Assignment 2: Water Purification Inventory Management

- See website for due date and late penalty info.
- Submit deliverables to CourSys: https://coursys.sfu.ca
- This assignment is to be done **individually**. Do not show another student your code, do not copy code from another person/online. Do not reuse your previous work (even if retaking the course).
- You may use general ideas you find online and from others, but your solution must be your own.
- See the marking guide for details on how each part will be marked.

Imagine you work at a software development company and your current customer builds, tests, and ships water purification systems ("units") around the world. For this assignment, you are to implement a Java program which allows the user to manage their inventory of units. It must track individual units, and the results of tests done on those units. It must also produce some useful reports.

1. Features

The course website has a capture of some sample output showing how the entire application operates.

1.1 Unit Info

Each water purification system has a:...

- Serial Number: up to 15 numeric digits [0 through 9]; no spaces, letters, punctuation, etc.
  - The last two digits are a checksum computed by add up the preceding digits and mod by 100.
  - Must be at least 3 digits long.
  - Example valid serial numbers:
    000, 101, 202, 505, 532515, 0102030006, 99999999999201
- Model: Up to 10 characters long; no restriction on which characters.

Each unit can be tested 0 or more times. The customer needs to know the following info for each test:

- Date of the test (displayed in the format YYYY-MM-DD)
- If the test passed, or failed.
- Comment about the test's results (for either a failed test, or a passed test).

Each unit can either be in-stock (not shipped), or be shipped-out to some other location. The customer does not need this system to track to where units were shipped: it only tracks the date the unit was shipped (display format YYYY-MM-DD). If it was not shipped, we'll print "-".

1.2 Text UI

Use a menu driven text UI to interact with the user.

- You may assume all inputs are the correct format (such as an `int`) when needed.
- For invalid selections on the menu, print an error and retry.
- Options available on the main menu are described below; see output for details.

Many different tables are outputted to the screen; leave two spaces between adjacent columns to improve readability.

1.3 Load from JSON file

Prompt the user to type in the path of an input JSON file. The program must load this data, replacing any data currently managed by the system. You may assume that the input JSON file is well formatted and contains only valid data.

Constraints on JSON file:

- Will be valid JSON
- Will have correct structure for data, as exemplified by sample file(s).
- Will have valid model, serial numbers, dates, and other fields.
- If a unit has no tests, it will have an empty list of tests (vs just not being present, which would give a null)
- `dateShipped` may or may not be present (i.e., will be non-null, or null).
- All tests will have valid `date`, `isTestPassed`, and `testResultComment` field; all will be non-null.
- No guarantee on the order of units in the file (cannot assume they are in a sorted order).
- Tests for a unit will be in chronological order (earliest test first).
- All dates in format "YYYY-MM-DD"

Note that the GSON library creates objects without using constructors for initialization, so you cannot rely on your constructor to build well formed objects or validate field constraints.

1.4 Display Unit Info

- Prompt the user to type in the serial number of a unit.
  - User may type in 0 to see a list of all units in the system, sorted by serial number.
  - User may type in -1 to cancel.
  - User may type in an invalid serial number to see an error and prompt to try again.
  - User may type in a valid serial number to see a summary of the unit.
- Unit Summary:
  - Show unit's serial number, model, and ship date (show "-" if not shipped).
  - Show details about each of the unit's tests. If there are no tests, display a message stating that.

1.5 Create Unit

- Prompt user to input the model. Just a blank line cancels.
- Prompt user to input the serial number. Just a blank line cancels.
  - If the serial number already exists, display an error message and prompt for new serial number.
  - If the serial number is invalid (checksum fails or too short) display an error message and prompt for a new serial number.
  - NOTE: Your low level class which stores the serial number must enforce the serial number is valid. It must throw an exception if someone tries to set an invalid serial number. The calling code must, at some point, catch such exceptions. Your text UI must rely on the low-level class for doing serial number validation vs repeating the code elsewhere.
- You do not need to enforce maximum lengths on the model and serial number; you may assume the user never exceeds those lengths.

1.6 Test a Unit

- Prompt user to type in a serial number of a unit (same as Display Unit Info option).
- Prompt user to enter if the test passed or not. Entering a blank line assumes the test is a pass. Otherwise user may enter 'y' or 'n', case insensitive for yes or no respectively.
- Prompt user to enter a comment. Comment may use any characters, including spaces. Comment may be empty.
- Record that this unit now has a new test with these values.

