DataUtils

DataUtils is a class designed to load and save all data from or to a file (e.g. PlayerIO is based on it) and is frankly just a wrapper for Google's gson project, adding some Minecraft-related type adapters.

Simple setup

Loading

Saving

In the most simple way you save your object by just calling `DataUtils.saveData(Object obj, Type objType, String filename)`.
ObjType has to be the type of the object to save. If your object has no generics you can simply use your objects class, otherwise I recommend using `new TypeToken<your object>(){}.getType()`. The filename is a string that will be handed over to a `java.io.File` constructor. If the filename has no extension `.json` will be used as default.

Example:

```
List<String> list = new ArrayList<>();
Type listType = new TypeToken<List<String>>(){}.getType();

// DataUtils.saveData(list, List.class, "testfile"); // WRONG - only use for objects without generics

DataUtils.saveData(list, listType, "testfile"); // CORRECT - will be saved to ./testfile.json
```

Adding your own TypeAdapters

If you want to save a class that can't be serialized (e.g. abstract classes) like a normal object (just fields) you have to write a TypeAdapter for it. You should inform yourself how to do that, here's a simple TypeAdapter:

```
public class UUIDTypeAdapter extends TypeAdapter<UUID> {
  @Override
```

```
    public void write(JsonWriter out, UUID value) throws IOException {
      out.value(value.toString() + " (" + Utils.orDefault(PlayerUtils.getName(value), "unknown") + ')');
    }

    @Override
    public UUID read(JsonReader in) throws IOException {
      return UUID.fromString(in.nextString().replaceFirst(" \\(.+\\)",
"").replaceFirst("(\\w{8})(\\w{4})(\\w{4})(\\w{4})(\\w{12})", "$1-$2-$3-$4-$5"));
    }
  }
```

To use this TypeAdapter in conjunction with DataUtils you can use `DataUtils.with(Collections.singletonMap(UUID.class, new UUIDTypeAdapter())`. This returns you a DataUtils instance with additionally registered UUIDTypeAdapter. `DataUtils.with` acctepts a Map with the Type as key and either TypeAdapter or TypeAdapterFactory (Also all other type adapters gson is supporting).

> For easy mapping of multiple type adapters you can use Utils.map(key, value, key, value, ...)

Revision #5
Created 5 July 2017 08:57:04 by deregges
Updated 17 October 2017 09:19:57 by deregges