Input and output

Introduction

- Data stored in variables is temporary
  - It's lost when a local variable goes out of scope or when the program terminates
- For long-term retention of data, computers use **files.**
- Computers store files on **secondary storage devices**
  - hard disks, optical disks, flash drives and magnetic tapes.
- Data maintained in files is **persistent data** because it exists beyond the duration of program execution.
- https://docs.oracle.com/javase/tutorial/essential/io/streams.html

Java.io

- Most classes to used to deal with writing to or reading from file are part of java.io package or subpackages (need import statement)

- When dealing with files, programmer has no control about some issues that may arise (e.g. connection problem), hence many methods form java.io classes throw ***checked*** exceptions that needs to be caught.

Byte-based streams

- Byte-based streams read from or write to file one byte at a time
  - FileInputStream
  - FileOutputStream

  - https://docs.oracle.com/javase/tutorial/essential/io/bytestreams.html

Character-based streams

- Character-based streams read from or write to file one byte at a time
  - FileReader
  - FileWriter

- To be able to read or Write by other units than a character use a BufferedReader and PrintWriter

https://docs.oracle.com/javase/tutorial/essential/io/charstreams.html

“human-readable” data

- The Scanner class in java.util allows to separate input into tokens that can be processed according to data-type.
- A Scanner object can be associated with an input stream:

```
Scanner s = new Scanner(new BufferedReader(new
FileReader(filename)));
```

Formatted output

- Several classes including PrintStream (datatype of System.out), PrintWriter, String include a **format** method that can be used to format output.
- Examples:

```
System.out.format("The square root of %d is %.4f.%n", i,
r);


String s = String.format("The square root of %d is
%.4f.%n", i, r);
```

Format string syntax:

https://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html#syntax

Inputting and outputting objects

- Two classes, **ObjectInputstream** and **ObjectOutputStream**, allow to read an write complete objects to a file using **readObjet** and **writeObject** methods.
- Restriction: only objects that are instance of a class implementing the **Serializable** interface can be written or read.
- We say that writeObject serializes that object and readObject deserializes the object.
- readObject may throw a ClassNotFoundException if the returned object isnot of the expected class.

https://docs.oracle.com/javase/tutorial/essential/io/objectstreams.html

Serializable interface

- The **Serializable** interface is a “tagging” interface: it contains no method.
- In a class that implements Serializable, every variable must be Serializable.
- Any one that is not must be declared **transient** so it will be ignored during the serialization process.
- All primitive-type variables are serializable.
- For object-type variables, check the class’s documentation (and possibly its superclasses) to ensure that the type is Serializable.