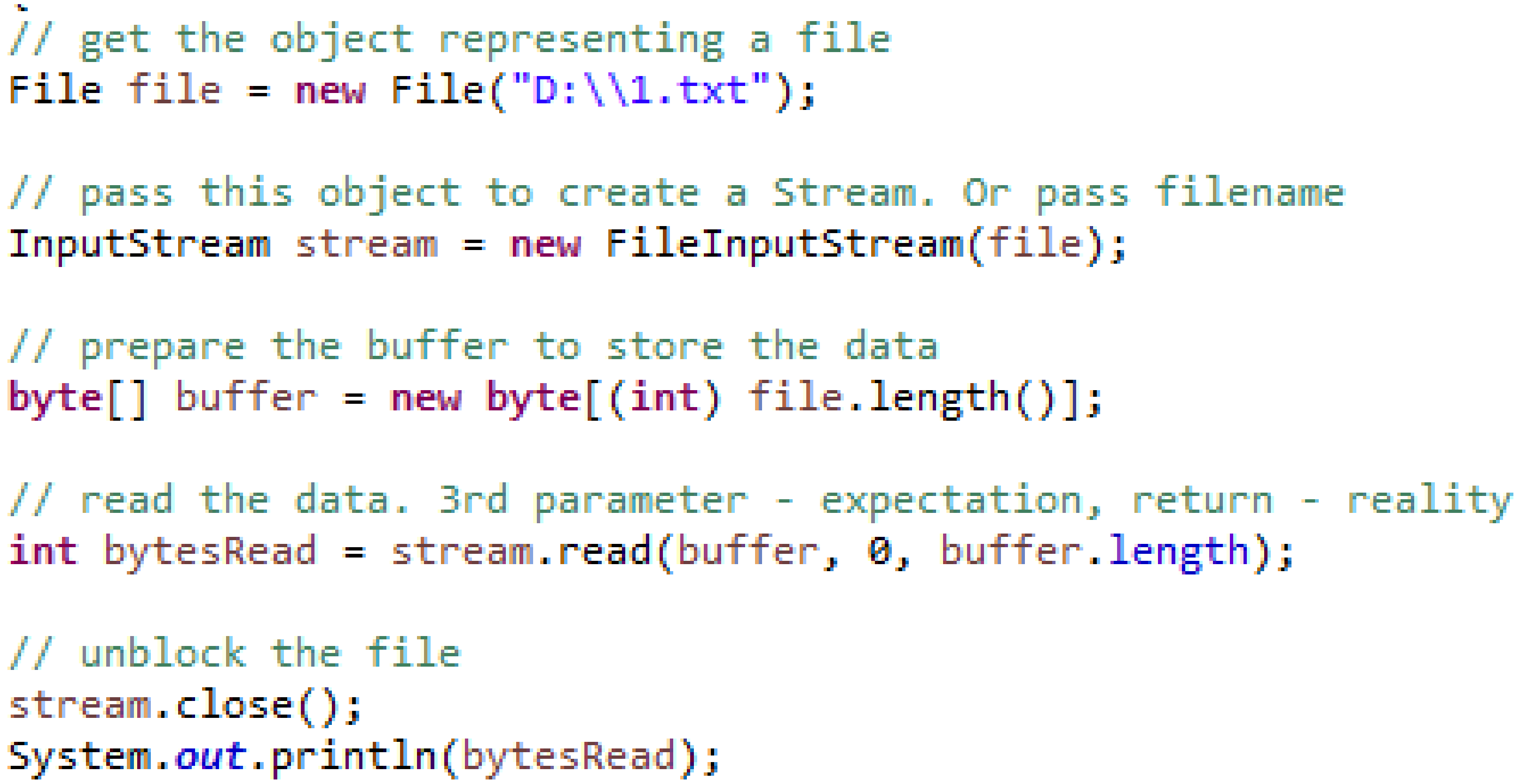
46 69 6C 65

Streams

Stream is a sequential representation of data (bytes).
Streams can represent **files**, sockets, console input etc.

