



## LONG-TERM STORAGE + DOCUMENTING YOUR CODE

DR. ISAAC GRIFFITH

IDAHO STATE UNIVERSITY



How do we reuse objects  
between executions of our program?

# Outcomes



After today's lecture you will be able to:

- Understand the basics of persisting data using
  - Java Serialization
  - JSON via Gson
- Understand and use JavaDoc to document your code following good practices
- Understand basics of source code licensing



# ⌘ Storing Objects

---

CS 2263



- Most applications will have some need to persist data between executions
- Additionally, data that is needed may be significantly larger than that which can be stored in main memory
- Thus, we need some mechanism to store and retrieve this data
- There are several options, but we will discuss the following
  - Java ObjectStreams
  - Java Serialization
  - Google Gson

- We can easily store and retrieve primitive data
- For this we will use a `FileOutputStream` (or `FileInputStream`) parameterized with an `ObjectOutputStream` (or `ObjectInputStream`)
- Example: **Writing Primitive Data**

## 1. Construct the output stream

```
FileOutputStream file = new FileOutputStream("file");  
ObjectOutputStream out = new ObjectOutputStream(file);
```

## 2. Store the data

```
int i = 7;  
char c = 'q';  
boolean b = true;  
double d = 3.14;  
out.writeInt(i);  
out.writeChar(c);  
out.writeBoolean(b);  
out.writeDouble(d);  
out.close();
```

- We can easily store and retrieve primitive data
- For this we will use a `ObjectOutputStream` (or `ObjectInputStream`) parameterized with an `FileOutputStream` (or `FileInputStream`)
- Example: **Reading Primitive Data**

## 1. Construct the input stream

```
FileInputStream file = new FileInputStream("file");  
ObjectInputStream in = new ObjectInputStream(file);
```

## 2. Read in the data

```
int i = input.readInt();  
char c = input.readChar();  
boolean b = input.readBoolean();  
double d = input.readDouble();
```

# Using Serialization

---

CS 2263



- To store objects we would expect to follow a similar procedure:

```
Television television = new Television();  
Account account = new Account();  
FileOutputStream file = new FileOutputStream("objdata");  
ObjectOutputStream out = new ObjectOutputStream(file);  
out.writeObject(television);  
out.writeObject(account);
```

- Unfortunately, it is not quite that simple

# Retrieving Objects



- And again, we would expect to retrieve objects like so:

```
Television television;  
Account account;  
FileInputStream file = new FileInputStream("objdata");  
ObjectInputStream out = new ObjectInputStream(file);  
television = input.readObject();  
account = input.readObject();
```

- Unfortunately, it is not quite that simple

- Unfortunately, due to the way objects are composed of other objects, there will be several issues we need to consider
  - **Reconstruction** - depending on how the object was stored, we will need information on how to retrieve it
  - **Complexity** - The more complex the object (more other objects it is composed of) the more objects that must also be read in to construct the original object.
- Because of these issues the creators of Java created **serialization**
  - Provides efficient reconstruction of objects
  - Provides correct storage of complex objects

- The process of Serialization is as follows:
  1. Each class to be serialized must implement the `java.io.Serializable` interface
  2. Need to open an `ObjectOutputStream` parameterized by a `FileOutputStream`
  3. Use the `writeObject(Object)` method from `ObjectOutputStream`

- The process of Deserialization is as follows:
  1. Open the file using an `ObjectInputStream` parameterized with a `FileInputStream`
  2. Use the `readObject()` method from `ObjectInputStream` to read the appropriate type
  3. Objects are to be read back in the order in which they were written
- Note: There is an issue. Every time a class changes, it can no longer deserialize data from prior versions of the class.

# Using Gson

---

CS 2263

- A language independent lightweight data-interchange format
- Designed to be:
  - Easy for humans to read and write
  - Easy for machines to parse and generate
- Built on two structures:
  - Collection of name/value pairs
    - i.e., object, record, dictionary, etc.
  - Ordered list of values
    - i.e., array, list, sequence, etc.

