COMP90041
Programming and Software Development
2020 - Semester 2
Lab 4

Yuhao(Beacon) Song

yuhsong1@unimelb.edu.au

yuhsong@student.unimelb.edu.au

Introduction

- **Timetable**
  - Tue(18) 16:15-17:15 (Melbourne Time)
    - https://unimelb.zoom.us/j/94854648719?pwd=WUY0NmR6MkI5UVZBUWhGNWFIU216Zz09
  - Wed(19) 17:15-18:15 (Melbourne Time)
    - https://unimelb.zoom.us/j/93723434766?pwd=MGh4QkJuZnhJR21LZ0VqeXhJQU52UT09

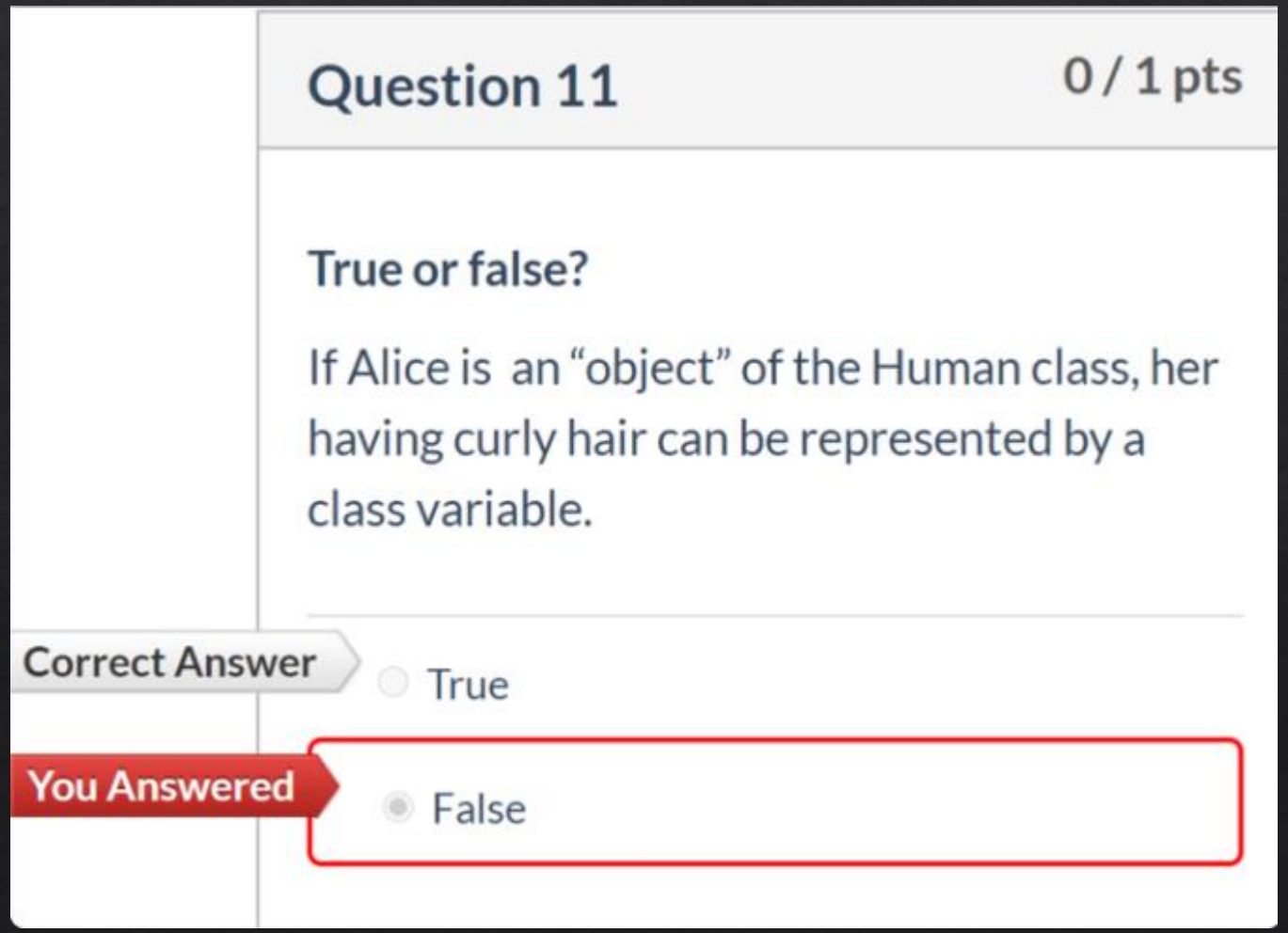
- **GitHub Page (Tutorial Materials, Solutions, Additional Resources)**
  - https://github.com/Beaconsyh08/COMP90041-2020SEM2

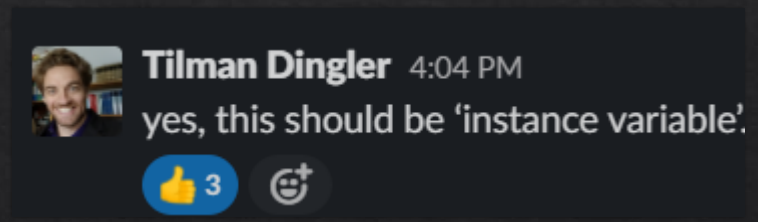
- **PollEv**
  - https://pollev.com/yuhsong

