Saving objects

- Almost any non-trivial application will need to save its state (text user has written, or game current situation, created playlists, etc.) permanently. In a Java application the state we'll want to save is described by all the instance variables of all the relevant objects in our application. There are two main approaches:

  1. Save the objects (values of instance variables for all objects). The values can be saved into a file or into a database. If saved into a file the values are not human-readable.

  2. Save the instance variable values as text in a format you specify. Options are custom-designed text format, or JSON, or XML, or other more standardized format.

  3. Use Hibernate or similar to do ORM (object-relation mapping) to save object states into a data base.

- We will take a look at option number 1.

Serializable

- For the instances of your classes to be automatically serializable, your classes will need to implement interface `Serializable`
  - `Serializable` does not require you to implement any methods, by saying you implement it you are only telling JVM that it is ok to serialize instances of the class
- Default behavior for serialization is that all instance variable values (if they are primitive) are saved, and all objects referred to are serialized. You can omit saving an instance variable value (or reference) by declaring it `transient`

Serializable

- When you declare that a class implements `Serializable`, Eclipse (and possibly other IDEs, too) will suggest you to declare `serialVersionUID`
  - this id can be used for distinguishing between different versions of your class - consider a case where you have written serialized version of an instance of a class to a file, then changed the class definition (added instance variables etc.)
  - take a look at `Serializable` documentation if you need to do something with your "older" objects
- Suggested way to handle `serialVersionUID` is that you declare it yourself, and increment the value when the class changes in a non-backwards compatible way
- (this id will get an automatic value if you don't give it a value, but the way this happens is not always what you want)

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

Streams

- Classes `ObjectInputStream` and `ObjectOutputStream` have methods with which it is possible to read & write objects:
  - `obout.writeObject(dogs);`
  - `dogs = (DogTreeSet)obin.readObject();`
- But… these streams must be chained to a connection stream (`FileOutputStream` and `FileInputStream`):

```
FileInputStream in = new FileInputStream("dogsmain.ser");
ObjectInputStream obin = new ObjectInputStream(in);

FileOutputStream out = new FileOutputStream("dogsmain.ser");
ObjectOutputStream obout = new ObjectOutputStream(out);
```

- Note: these steps may throw exceptions - try - catch -block is needed