## Example

```
{  
  "menu": {  
    "id": "file",  
    "value": "File",  
    "popup": {  
      "menuitem": [  
        {"value": "New", "onclick": "CreateNewDoc()"},  
        {"value": "Open", "onclick": "OpenDoc()"},  
        {"value": "Close", "onclick": "CloseDoc()"}  
      ]  
    }  
  }  
}
```

- Library which:
  - Converts Java Objects into a JSON representation
  - Converts JSON Strings into Java Objects
  - Overcomes the issue of deserialization
- To use Gson with gradle, you need only add the following dependency:

```
dependencies {  
    implementation 'com.google.code.gson:gson:2.8.6'  
}
```

- **Issues**
  - You cannot serialize objects with circular references
    - results in infinite recursion
  - Classes to be serialized require a no-args constructor in order to be serialized



# Storing an Objects



## 1. Create the Gson object

```
Gson gson = new Gson();
```

## 2. Convert the object to a JSON String

```
class SomeObject {  
    private int value1 = 1;  
    private String value2 = "abc";  
  
    SomeObject() {} // no-args constructor req'd  
}  
  
SomeObject obj = new SomeObject();  
String json = gson.toJson(obj);
```

## 3. Write out the JSON string to a file



# Retrieve an Object

1. Read the json data from a file into a string
2. Create the Gson object

```
Gson gson = new Gson();
```

3. Convert theString into the object

```
String json; // json string  
SomeObject obj2 = gson.fromJson(json, SomeObject.class);
```

# § Documenting Your Code

---

CS 2263

- JavaDoc provides you a means to communicate the purpose and use of your code to yourself and others
- Towards this end, Javadoc provides the following tags:

Tag	Description	Syntax
@author	Adds the author of a class	@author name-text
{@inheritDoc}	Inherits a comment from the <b>nearest</b> inheritable class or implementable interface	
@version	Adds a “version” subheading with the specified version-text to the generated docs when the -version option is used	@version version-text
@param	Adds a parameter with the specified parameter-name followed by the specified description to the “Parameters” section	@param parameter-name description
@return	Adds a “Returns” section with the description text	@return description

- JavaDoc provides you a means to communicate the purpose and use of your code to yourself and others
- Towards this end, Javadoc provides the following tags:

Tag	Description	Syntax
<code>@exception</code>	Adds a <b>Throws</b> subheading to the generated documentation, with the classname and description text	<code>@exception class-name description</code>
<code>@throws</code>	synonym for <code>@exception</code>	
<code>@see</code>	Adds a “See Also” heading with a link or text entry that points to reference	<code>@see reference</code>
<code>@since</code>	Adds a “Since” heading with the specified since-text to the generated documentation	<code>@since release</code>
<code>@deprecated</code>	Adds a comment indicating that this API should no longer be used	<code>@deprecated deprecatedtext</code>

- JavaDoc provides you a means to communicate the purpose and use of your code to yourself and others
- Towards this end, Javadoc provides the following tags:

Tag	Description	Syntax
<code>@link</code>	Inserts an in-line link that the visible text label that points to the documentation for the specified package, class, or member name of a referenced class.	<code>{@link package.class#member label}</code>
<code>@linkplain</code>	Identical to <code>{@link}</code> , except the link's label is displayed in plain text rather than code font.	<code>{@linkplain package.class#member label}</code>
<code>{@code}</code>	Displays text in code font without interpreting the text as HTML markup or nested javadoc tags	<code>{@code text}</code>
<code>{@value}</code>	When <code>{@value}</code> is used in the doc comment of a static field, it displays the value of that constant	<code>{@value package.class#field}</code>

- Write Javadoc to be read as source code
- All public/protected methods should be fully defined with Javadoc
  - If a method is overridden use `@Override` to indicate that the Javadoc is inherited
- Use the standard style for javadoc comment

```
/**  
 * Standard comment  
 */
```

- Use “this” to refer to an instance of the class
- Aim for short single line sentences

- Do not use `@code` for `null`, `true`, or `false`
- Use `@param` for generics
- Use one blank line before `@param`
- Treat `@param` and `@return` as a phrase
- Treat `@throws` as an if clause
- `@param` should have two spaces after param name



- Define null-handling for all parameters and return types
- Include implementation notes **Example**
- Use a single `<p>` tag between paragraphs
- Use a single `<li>` tag for items in a list

