IO

CONTENT

Java.io API

Files and directories

Read and write – console

BufferedReader/-Writer

Files

Streams

FILES AND DIRECTORIES

- File system: hierarchically organized files and directories
- File: record in a file system with data
- Directory: record in a file system containing files and other directories
- Root directory: top directory in the file system
  - Windows: c:\ (or another letter)
  - Linux: /
- Different OS use different file systems
- Path: String representation of a file/directory

FILE CLASS

- Java class in the java.io API
- Instance of a File object has a pathname of a file/directory in the file system
- Used to read information about files and directories, but cannot read content of file
  - Such as list contents
  - Create/delete files/directories

FILE CLASS

- File object is initialized with a path:
  - Absolute: looking from the root of the file system (start with root)
  - Relative: looking from the current directory
- File separator might differ for OS, can be requested by:
  - System.getProperty(“file.separator”)
  - java.io.File.separator
- Creating a file:
  - File file = new File(“/home/directory/file.txt”);

METHODS IN THE FILE CLASS

- exists()
- getName()
- getAbsolutePath()
- isDirectory()
- isFile()
- length() > returns nr of bytes in file
- lastModified()
- delete()
- renameTo(File)
- mkdir() > create directory named by path
- mkdirs() > create directory and nonexisting parent directories
- getParent() > parent directory
- listFiles() > returns File[] in directory

EXERCISE

Use Java to:

- Create a new folder “example”
- Add a file “log.txt” to this folder
- Add a folder “messages” to the folder example
- Create a txt file for messages in this folder
- Print the pathname of the messages.txt to the console
- Verify both files exist
- Verify whether example/nothere.txt exists

I/O STREAMS

- Input/output streams
  - Input: reading
  - Output: writing
- Not related to the Stream API

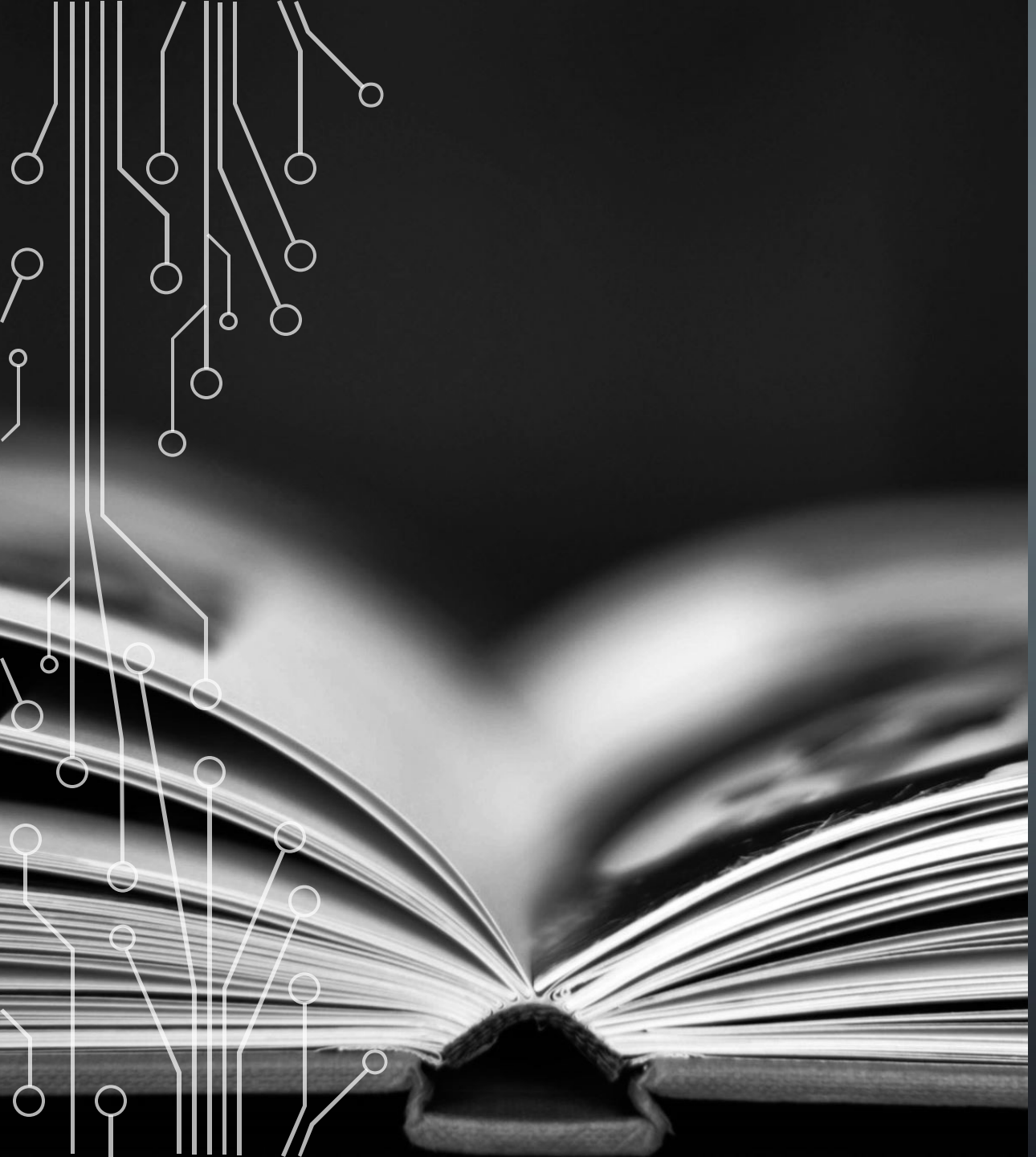
- Files can be read or written using a stream.
- Stream is a long list of sequential data elements in the file
- Data is read/written one chunk of the list at time

