Assignment 5

Refer to Canvas for assignment due dates for your section.

Objectives:

- Use overloading to support multiple implementations of important functionality, where appropriate.
- Take advantage of subtype polymorphism.
- Continue to meet the objectives from previous assignments where applicable.

General Requirements

Create a new Gradle project for this assignment in your course GitHub repo. Make sure to follow the instructions provided in “Using Gradle with IntelliJ” on Canvas.

Create a separate package for each problem in the assignment. Create all your files in the appropriate package.

To submit your work, push it to GitHub and create a release. Refer to the instructions on Canvas.

Your repository should contain:

- One .java file per Java class.
- One .java file per Java test class.
- UML diagrams for each problem. UML diagrams can be generated using IntelliJ or hand-drawn.
- All non-test classes and non-test methods must have valid Javadoc.

Your repository should **not** contain:

- Any .class files.
- Any .html files.
- Any IntelliJ specific files.

Problem 1

For this problem, you will find it helpful to use Java’s LocalDateTime class for all dates.

Design and implement a system that will enable non-profits to track the donations they receive. All donations have an amount, and the date and time that the donation was made. The system should track the following types of donations:

- **One-time donations**. These donations are for a single amount, made on the provided date.

- **Monthly donations**. These donations repeat once a month until they are cancelled. For example, if a monthly donation of $25 is created on 2/15/2024 at 5:45 PM, $25 will be donated at 5:45 PM on the 15th of every month, starting on 2/15/2024 and continuing until the donation is canceled. Monthly donations should not have a cancellation date/time when they are first created, but there should be a method that allows the cancellation date/time to be set. This method should ensure that the proposed cancellation is not prior to the creation date and time.
- **Pledges**. These are promises to donate a given amount at some point in the future. Pledges are used for special cases such as matching donations (e.g., a donor says they will donate $X if the non-profit is able to raise a certain amount of money by a certain date) or bequests (a donor pledges to leave money to the non-profit when they die). When a pledge is made, the date that the donation will be processed may or may not be provided. For pledges, the donation date should be tracked separately from the creation date that is common to all donation types. It should be possible for the processing date to be changed (or removed), so long as the proposed new processing date is not prior to the creation date and time.

Your system should also include a `NonProfit` class, which will track:

- The organization's name.
- A collection of all donations made to the non-profit.

The `NonProfit` class should include a method, `getTotalDonationsForYear`, that takes one parameter, an integer year, and returns the sum of all donations processed in the specified year. Use the following logic to determine whether a particular donation occurred in a particular year:

- For one-time donations, the amount should only be included in the total for the year that the donation was created.
- For monthly donations, sum all donations that occurred in the given year. For example, if the specified year is 2023, and a monthly donation of $10 is created some time on 3/5/2023, then the 2023 total for that donation will be 10 x $10 as long as the donation does not have a cancellation date, or the cancellation date is in 2024 or beyond.
- For pledges, the amount should only be included in the total for the year that the donation is actually processed. If no processing date is set at the time `getTotalDonationsForYear` is called, the pledged amount should not be included in the total.

As you design your system, keep in mind that additional types of donations may need to be supported in the future. A well-designed object-oriented solution will not require modifications to the `NonProfit` class to account for donation types added in the future.

Problem 2

You are tasked with developing a system that will track a library's collection of books and music and enable that collection to be searched based on various criteria.

Every item in the collection needs to contain the following information:

- The item's creator e.g., author, band, recording artist,
- The item's title,
- The year the item was released or published.

Creators can be:

- An individual person, either an `Author` or a `RecordingArtist`. All individuals should have a first and last name, tracked separately.
- A group. For now, the only type of group the system needs to support is a `Band`. A `Band` should have a name and a collection of `RecordingArtist`s, the `Band`'s members.

Items can be:

- A `Book`, which has an `Author` as its creator.
- `Music`, which can have either a `RecordingArtist` or a `Band` as its creator.

Create a class called `Catalog`, which will be used to store a collection of all the items in the library and to implement search functionality. The `Catalog` should only have a single instance variable: a collection of all items in the library. It should be possible to instantiate a `Catalog` with or without specifying the items in the collection. It should also be possible to add items to and remove items from the collection. The `Catalog` should have three overloaded `search` methods, all of which return the subset of items in the library's collection that match the criteria. The requirements for the `search` methods are as follows:

- `search(String keyword)`. This should retrieve all items in the catalog, regardless of type, that have a title *containing* the keyword. For example, if search is passed the keyword "a", all items with a title that contains "a" should be in the returned collection. The search should be case insensitive. This means that, following on from the previous example, searching for "a" would return items that have either "a" or "A" anywhere in the title.
- `search(Author author)`. This should retrieve all items in the catalog that have an exact match for the given author. As only `Book`s have authors, it is expected that the collection returned will not contain any `Music`.
- `search(RecordingArtist artist)`. This should retrieve all items in the catalog that have an exact match for the given artist. As only `Music` items have `RecordingArtist`s, it is expected that the collection returned will not contain any `Book`s. The artist may be the sole creator of the `Music` or one member of a `Band`.

Please note that in this homework assignment you are allowed to use built-in collections, such as an `ArrayList`.