Brief Video About Git & GitHub for Lab03

- https://www.youtube.com/watch?v=Sfxt7r0O5xA

Practice Quiz





Outline

- Last week's Q
- Lecture Review
- Exercise & demo
- The previous slides (Optional)

Class

- **A Class Is a Type**
- A class is a special kind of programmer-defined type, and variables can be declared of a class type
- A value of a class type is called an **object** or an **instance** of the class
  - If A is a class, then the phrases "bla is of **type** A," "bla is an **object** of the class A," and "bla is an **instance** of the class A" mean the same thing
- A class determines the types of **data** that an object can contain, as well as the **actions(methods/functions)** it can perform

Java Class Structure

```
//  comments about the class
public class MyClass
{  //  comments about the variable
   Account myVar

   //  comments about the method
   public int myMethod (String arg)
   {
   }
}
```

method body

method header

Constructor

- A **constructor** is a special kind of **method**

  → to **initialize** the **instance variables** for an **object**:

  public ClassName(anyParameters){code}

  - A constructor must have the same name as the class
  - A constructor has no type returned, not even void
  - Constructors are typically overloaded (talk later)

- A **constructor** is called when an **object** of the class is created using new

  *ClassName **object/instance** = **new** ClassName(anyArgs);*

  - This is the **only** valid way to invoke a constructor: a constructor cannot be invoked like an ordinary method

Constructor

- If you do not include any constructors in your class, Java will automatically create a default **no-argument constructor** that takes no arguments, performs no initializations, but allows the object to be created.

```
public class Dog {
    // Default Constructor
    public Dog() {
    }
}
```

- If you include even one constructor in your class, Java will not provide this default constructor.
- Normally you should provide your own **no-argument constructor**.

Overloading

- **Overloading** is when two or more methods **in the same class have the same method name**
- But, to be valid, any two **must have different signatures**
- **Signature**: the method name and the list of **types** for parameters →

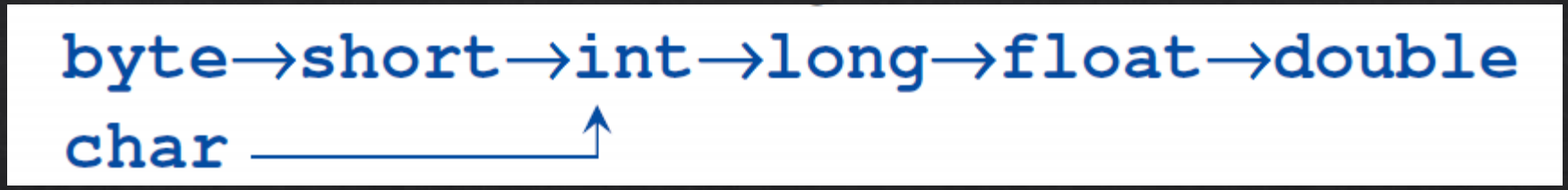
  Return Type Not Included!

```
public void setDate(int month, int day,int year)
public void setDate(String month, int day,int year)
public void setDate(int year)

setDate(int, int, int)
setDate(String, int, int)
setDate(int)
```

Automatic Type Conversion

- If Java cannot find a method **signature** that **exactly** matches a method invocation, it will try to use **automatic type conversion**.



Ambiguous Invocation

- The interaction of overloading and automatic type conversion can have unintended results
  - In some cases of overloading, because of automatic type conversion, a single method invocation can be resolved in **multiple ways**
  - **Ambiguous method invocations** will produce an **error** in Java

public and private Modifiers

- **public** → no restrictions
- **private** → cannot be accessed by name outside of the class
- Make all **instance variables** **private**, good programming practice!
- Usually, methods are **private** only if used as helping methods for other methods in the class

	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	World
public	+	+	+	+	+
protected	+	+	+	+	
no modifier	+	+	+		
private	+				

+ : accessible blank : not accessible

Accessor and Mutator Methods

- **Accessor(getter)** → obtain the value
  - The data can be accessed but not changed
  - The name of an accessor method typically starts with the word get

- **Mutator(setter)** → change the value
  - Incoming data is typically tested and/or filtered
  - The name of a mutator method typically starts with the word set

Writing Java Class

1. Write a class named `Appointment` that contains instance variables `startTime`, `endTime`, and a `date` which consists of a `day`, `month` (valid values are January through December) and `year`. All times should be in military (24 hours) format, therefore it is appropriate to use integers to represent the time. Write accessor and mutator methods, and helper methods for setting and displaying an appointment.

2. Add two constructors to the `Appointment` class, including a default constructor and a constructor to initialise an Appointment to suitable arguments.

3. Write a driver program that tests your `Appointment` class.

4. Rewrite the mutator method for setting the `day` in the `Appointment` class in a way that it has to use the `this` keyword.

5. Modify mutator methods for setting `day`, `month` and `year` so that only valid values for those instance variables are accepted. Then test the class again (using your driver class) and make sure that it still works and accepts valid values only.

6. Write a precondition for the `endTime` mutator method that ensures validity of `endTime`.

7. ~~Draw a UML class diagram for the classes used in the exercises above.~~

Exercise