CS5004, Object-Oriented Design and Analysis

Lab5: Inheritance, Interfaces and ADTs

Therapon Skoteiniotis, Tamara Bonaci and Abi Evans

Due Date: Saturday February 26th, 11:59pm

1. Summary

In today's lab, we will continue our conversation about inheritance, exceptions, and unit testing, as well as about interfaces and Abstract Data Types (ADTs). We will focus on:

- Inheritance and abstract classes
- Public behavior, contracts and interfaces
- Relationships between abstract classes and interfaces
- Abstract data types
- Exceptions: throwing and properly handling (catching) exceptions, as well as writing our own exceptions
- Testing methods that can throw exceptions

Note 1: Labs are intended to help you get started, and give you some practice while the course staff is present and able to provide assistance. You are not required to finish all the questions during the lab, but you are expected to push your lab work to a designated repo on the Khoury GitHub.

GitHub at the end of the lab.

2. Inheritance, Abstract Classes and Interfaces

In our last several lectures, we talked about inheritance, abstract classes and interfaces in Java. Let's see how to use these concepts in practice.

Problem 1 – Digital Index of Artists

A non-profit organization in Seattle is trying to organize a digital index of all artists who were born, and/or lived and created their art in Seattle. They have an idea how they'd like to go about it, but they need your help with implementation. Here's their idea:

An `Artist` can be:

- An `Actor`
- A `Musician`
- A `Painter`
- A `Photographer`
- A `Filmmaker`
- A `Dancer`
- A `Poet`

All `Artists` contain the following information:

- `Name`, containing information about an `Artist's` first and last name
- `Age`, which is an Integer in the range [0, 128], containing information an `Artist's` age
- `Genres`, which is a String array, representing an `Artist's` genres
- `Awards`, which is a String array, representing all awards that an `Artist` received

Additionally, all `Actors`, `Dancers` and `Filmmakers` keep track of the following information:

- `Movies`, a String array, listing all movies that they worked on (acted in)
- `Series`, a String array, listing all TV series that they worked on (acted in)
- `Other multimedia`, a String array, listing all other multimedia content that they worked on (acted in)

Similarly, all `Painters` and `Photographers` keep track of additional information:

- `Exhibits`, a String array of all exhibits where their art was shown

A `Musician` also keeps track of:

- `Recording company`, a String representing a `Musician's` recording company
- `Last record album`, a String representing the title of the latest recorded album

Lastly, a `Poet` also keeps track of:

- `Publishing company`, a String representing a `Poet's` publishing company
- `Last published collection`, a String representing the title of the latest published collection of poems

Finally, all `Artists` can receive an award, which gets added to their current array of awards, and the method signature for that behavior is `Artist receiveAward(String award)`.

- Design a Java program to capture the information and properties described by the non-profit organization's designers.
- Generate the final UML Class Diagram of your solution using IntelliJ, and push it to the package Problem 2 along with your code.

3. Abstract Data Types (ADTs)

Problem 2 – List of Strings ADT

Please design and implement a **List of `Strings`** ADT. Your implementation should implement a linked list, and it should provide the following operations on a `Lists of Strings`.

1. `isEmpty` : checks whether or not the list is empty.
2. `size` : gets the total number of elements in the list.
3. `contains` : consumes a `String`, and checks if the `String` is in the list or not.
4. `containsAll`: consumes another list of `Strings`, and checks that all elements of this list are in the list passed as argument.
5. `filterLargerThan` : takes the maximum `String` length, and returns a list with all elements whose length is greater than the maximum length removed.
6. `hasDuplicates`: check if the list has at least one duplicate element.
7. `removeDuplicates`: returns the list with all duplicates removed.