Australian National University

# Persistent Data

Lab

## Task for this week

- Implement methods to save and load data to/from different file formats.

- The code structure/skeleton is available on Wattle
- This lab contains assessable items!

- Submission Guidelines
  - The last slide contains information about the submission
  - Read it carefully to avoid losing marks!

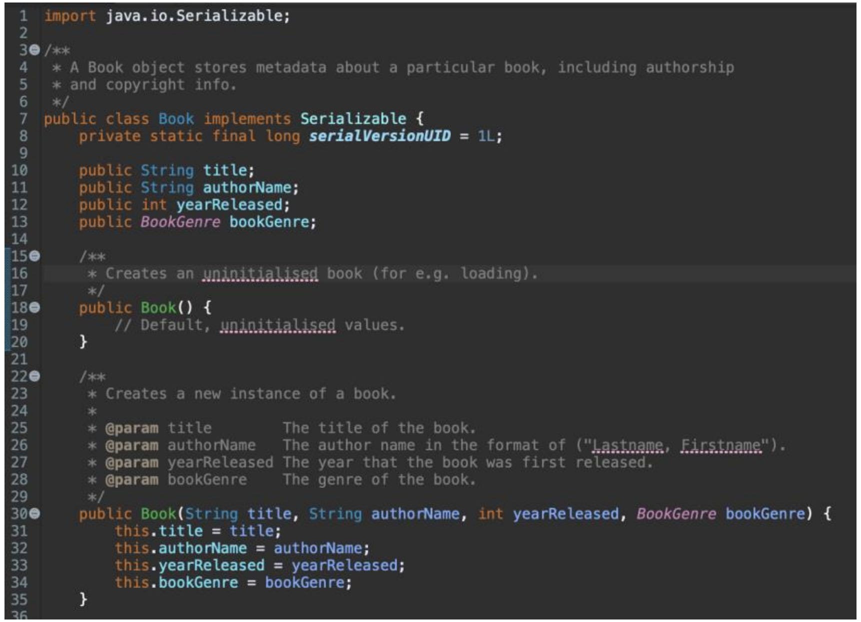
# Task 1 – Storing data (1 mark)

The main objective of this task is to understand how to store data in different file formats: Bespoke, JSON and XML. The partial **code is available on Wattle/Repo.**

Step 1) Go through and read the file **`Book.java`**. It is just a simple class to store information about books.

This class has two different constructors: one that does not initialize the book's attributes and another that creates and initializes a new instance of book.

Note that the visibility of the book's attributes is set to public and this class implements Serializable.



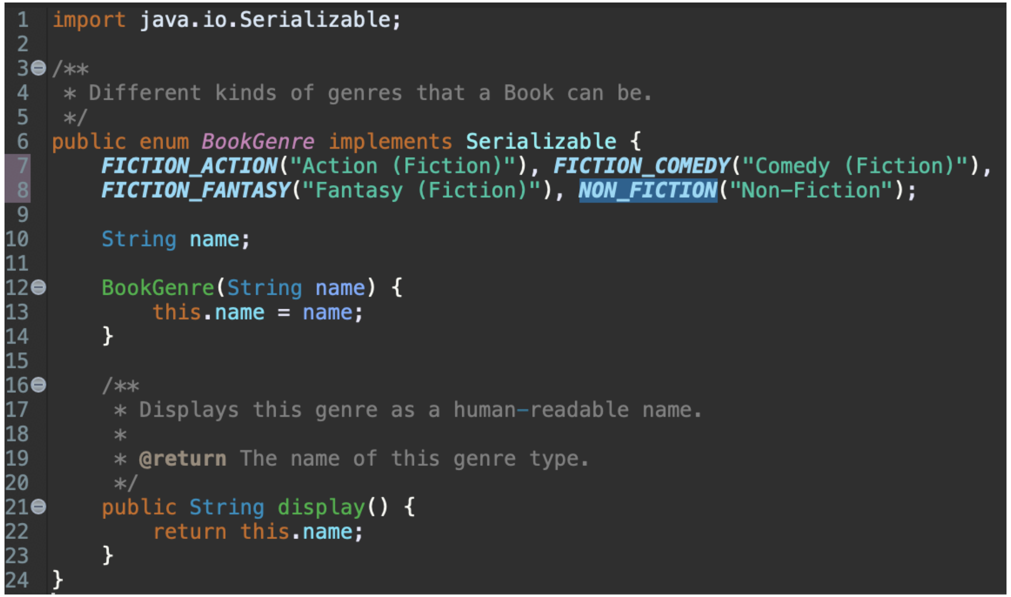
```java
import java.io.Serializable;

/**
 * A Book object stores metadata about a particular book, including authorship
 * and copyright info.
 */
public class Book implements Serializable {
    private static final long serialVersionUID = 1L;

    public String title;
    public String authorName;
    public int yearReleased;
    public BookGenre bookGenre;

    /**
     * Creates an uninitialised book (for e.g. loading).
     */
    public Book() {
        // Default, uninitialised values.
    }

    /**
     * Creates a new instance of a book.
     *
     * @param title        The title of the book.
     * @param authorName   The author name in the format of ("Lastname, Firstname").
     * @param yearReleased The year that the book was first released.
     * @param bookGenre    The genre of the book.
     */
    public Book(String title, String authorName, int yearReleased, BookGenre bookGenre) {
        this.title = title;
        this.authorName = authorName;
        this.yearReleased = yearReleased;
        this.bookGenre = bookGenre;
    }
```