I/O STREAMS

- Very many different streams: streams that read bytes, character, strings, groups of bytes etc
- Most streams in the java.io API work with (arrays of) bytes at a time
- Used for processing and writing to files, but also for sequential data source
- Example of output stream: System.out
- Very many different streams in java.io API

BYTE VS CHARACTER STREAM

- Byte: classes with the word stream in it
- Character: classes with the word reader/writer in it
- Different way of reading the stream
  - Byte: used for reading/writing all binary/byte data
  - Character: used for reading/writing string/character data
- Technically, you can always use byte streams (because Strings fit in).
- Character streams are for convenience, no need to mind byte encoding of files



INPUT AND OUTPUT

- Stream classes with the word input are paired with a class with a similar name but output instead of input
  - E.g. FileInputStream and FileOutputstream
- The data that is written by a specific inputstream can be read with the almost-same-name outputstream
- Same is true for reader and writer
  - E.g. FileReader and FileWriter

LOW VS HIGH LEVEL

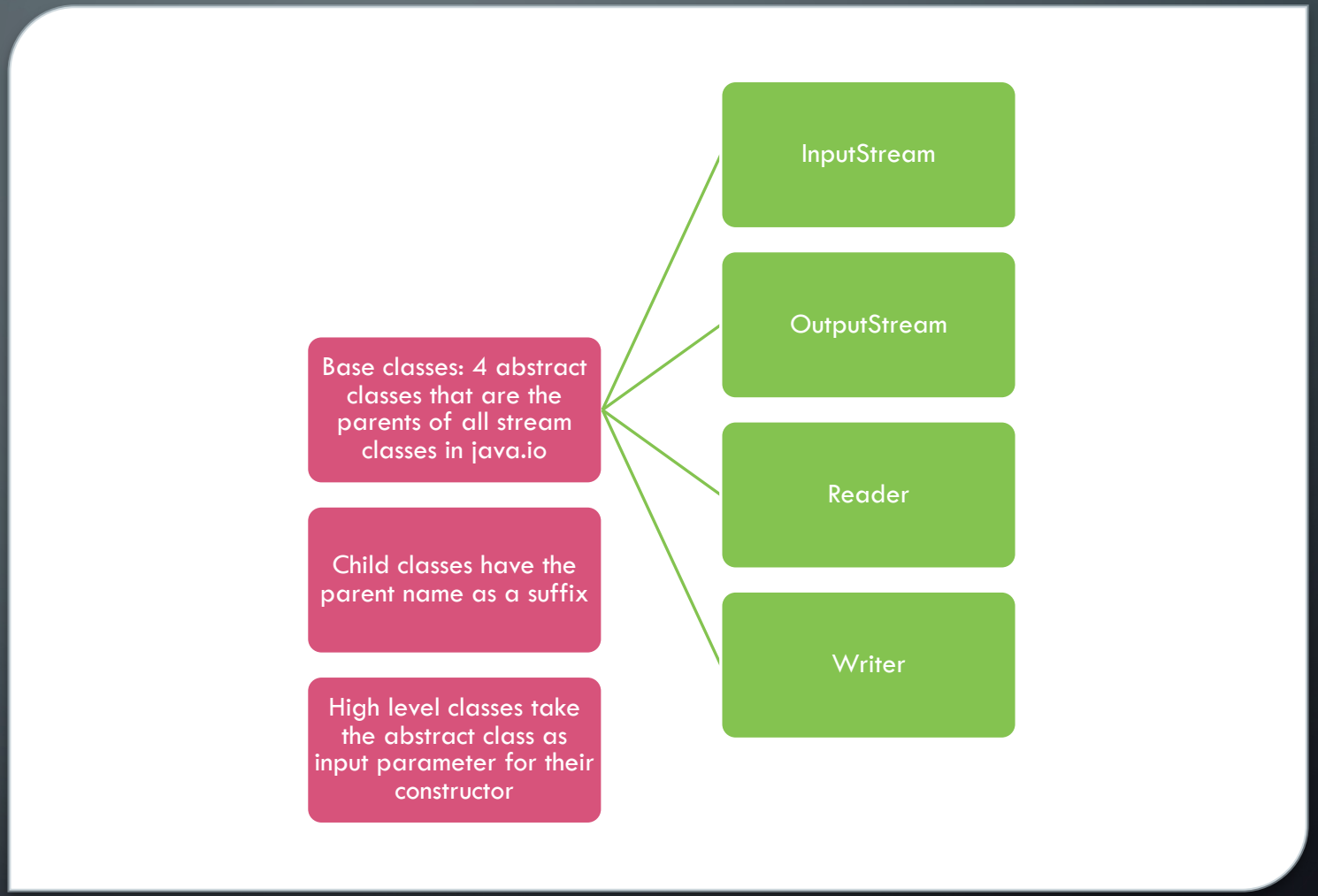
Low level: directly connects with the source (e.g. file, array, string)

High level: built on top of another stream; constructor of this high level stream takes a low level stream

E.g.: new BufferedWriter(new FileWriter("messages.txt")) > could be any subclass of abstract class Writer as input

Best practice: use Buffered class when you don't need to do something specific that requires you to read a byte at a time, it increases performance