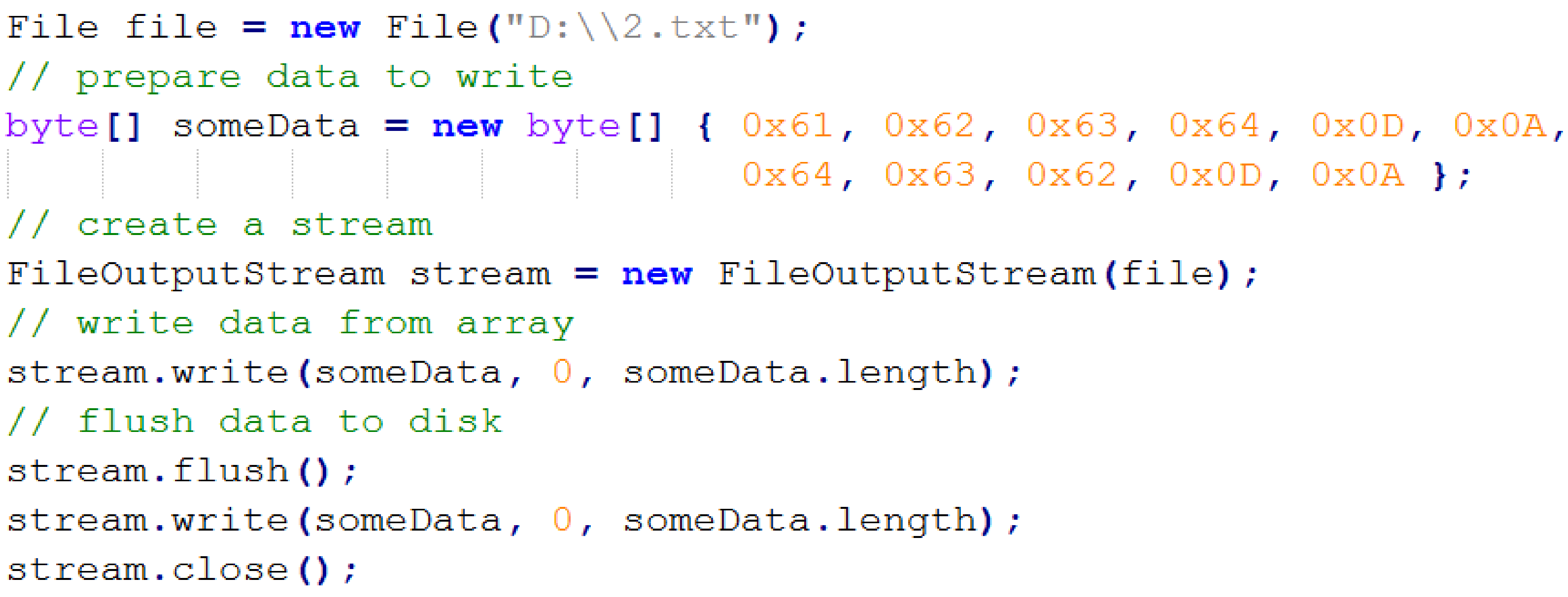
- **java.io.InputStream**
  – has read(…) methods to get the data
- **java.io.OutputStream**
  – has write(…) methods to write the data

Reading binary files with java.io.FileInputStream



```java
// get the object representing a file
File file = new File("D:\\1.txt");

// pass this object to create a Stream. Or pass filename
InputStream stream = new FileInputStream(file);

// prepare the buffer to store the data
byte[] buffer = new byte[(int) file.length()];

// read the data. 3rd parameter - expectation, return - reality
int bytesRead = stream.read(buffer, 0, buffer.length);

// unblock the file
stream.close();
System.out.println(bytesRead);
```