1.7 Ship a Unit

- Prompt user to type in a serial number of a unit (same as Display Unit Info option).
- Record that the unit has shipped on today's date. If the unit had previously shipped, update the ship date to today's date.

1.8 Print Report

- Prompt the user to select one of the following reports:
  - **All**: Print report of all units.
    - Display columns: model, serial number, total # tests done on this unit, ship date (YYYY-MM-DD, or "-" if none).
  - **Defective**: Print reports showing all units which have been tested, and whose most recent test was a failure.
    - Display columns: model, serial number, total # tests done on this unit, most recent test date (YYYY-MM-DD), comments recorded about most recent test.
  - **Ready-to-ship**: Print report showing all units which have been tested, whose most recent test was a success, and which have not yet been shipped.
    - Display columns: model, serial number, most recent test date (YYYY-MM-DD).
- For all reports, sort the units by the selected sort order (see next menu option for available sort orders and how to select one)
- For all reports, display a header at the top of the report showing:
  - Name of the report, followed by a row of '*'s
  - Column headers for each column,
  - Underline each column header using '-' characters (showing the columns)
- Ensure column data are well aligned
  - Note that when present, test result comments are always the last column and are left-justified so their length does not matter when displaying the table. You may assume comments are short enough as to not wrap around and disturb the next line.

1.9 Set Sort Order

- Prompt the user, allowing them to select the sort order they want used for future reports:
  - Sort units by serial number (ascending); this is the default. **This is sorted numerically**, not lexicographically. For example, the unit with serial number 909 comes before unit 1001.
  - Sort units by model (ascending), then by serial number (ascending). Model is compared lexicographically (alphabetical); serial number is compared numerically.
  - Sort units by the date of their most recent test (ascending, with oldest on top).
    - Units which have no tests come after units which have tests; ordering among the units which have no tests is unspecified (any order OK).

This menu option does not itself display a report; it simply sets the sort order for future reports the user might later chooses to display. The use can then come back to this menu option to change the selected sort order later when needed.

2. Requirements

- You must use:
  - Model/View separation: create one package to hold the model (the data) classes; create a separate package to hold the view (text UI) classes.
  - Exceptions: you must use Java exceptions to validate the unit serial number.
  - Comparators: you must use Comparators to sort the items in the report.
- Your code must conform to the programming style guide for this course; see course website.
- All classes must have a class-level JavaDoc comment describing the purpose of the class.
- Functions should not be longer than about 30 lines long.
- Lines of code should not be longer than about 120 characters long.
- Code should not be more than 3 control structures deep (ex: `if` in `loop` in an `if`).
- Your classes' `toString()` methods may not be used to generate output for the screen.
- `main()` must instantiate the model, instantiate the view and pass it the model, then call the view's menu/start function.
- Model package must have 3 or more non-trivial classes
- View package must have 2 or more non-trivial classes
- Your main can be placed in either package, or outside of either package. (It is a trivial class).

3. Suggestions

- Think about the design before you start coding.
  - List the classes you expect to create.
  - For each class, decide what its responsibilities will be.
  - Think through some of the required features. How will each of your classes work to implement these features? Can you think of design alternatives?
  - Think about what parts of the code may lead to a lot of duplicate code. How can you reduce the duplication? For example, think about the menus or reports.
- You may use the sample solution for assignment 1 to inspire you for assignment 2. You may not copy-and-paste the code, but you can have a solution which is directly inspired by it! For completeness, please cite the solution if used.
- For all dates, I recommend using Java's `LocalDate` class. For example:
```
LocalDate now = LocalDate.now();
String display = now.format(DateTimeFormatter.ISO_LOCAL_DATE);
```
- Sorting serial numbers numerically is tricky because they are strings, not integers. For sorting, use the `BigInteger` class to take the string and convert it to a `BigInteger`, and then compare the `BigInteger`s using their `compareTo()` method.
- Use a library to read JSON files, such as GSON; see website for resources.
  - Note that a special adapter class is needed to use `LocalDate` objects with Gson; see website for resources.
- Numerous parts of the UI require reading in a line of text. Consider always using `.nextLine()` on a scanner, and then parse the string as needed using `Integer.parseInteger(...);`
- Serial numbers may start with a 0, so don't store it as an `int`.

4. Deliverables

Submit a ZIP file of your project to CourSys. See course website for directions on creating and testing your ZIP file for submission. All submissions will automatically be compared for cheating.