JAVA.IO STREAM CLASSES



EXERCISE

- Add some text to your file messages.txt
- Read this file and write the content to log.txt using the most appropriate stream class

- Try it with: byte and character stream, high level vs low level

- Bonus: reverse the content
- Bonus: how to reverse the content when the file is huge?

STREAM OPERATIONS

Close	close() > used to close a stream, automatically called after try-with-resources
Flush	flush() > writes al data to disk immediately, delays application but avoids data loss if program crashes before cache is written to file system
Mark	mark(int x) > if a stream supports mark, you can call mark with an integer. This integer represents after how many bytes further you can still revert to that point when you call reset() on the stream
Skip	skip(long l) > skips the specified number of bytes
Read	read() and write()

EXERCISE

CLOSE

- Create a new file “close.txt”
- Write to this stream using a try block, make sure the program doesn’t terminate
- Don’t close the stream
- What do you think will happen if you try to open close.txt and modify it manually
- And why?

MARK

- Create an inputstream
- Use the smalles number of initial values to achieve “trololololol” using mark and reset

STREAM CLASSES

- FileInputStream and FileOutputStream
- BufferedInputStream and BufferedOutputStream
- FileReader and FileWriter
- BufferedReader and BufferedWriter
- ObjectInputStream and ObjectOutputStream

EXAMPLE OF EACH STREAM PAIR

Take 15 minutes to prepare a 2 minute explanation + demo of one of the stream pairs of the previous slide.

WHAT STREAM TO USE?