# Task 1 – Storing data

Step 2) Now, read the file **BookGenre.java**.

BookGenre is an Enum Type that has a private constructor that is used to initialize the name of the genre type.

Note that the Enum implements Serializable.



```java
import java.io.Serializable;

/**
 * Different kinds of genres that a Book can be.
 */
public enum BookGenre implements Serializable {
    FICTION_ACTION("Action (Fiction)"), FICTION_COMEDY("Comedy (Fiction)"),
    FICTION_FANTASY("Fantasy (Fiction)"), NON_FICTION("Non-Fiction");

    String name;

    BookGenre(String name) {
        this.name = name;
    }

    /**
     * Displays this genre as a human-readable name.
     *
     * @return The name of this genre type.
     */
    public String display() {
        return this.name;
    }
}
```

## Task 1 – Storing data

Step 3) It is time to read the file `BookCollection.java,` a class that represents a book collection (list of books).

Now, **you must implement the following methods:**

- **saveToBespokeFile**: For this method, you have two options: save the class objects using the serialization method or define your own format (e.g. a CSV file) to save the book collection.

- **saveToJSONFile** (…): For this method, you must save the book collection in JSON format. There are several ways to write JSON documents: during the lecture we used Gson, you can use it here as well. To use Gson, you need to add the Gson jar file to your classpath (Gson library is available on Wattle/Repo).
  More info about Gson can be found at: https://github.com/google/gson/blob/master/UserGuide.md

- **saveToXMLFile** (…): For this method, you must save the book collection using XML. There are several ways to write XML documents. During the lecture, we saw the DOM-based approach. You can use the DOM-based approach, or you may want to use an alternative method.

Please just make sure that your implementation can be compiled using Java 8+ (use standard library). Do not use non-standard libraries, i.e., external libraries that are not explicitly mentioned here).

## Task 2 – Loading data (1 mark)

The main objective of this part is to understand how to **load** data from different file formats: Bespoke, JSON and XML.

In task 1 you defined the format for saving the files, now you must follow the formats you specified to load the information stored in the saved files. Implement the following methods:

- **loadFromBespokeFile** (load a book collection which is stored by saveToBespokeFile method)
- **loadFromJSONFile** (load a book collection which is stored by saveToJSONFile method)
- **loadFromXMLFile** (load a book collection which is stored by saveToXMLFile method)

For both tasks (save/load), you can add ***throw*** statements to each method signature, if necessary.

## How to test your code?

Use the `BookCollectionTest.java` file to increase your confidence that your implementation is correct. Note that the test cases provided do not validate the correctness of your JSON or XML files. You may want to validate the correctness of the file format with additional test cases.

The `BookCollectionTest.java` file has 3 test cases.

To assess your code, we will use 6 different test cases, each is worth 1/6.

## Submission Guidelines

- **Assignment deadline: see the deadline on Wattle (always!)**
- Submission mode: via Wattle (Lab Persistent Data)

Submission format (IMPORTANT):

- Upload **only** your final version of: **`BookCollection.java`** to Wattle
- Each test case must **run for at most 1000ms**, otherwise it will fail (zero marks).
- **Do not** change the file names
- **Do not** upload any other files (only the specified files are needed)
- **Do not** upload a folder (your submission should be only **the specified java file**).
- The answers will be marked by an automated marker.
  - **Do not** change the structure of the source code including class name, package structure, etc.
  - You are only allowed to edit the designated code segment indicated in the comments.
- Any violation of the submission guidelines will result in zero marks
- Reference for this lab: see lecture video and demos