The following are several tips/best practices for documenting your code with JavaDoc

1. Use `@link` and `@linkplain` for point to some code
2. Use `@code` for Code Snippets
3. Use `@value` to insert the value of a field in the documentation
4. Indicate when features have been available with `@since`
5. Don't be anonymous, use `@author`
6. For non-void methods, always use `@return`
7. Clarify what parameters mean with `@param`
8. Use DocCheck to your advantage

# Generating JavaDoc



- You can utilize Gradle to automate generating JavaDoc
- This is part of the `java` plugin
- In your build file add the following (assuming standard directory structure)

## `build.gradle`

```
plugins {  
    id 'java' // not needed if application  
}  
  
javadoc {  
    source = sourceSets.main.allJava  
}
```

- Generate using the following command

```
$ gradle javadoc
```

- This will produce the JavaDoc in the following location  
`project-root/build/docs/javadoc`

- GitHub provides the capability of adding a static site to a project or organization in order to allow you to have a landing page to showcase what you have done.
- These pages use markdown and jekyll to generate html.
- You can configure your project (in the settings) to utilize the `docs/` folder to serve these pages.
- Thus, if you provided an index with a link to `javadoc/index.html` you could copy your generated javadoc to this location as part of the build.
- With this in place you could then provide api documentation automatically.

## build.gradle

```
javadoc {  
    source = sourceSets.main.allJava  
    destinationDir = file("${rootDir}/docs/javadoc")  
}
```

# ⌘ Licensing Your Code

---

CS 2263

- Programming and software development are an exercise in creativity.
- As such, they are a form of property known as Intellectual Property.
- Thus, it needs protection
- Code can be protected under copyright law.
- In addition to copyright, you should protect your code and yourself by stating the terms of use via a license.
- All companies license their software under either a conventional closed source or an open source license.
  - The latter, allows for the users to not only have free use of the software, but also rights to use the source code (according to the constraints of the license)
- There are many variations of these licenses
  - For open source license information: <https://opensource.org/licenses>

- Popular Open Source Licenses are:
  - Apache License 2.0
  - BSD 3-Clause “New” or “Revise” license
  - BSD 2-Clause “Simplified” or “FreeBSD” license
  - GNU General Public License (GPL)
  - GNU Library or “Lesser” General Public License (LGPL)
  - MIT License (my favorite)
  - Mozilla Public License 2.0
  - Common Development and Distribution License
  - Eclipse Public License version 2.0
- [choosealicense.com](https://choosealicense.com) is an extremely helpful site that is designed to help you select an open source license that is right for you.
  - They also note what happens if you choose not to license your code.

- Regardless of which license you select, you need to add a LICENSE file to the root directory of your project (next to the README.md and CHANGELOG.md files)
- Additionally, it is considered good practice to insert (as a comment) the license header at the top of each of your source code files.
  - As this is a tedious process, there is a gradle plugin that will help you deal with this issue
  - The Gradle License Plugin, which uses your LICENSE file and adds its contents to each code file in your project.

## build.gradle

```
plugins {  
    id "com.github.hierynomus.license-base" version "0.16.1"  
}  
  
license {  
    header = project.file('LICENSE')  
}
```



- This adds several tasks to the build lifecycle
  - `licenseMain(LicenseCheck)`: checks header consistency in the main source set
  - `licenseFormatMain(LicenseFormat)`: applies the license found in the header file in files missing the header
  - `licenseTest(LicenseCheck)`: checks header consistency in the test source set
  - `licenseFormatTest(LicenseFormat)`: applies the license found in the header file in files missing the header in the test source set
- The `licenseCheck` tasks are executed during the build process in the `check` phase
  - will fail the build if it detects files that are not consistent or missing the header
- You can run the format using the following command:

```
$ gradle licenseFormat
```

from the root directory of your project

# Resources

- [JSON](#)
- [Gson User Guide](#)

# For Next Time



- Review Chapter 4.6
- Review the Gson Tutorial
- Review the JavaDoc Tutorial
- Review this Lecture
- Come to class
- Read Getting Started with JavaFX
- Read the JavaFX Tutorial
- Continue working on Homework 03





# Are there any questions?