CMPUT 301
Lab2 Presentation

Lab 2 Overview

OOP Review

Android Basics

Lab demo of ListView

OOP Review

- Inheritance
  - Superclass and Subclass
- Polymorphism
  - @Override
- Encapsulation
  - Why do we need encapsulation?

Inheritance

- subclass (child) - the class that inherits from another class
- superclass (parent) - the class being inherited from
- The subclass will inherit all fields and methods. (basic constructor, attributes and all the user defined functions)

Inheritance

- What does super() or super(variables) do?
  Construct its super class. Calls the parent class's constructor
- Which constructor the program runs first? (super or sub)
  Super! Because ImportantTweet is a Tweet, so we need to be a Tweet first
- Without super() in subclass, super() would automatically called.

  (However, when your superclass doesn't have a no-arg constructor, the compiler will require you to call super with the appropriate arguments.)

Polymorphism

Override

```
1 usage  1 inheritor
public abstract class Animal {
    1 usage
    protected String name;
    1 usage
    public Animal(String name) {
        this.name = name;
    }
    no usages  1 implementation
    public abstract String speak();
}
```

```
no usages
public class Dog extends Animal {
    no usages
    public Dog(String name) {
        super(name);
    }
    no usages
    @Override
    public String speak() {
        return "woof";
    }
}
```

Polymorphism

Overload

```
no usages
public class Main {
    1 usage
    public static int add(int a, int b) {
        return a + b;
    }
    1 usage
    public static int add(double a, double b) {
        int int_a = (int) a;
        int int_b = (int) b;
        return int_a + int_b;
    }
    no usages
    public static void main(String[] args) {
        System.out.println(add( a: 2,  b: 3));
        // Round down then add 2 integers
        System.out.println(add( a: 2.3,  b: 4.6));
    }
}
```

Polymorphism

Upcasting

```
public static void main(String[] args) {
    ArrayList<Animal> animalArrayList = new ArrayList<>();
    Animal dog = new Dog( name: "Doug");
    Animal cat = new Cat( name: "Walter");
    animalArrayList.add(dog);
    animalArrayList.add(cat);
}
```

Data Abstraction and Encapsulation

To prevent misuse, data are only visible and accessible by related functions.

For example, from Lab 1, class Tweet does not allow general access to *message* and *date*, and only allows access through getters and setters. This is a form of data abstraction.

Data Abstraction and Encapsulation

Modifier	Class	Package	Subclass	World
`public`	✔	✔	✔	✔
`protected`	✔	✔	✔	✘
`no modifier*`	✔	✔	✘	✘
`private`	✔	✘	✘	✘

Data Abstraction and Encapsulation (cont.)

Encapsulation – Hides how a class work internally.

Why? – Encapsulation encourages *decoupling*

Decoupling – When developing a feature A related to an existed feature B, if you know how feature B works and make too many assumptions, then you are relying on your knowledge about feature B. If feature B changes in the future, you have to change A's implementation too.

Reminder: Assignment 0

If you have questions or difficulties with Assignment 0, don’t hesitate to ask a TA