Anime Track

An application that searches different websites and pulls out info of upcoming animations

# IDE

Eclipse Oxygen. Lookup refactoring,

# Design

### Rethink data structure for storing anime details. Think inserting/retrieving operations and their complexity.

## Design Ideas:

1. Create an AnimeData class that holds all the info on an anime object and store them in a dictionary with the name of an anime as the key to the corresponding object value that has all its info
2. Store as a linked list with each data node that is composed of a record holding all the info on a specific anime. Each data node would hold info for one anime.

# Code Plan

## Setting up the “database” for the upcoming anime

1. Get the list of upcoming anime/movies/ovas from some website.
2. Parse release date to determine month of start date
3. Parse Info to determine genre
4. Classify upcoming anime according to month and genre
5. Store all upcoming anime in a master array
6. Store list in hash table using open hashing; hash function: TBD

## Searching the “database”

1. Function that returns all values corresponding to a specific index (Jan, Feb, etc.) (Searching according to month/genre/seasons)
2. Function that searches master array to check if a specific anime is upcoming (For specific anime searches)

## Defining The GUI

1. Search box for typing specific anime name
2. List of months and genre; click (preferably swipe) to get results.

# Progress log

1. Using urllib.request to fetch the myanime
2. Using BeautifulSoup to parse the HTML of the website with lxml parser
3. Been able to generate the table of genre

# Lookups!

* User-Agent header

# To-Do

1. Work on getting links to synopsis/or the synopsis itself/tags
2. Create a master program that selects the right url for the given season
3. Expanding the image in the main frame of the window in tkinter
4. Create Toplevel windows in tkinter for each of the seasons
5. Figure out how to pull out multiple links within a tag
6. The event handlers call the get\_Anime function and pass in the url of the season they’d like to get the anime info from
7. get\_Summer() – event handler for “Summer” button
8. get\_Winter() – event handler for the “Winter” button
9. get\_Spring () – event handler for the “Spring” button
10. get\_Fall() – event handler for the “Fall” button

# Issues

1. If Summer button is clicked, regardless of the month, the function will only return the “upcoming” anime for that summer period. Summer button should take the user to a window that presents the dates available in the charts, then the user can select the year they want info for.

# OOP

Inheritance: A subclass **is a** more specific instance of a superclass. Example: A shepherd **is a**dog, which **is an**animal.

OO is about grouping DATA with the FUNCTIONS that manipulate that data and hiding HOW it manipulates it so you can MODIFY the behavior through INHERITANCE

OOP is about such concepts as encapsulation, inheritance, polymorphism, abstraction and many others. It is a specific way of software design, a specific way of mapping a problem to a software solution.

the object-oriented approach encourages the programmer to place data where it is not directly accessible by the rest of the program.

Instead the data is accessed by calling specially written functions, commonly called methods, which are either bundled in with the data or inherited from "class objects" and act as the intermediaries for retrieving or modifying those data. The programming construct that combines data with a set of methods for accessing and managing those data is called an object.

Encapsulation: It refers to the bundling of data with the methods that operate on that data.[[5]](https://en.wikipedia.org/wiki/Encapsulation_(computer_programming)#cite_note-Rogers01-5) Encapsulation is **used to hide the values or state of a structured data object inside a class**, preventing unauthorized parties' direct access to them.

Existential types: Separation of Implementation from Interface??

Interface: T = ∃X { a: X; f: (X → int); }" describes a module interface that has a data member named *a* of type *X* and a function named *f* that takes a parameter of the *same* type *X* and returns an integer

### Implementation example:

* intT = { a: int; f: (int → int); }
* floatT = { a: float; f: (float → int); }

These types are both subtypes of the more general existential type T and correspond to concrete implementation types, so any value of one of these types is a value of type T