I/O

Serialization

Serialization and De-serialization

- serialization is the process of saving in-memory Java object in the physical file
- de-serialization is the opposite: reading from file and creating Java object
- to make a class serializable:
  - it must implement the *marker interface* `Serializable`

    (marker interface is an interface which has no methods)
- when serializing the object, **only instance** members are serialized (not static)

serialVersionUID

- a special field in serializable classes which is serialized even though it's static:

```
    private static final long serialVersionUID = 1L;
```

- this field serves as an unique identifier for each class in (de)serialization process
- during deserialization JVM checks if the `serialVersionUID` of the loaded class is the same as the `serialVersionUID` of the serialized object
    - if they match, it means that the two versions of the class are compatible
    - if they don't not match, the JVM throws an `InvalidClassException`

Transient Fields

- if you don't want a field to be serialized, you can mark it as **transient**, e.g.

```
private transient String myPassword;
```

- when being deserialized, transient field will revert to it's default Java value

  (`null` for `String`, 0 for `int`, `false` for `boolean`, etc.)

- if you have instance variables in your serializable class, make sure that these objects are also marked as serializable, e.g.
  - if you want to serialize class `Student` which has an instance variable of type `Address`, you have to make `Address` class serializable as well

- **remember: only non-transient instance members will be serialized!**

Serialization Tools

- in order to perform serialization you have to use this classes:
  - `ObjectOutputStream` and `ObjectInputStream`
- these classes are high-level classes and they usually wrap lower-level classes:
  - `FileOutputStream` and `FileInputStream`
- usually we start with a file stream
  - then we wrap it in a buffered stream to improve performance
  - and then wrap the buffered stream with an object stream to access serialization/deserializaton methods

```
-> SerializationExample.java
```