CSC207 - Managing Objects

Ilir Dema

Summer 2016

Managing Objects

- Often we need to maintain objects (in memory) in our application so we can easily store/retrieve then using a single piece of information, i.e. an ID.
- Also when the application terminates, we have two options:
  - We have done some kind of computation, may be displayed/printed results, and next time we do a fresh start.
  - Store the results of the computation so they can be used next time when the application runs.
- Both topics described above are clearly related.
- Q: How do we make the data persist between runs of an application?
- Q: How do we organize the objects so we can easily access them?

StudentManager

- Say we have defined a class `Student` where every instance does have a property `ID` which is unique.
- We need the capability to read (previously entered!) `Student` objects and store them into a `Map`.
- Once we have completed processing of the existing `Student` objects and may added a few more, we want to persist `Student` objects from the `Map` into a file.

Launching the application

- `StudentManager` class is *responsible* for:
  - Reading the data from a `csv` file
  - Constructing `Student` objects based on data from the `scv` file and populating a `Map` with those `Student` objects.

Before the application terminates

- `StudentManager` class is *responsible* for:
  - Writing `Student` data to a file.
- Q: How do we represent the data to be written to a file?
- If a CSV file is used, then the next time the application is launched, we would need to parse the file and reconstruct the student objects by calling on the Student constructor and passing in arguments.

Serializable Data

- Rather than writing the values of an objects instance variables to file, a representation of the object itself can be written to file.
- An object is *serializable* if it can be be represented as a sequence of bytes.
- The *serialized* object can be written to file.
- The object can later be deserialized. That is, the object can be reconstructed using the data read from the file.

Java's `Serializable` interface

- Java provides interface `Serializable` to serialize objects.
- In order for a class to be serializable, it and its ancestor(s) must implement the `Serializable` interface and every instance variable in the class must also be `Serializable`.
- All of Javas primitive types are serializable.

There are no methods to be implemented!

- In other words, for your class to be `Serializable` you simply need to say implements `Serializable`.
- How can Java write/read instances of any class?
- What could the write/read method look like??

StudentManager

Responsibilities (revised):

- For the first launch of the application, reading Student information from a CSV file, constructing Student objects, and populating a Map.
- Reading serialized Student objects from a file to a Map.
- Maintaining a Map of student id to Student objects when the app is running.
- Writing serialized Student objects from a Map to a file.

Logging

- Logging is the process of recording events that occur during execution of a program in a central location.
- Messages may be written to a log file or to another location such as the standard error stream, System.err

Aside: `stdin`, `stdout`, and `stderr`

- These are streams that connect a program to its environment.
  - stdin: the standard input stream (defaults to the keyboard)
  - stdout: the standard output stream (defaults to the terminal)
  - stderr: the standard error stream (defaults to the terminal)
- We can override the defaults and redirect these streams to something different.
- This is the case for unix commands, and for Java programs (and Python programs, etc.)

java.util.logging.Logger

- `java.util.Logger` provides logging capabilities.
- When we generate a log message, we give it a level of severity:

  SEVERE (highest)
  WARNING
  INFO
  CONFIG
  FINE
  FINER
  FINEST (lowest)

- By default, only messages with level INFO or higher are shown. But we can control that threshold using setLevel.

java.util.Handler

- A handler receives messages from the logger and either writes them to file or the console, or passes them on to be handled elsewhere.
- In Java, Handler classes include:
  - ConsoleHandler (sends log messages to stderr)
  - FileHandler (sends log messages to a file).