A: You want to read from a file with default character encoding

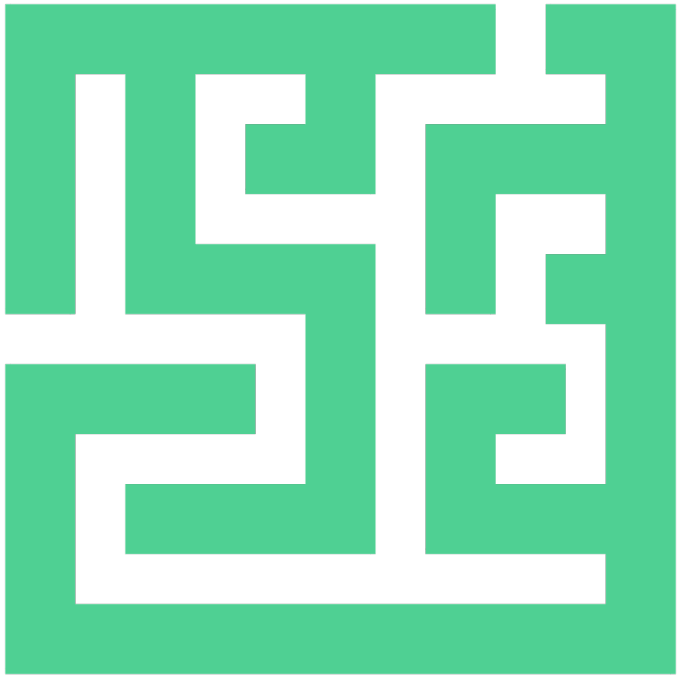
B: You want to read from a inputstream

C: To serialize your custom Car object

D: Read from a file with special character encoding

SERIALIZABLE INTERFACE

- Serializing: converting object to stored data format
- Deserializing: converting stored data to object
- Marker interface
- Class that can be serialized implements serializable interface
- Empty interface, so no methods need to be implemented
- When a class implements serializable it means that the developer has ensured that the class can be serialized, so all instance variables in class and all nested instance variables must also be serializable, unless it's marked transient (then it will be skipped during serializing)
- Static members are ignored during serializing
- Good practice to keep the version in a serialVersionUID of the serializable object, so that deserializing an old data version to the new version can be prohibited to avoid exceptions



QUESTION EXERCISE

If static members are ignored, then why a static version id that is saved serializing the object?

SERIALIZING AND DESERIALIZING OBJECTS

- Can be done with ObjectInputStream and ObjectOutputStream
- ObjectOutputStream has a method writeObject(Obj) to serialize the object
- ObjectInputStream can convert data to an object using readObject; result needs to be casted to desired object
- Exercise: try to serialize an object that doesn't implement the Serializable interface; what happens? Fix it ☺

EXERCISE

- Create a new class Animal, give it some properties, make it serializable
- Create an instance and serialize it and save to a file
- Read the instance and deserialize it

- What exceptions do you need to catch?