Writing binary files with java.io.FileOutputStream

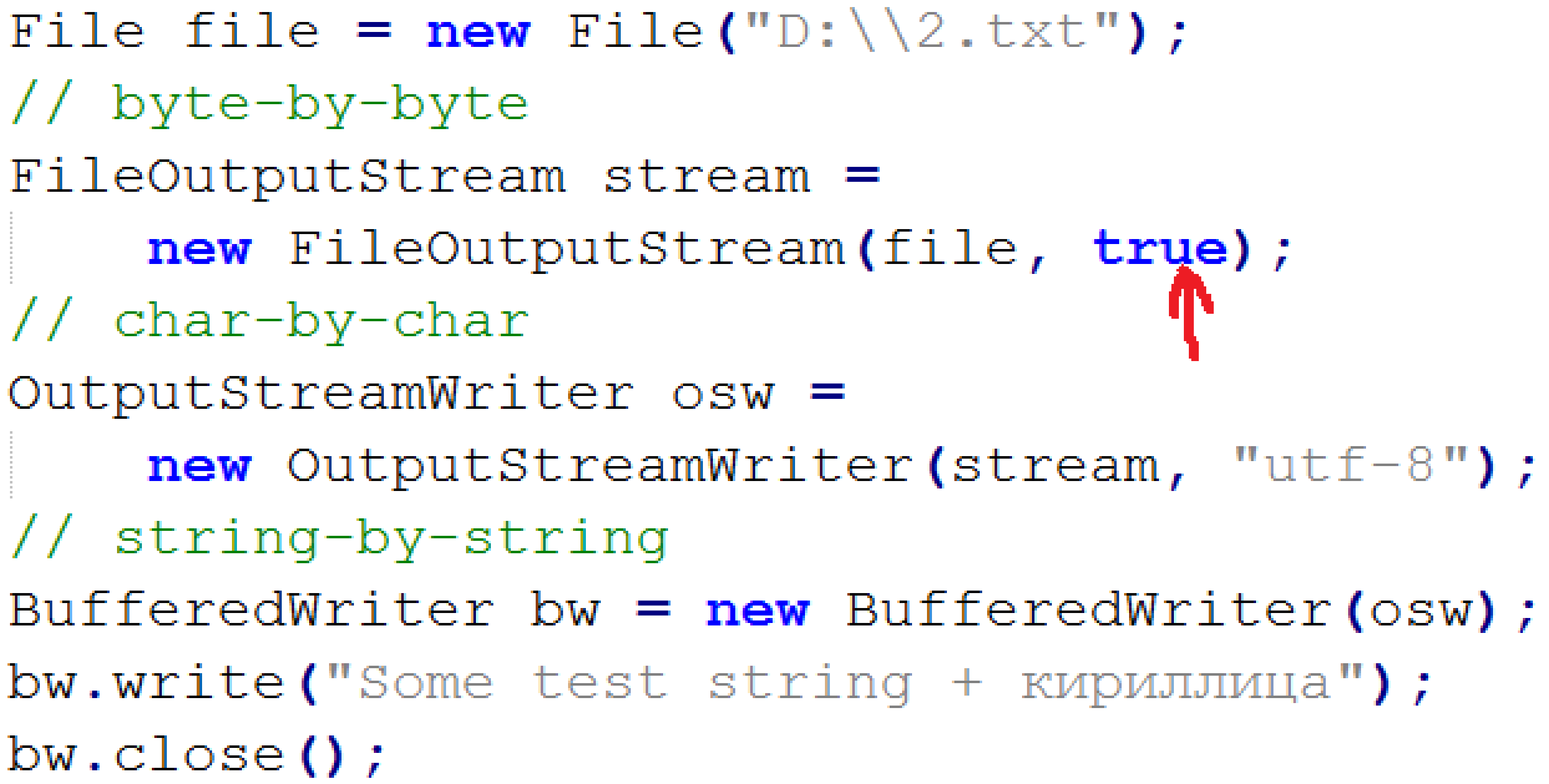


```java
File file = new File("D:\\2.txt");
// prepare data to write
byte[] someData = new byte[] { 0x61, 0x62, 0x63, 0x64, 0x0D, 0x0A,
                               0x64, 0x63, 0x62, 0x0D, 0x0A };
// create a stream
FileOutputStream stream = new FileOutputStream(file);
// write data from array
stream.write(someData, 0, someData.length);
// flush data to disk
stream.flush();
stream.write(someData, 0, someData.length);
stream.close();
```

Reading text files with java.io.BufferedReader

```java
// same start
File file = new File("D:\\1.txt");
InputStream stream = new FileInputStream(file);

Charset cs = Charset.forName("utf-8");
// here we can read chars
InputStreamReader charStream  = new InputStreamReader(stream, cs);
// this allows us to read line-by-line
BufferedReader textFileReader = new BufferedReader(charStream);
System.out.println(textFileReader.readLine());
textFileReader.close();
```

