Serialization

● Last week we discussed *serialization*
  ○ Converting between objects (in memory) and data (that can be written to persistent storage)
  ○ At the heart of many modern software systems
    ■ Serialize the state of an application and send/receive it over the network

● We saw the solutions that evolved over time
  ○ Built-in, language specific, binary serialization
  ○ Text-based, language agnostic serialization formats like XML and JSON
  ○ Cross-language, high-performance, binary serialization frameworks
Individual Assignments

- A2 auto-marker results have been pushed to your repo
  - A new branch, called auto-marker, in your original A2 repo (not your fork)
  - Please double check the results and let me know if there is a mistake

- A3 is out
  - Start early!
  - Finish the assignment (at least) 12 hours before the deadline.
    - There are no exceptions with auto-marked assignments, don’t leave the submission to the last minute.
  - Do NOT change the interfaces or testing code!
    - These are the specifications.
  - A3 is not huge, but it’s still heavier than the previous assignments
Individual Assignments

● A3 is about serialization
  ○ First, implement the `IGrid<T>` data type
    ■ Maps keys (of type `GridCell`) to values (either objects of type `T` or `null`)
    ■ We use an `IGrid<Rack>` to represent the floor plan of a warehouse.
  ○ Then create two grid implementations
    ■ One supports only rectangular shaped grids
    ■ The other supports any shape
  ○ Then, implement two serializer/deserializers
    ■ Each implementation uses a different text format
    ■ Important: Both implementations serialize/deserialize `IGrid<T>` instances
      (i.e. They are NOT limited to a specific implementation)
  ○ With this design we can convert between files in different formats
    ■ Deserialize `.rect.txt` file to `IGrid<Rack>` instance, then serialize to `.flex.txt` file
    ■ Similar to the way one might convert between Word documents and PDF files.
Before we wrap up the discussion on serialization, let’s take a detailed look at a couple of related challenges ...
Serialization, Versioning

1. Serialize instances of the following class

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

2. Source code changes

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

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

- Supporting versioning is difficult
  - Java’s built-in serialization can handle some changes (but only some)
  - 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

```java
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
Serialization, Object References

- Possible solution: Serialize objects and relations separately.
  - For example, serialize users and posts separately from the *is-author* relation between users and their posts.
  - In some domains, we might have to follow relations/links recursively.

- If objects have unique id’s, we can use them.
  Otherwise, make up ID’s during serialization/deserialization.

- Need to handle cycles!

- Might need to limit how much of the object graph we actually serialize
  - Can use a lazy-loading for deserialization (i.e. some parts of the graph will be loaded to memory, only if we actually try to access them)
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

```java
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)