Theo Rowlett
CS202 Project 4
Design and Efficiency Writeup

Overview

The purpose of Project 4 is to create an application to add various different types of activities utilizing inheritance, dynamic binding, and a data structure using an an array of linearly linked lists. This program must be built using Java. For my project, I decided to utilize two different categories of activities, exercise and photography, each with 3 specific subclasses. As always, we do not use getters and setters as they are not object oriented.

Inheritance Hierarchy

The inheritance hierarchy is very similar to C++. Instead of defining a subclass like in C++:

```
class subclass:public baseclass
```

In Java it would look like:

```
public class Subclass extends Baseclass
```

The keyword extends defines the relationship in Java, in this case saying Subclass “is-a” Baseclass. Utilizing this we created an inherited relationship with all of our activities. Activity is the base class, or super class, from which all of the other classes inherit data from. Activity is an abstract class, since we never intend to actually create an Activity object this is only an Activity object. Since this class is common, start time, end time, and name are all protected data within Activity. If we wanted to override or access this data within a subclass, we would use the super keyword. Super works similarly to this in c++. For example, if we also had a start_time variable within one of our derived classes, but we wanted to access the Activity start_time, we would call it using the super keyword (ex. super.start_time).

Function overloading vice overriding

Function overloading, at least in my limited exposure, seems to work exactly the same as in C++. Function overloading is just creating a method with the same name that has a different argument list. For my application I used it for recursively building and displaying my linear linked lists within the array. Function overriding, works a little bit different.

Overriding is when multiple methods exist with the same name within an inheritance. There’s several ways to go about this, but for my program 4 the display() and build() methods within the Activity superclass are overridden in the subclasses. This works because Activity is an abstract class. We can also do this manually using the @Override command, but this approach wasn’t necessary or really desired for the implementation of Program 4.

Dynamic Binding

Dynamic binding is different than in C++, in that we don’t need a dynamic binding keyword. If binding static methods binding will take place at compile time, dynamic binding conversely takes place at run time. Since when we create our different activity types we use dynamic binding, we are able to call the proper overridden methods.