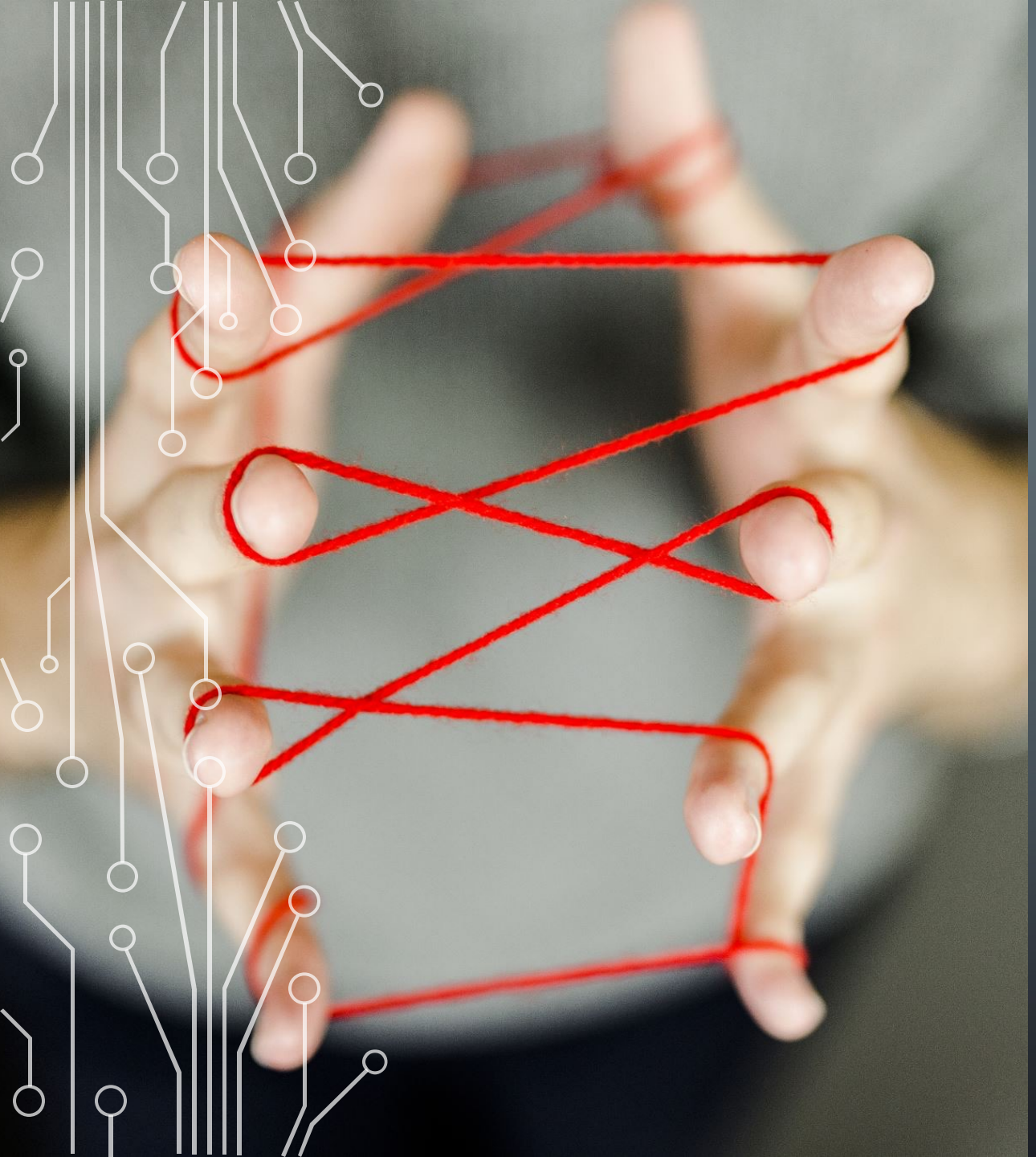
- What happens if you make some variables static? And transient?

OBJECT CREATION

- When object is being deserialized and created, the first no-args constructor in the inheritance line is taken and static members and default initializations are ignored for transient and static members

PRINTSTREAM AND PRINTWRITER

- High level stream classes
- Write representation of Java objects to text output stream
- PrintStream: writes as bytes
- PrintWriter: writes as characters
- No reader/input associated with it



OLD WAY OF USER INTERACTION

- Creating a BufferedReader, with as argument a InputStreamReader, with as argument System.in
- Calling readLine() on the BufferedReader and capturing this in a String so something can be done with the input

NEW WAY OF USER INTERACTION

- Console class (singleton)
- Uses reader/writer instead of inputstream/outputstream

- Has methods for convenience
  - writer() and reader() return access to instance of Reader and PrintWriter
    - Console.writer().println > does the same as System.out.println
  - format() (= printf()) > String format with arguments
  - flush()
  - readLine()
  - readPassword() > user doesn't see text that is being entered on the screen

EXERCISE

- Create a small console application using the new way
- Ask the person for a username
- Ask the person for a password
- Write these to a file.

- You may need to run your application from cmd line for this

- Bonus/alternative: make a much cooler console application, like snake, or tic tac toe, or..? Write the actions of the user to a file.

IO EXERCISES

- If you want extra practice, take these exercises:

https://www.cosc.brocku.ca/mentor/javaio_tutorial

CASE

- Write the transport information to a file, so who or what moved from where to where and when
- Have another method read this file and monitor this output for any products that should have a license but didn't have (a valid) one. Write these to a separate file so the moon police knows what illegal transportations took place
- Make your moon administration serializable
- Ask the user for input on this administration
- Upon ending the console app, write it to a file
- Open it when the user later wants to continue a previous session