Writing text files with java.io.BufferedWriter / FileWriter



```
File file = new File("D:\\2.txt");
// byte-by-byte
FileOutputStream stream =
    new FileOutputStream(file, true);
// char-by-char
OutputStreamWriter osw =
    new OutputStreamWriter(stream, "utf-8");
// string-by-string
BufferedWriter bw = new BufferedWriter(osw);
bw.write("Some test string + кириллица");
bw.close();
```

Reading text files with java.util.Scanner

```java
// initialize scanner with encoding provided
Scanner sc = new Scanner(
                new File("D:\\1.txt"), "utf-8");
// default delimiter is space - " "
// you can set other using
// sc.useDelimiter(";");

// read integer value
int number = sc.nextInt();
// read word
String word = sc.next();
// read until line ends
String line = sc.nextLine();
System.out.println(number + " " + word);
System.out.println(line);
sc.close();
```

System.in, System.out, System.err

- ***.in*** – console input
- ***.out*** – console *output*
- ***.err*** – console error *output*

```java
package files;
import java.util.Scanner;

public class Splitter {
    public static void main(String[] args) {
        String line = "";
        Scanner sc = new Scanner(System.in);
        do {
            line = sc.hasNextLine() ? sc.nextLine() : "";
            System.out.println(line.toUpperCase());
            System.err.println(line.toLowerCase());
        } while (!line.equals(""));
        sc.close();
    }
}
```

```
java files.Splitter 1> out.txt 2> err.txt < in.txt
```

What is object?

Object =

IDENTITY

+

STATE

+

BEHAVIOUR

Serialization and Deserialization

- **Commonly** serialization is a restorable **object state** representation as byte stream.
- **Serialization/Deserialization** – is a very convenient way to persist Java objects.
  - **Serialization** takes all fields from object (and superclasses), and writes them into Stream.
  - **Deserialization** reads fields form Stream and creates an object filled with this values.
  - This mechanism allows to save object graphs.

```
AC ED 00 05 73 72 00 0A 66 69 6C 65 73 2E 50 61  ¬í ♣sr ◙files.Pa
63 6B C9 5F 96 FD 18 2A 6B 9F 02 00 03 49 00 06  ckÉ_–ý↑*kŸ☻ ♥I ♠
6E 75 6D 62 65 72 5B 00 05 61 72 72 61 79 74 00  number[ ♣arrayt
02 5B 49 4C 00 06 73 74 72 69 6E 67 74 00 12 4C  ☻[IL ♠stringt ↕L
6A 61 76 61 2F 6C 61 6E 67 2F 53 74 72 69 6E 67  java/lang/String
3B 78 70 00 00 00 0D 75 72 00 02 5B 49 4D BA 60  ;xp    ♪ur ☻[IMº`
26 76 EA B2 A5 02 00 00 78 70 00 00 00 05 00 00  &vê²¥☻  xp    ♣
00 00 00 00 00 01 00 00 00 02 00 00 00 03 00 00       ☺   ☻   ♥
00 04 74 00 0B 53 6F 6D 65 20 53 74 72 69 6E 67   ♦t ♂Some String
```

Serialization

Unlike other languages, to make object serializable you should only implement *java.io.Serializable* interface

```
public class Pack implements Serializable {
        ...
}
```

Serialization

… and then

```
Pack p = new Pack(13, "Some String", 5);
FileOutputStream fos = new FileOutputStream("D:\\temp.out");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(p);
oos.close();
```

Deserialization

Also that easy:

```
FileInputStream fis = new FileInputStream("D:\\temp.out");
ObjectInputStream ois = new ObjectInputStream(fis);
Pack p2= (Pack)ois.readObject();
ois.close();
```

Serialization: NB

- Serialization stores class graphs
- Class version can be provided explicitly:

```
static final long serialVersionUID = 42L;
```

- You can send a representative instead of you:

```
Object writeReplace() throws ObjectStreamException {
    return ":)";
}
```

- … and then comes readResole()
- You can skip fields using keyword:

```
private transient Pack foo;
```

s.protasov@innopolis.ru