CSC301

Serialization II, Persistence & DAO

Team Project

- First deliverable due this week
  - Proofread your team members' work
- Any team *in the evening section* looking for extra members?
  - We have a few students who are stuck without a team (a team that started small, then a couple of members dropped the course).
- The next phase of the project involves coding, not just planning
  - *Everybody* codes! If you do not commit code, you will receive a 0 for the second team deliverable (regardless of your team's mark)
  - Commit your code frequently, in small chunks
  - Make sure you understand the team's Git workflow

Before we wrap up the discussion on serialization, let’s take a detailed look at a couple of related challenges ...

1. Serialize instances of the following class

```
public class Person{
      int age;
      // ...
}
```

2. Source code changes

```
public class Person{
      LocalDateTime birthday;
      // ...
}
```

3. How do we deserialize instances that were serialized in step 1?

Serialization, Versioning

- Supporting versioning is difficult
  - Java Serializable simply fails on version mismatch
  - Newer frameworks support schema evolution
    - Additional artifact(s) to describe the schema
      - Class name, field names, types, etc.
    - Resolution rules to handle version mismatch
  - Sometimes impossible, as some changes are backwards incompatible

Serialization, Object References

Consider a hypothetical `IUser` interface

```
public interface IUser{
     String getName();
     Iterator<IPost> getPosts();
     // ...
}
```

Now, say we serialize an `IUser` instance ...

Serialization, Object References

- When deserializing an `IUser`
  - We expect `getPosts` to return an iterator that visits all of the user's posts.
  - Therefore, when serializing an `IUser`, we need to serialize all of its `IPosts`.
- Therefore, we need to serialize/deserialize a network of objects.

Serialization, Object References

- Garbage Collection makes things challenging
  - Java object references ≠ C pointers
  - Compaction results in objects being moved around in memory. Asynchronously to your program!
  - Cannot identify an object based on “its address”
  - True for every memory-managed language
- Idea: Replace object references with some identification during serialization/deserialization

From Serialization To Persistence

- Once we decide on a serialization method, how do we persist serialized data?
- Simple approach - Write to a file
  - Serialize all of our program’s objects
  - Write them, one by one, to a flat file

Flat-File Persistence

- Good for simple cases
  - Very small files that change infrequently
  - Ex: Settings or configuration files
- But limited
  - Impractical for a large set of objects
    Ex: Facebook’s data cannot fit in a single file
  - Search/update require going through the whole file

Additional Limitations

- Concurrent writes may corrupt the data
- Potential data loss
  - Need a backup
  - Hard to guarantee no data loss, even with backup

Persistent Data Store

- Persistence is challenging when the data
  - is large
  - is accessed by many clients simultaneously
  - changes frequently
  - needs to be queried efficiently
  - needs be highly-available (i.e. no down time, ever)
- There are many persistent data store solutions, of many shapes and flavours.

The *Data Access Object* Design Pattern

- *Data Access Object*
  - Define data-access methods in an interface
  - Implement the DAO interface for different data stores
  - Code is written against the interface, without knowing/caring about the underlying data store.
- DAO is a design pattern for abstracting the details of an underlying data store
  - Allows us to write database-agnostic applications

Data Access Object

Allows your application code to look like

```
public static void example(ITweetEngine dao){
     Iterator<ITweet> itr = dao.getTrendingTweets();
     while(itr.hasNext()){
          System.out.println(itr.next());
     }
}
```

- Application code is not responsible for persisting data
- Application code does not depend on a specific data store
- DAO methods use domain language (e.g. trending tweets)

Data Access Object

- Can be extremely useful when starting to work on a project:
  - Many applications (especially on the web) rely on a database / data store.
  - When you start working on a project, you might want to avoid
    - Choosing (i.e. committing to) data store(s)
    - Integrating the data store into your application
  - A simple solution is to define the DAO, and create a simple in-memory implementation
    - The in-memory implementation is not persistent and does not scale, but it allows us to get started quickly.
    - Once the basic structure and design of your application is ready, you can replace your in-memory implementation with a more realistic one (e.g. backed by a relational database)