COMP90041
Programming and Software Development
2020 - Semester 2
Lab 8

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

Outline

- Lecture Review
- Exercise
- Assignment 2

Object-Oriented Programming (OOP)

- **Encapsulation**
  - This is the practice of keeping fields within a class **private**, then providing access to them via **public methods**. It's a protective barrier that keeps the data and code safe within the class itself. This way, we can re-use objects like code components or variables without allowing open access to the data system-wide.
- **Inheritance**
  - This is a special feature of OOP in Java. It lets programmers create new classes that share some of the attributes of existing classes. This lets us build on previous work without reinventing the wheel.
- **Polymorphism**
  - This Java OOP concept lets programmers use the same word to mean different things in different contexts. One form of polymorphism in Java is **method overloading**. The other form is **method overriding**.

Polymorphism

- **Polymorphism** is the ability to associate many meanings **to one name**

- **Recap**
- **Overloading**: same class or could happen in derived class and base class; different **signature** and same **name**.
- **Overriding**: derived class and base class; same **signature** and **returned type**.

Late Binding/Dynamic Binding

- The process of associating a method **definition** with a method **invocation** is called **binding**
- **Early/Static binding:** the method definition is associated when the code is **compiled**
- **Late/Dynamic binding:** the method definition is associated when the method is **invoked** (at **run time)**

- **Compile-time:** the time the **source code** is **converted** into an **executable code (javac)**
- **Run time:** the time the **executable code** is started **running (java)**

Upcasting and Downcasting

- **Upcasting** is casting to a supertype
  - Dog dog = new Dog();
  - Animal upcastedAnimal = (Animal) dog;
- **Downcasting** is casting to a subtype
  - Animal animal = new **Dog**();
  - Dog downcastedDog = (Dog) animal;
- **ClassCastException – Run Time**
  - Animal animal = new **Animal**();
  - Dog downcastedDog = (Dog) animal;
  - animal's **runtime type** is Animal, and so when you tell the **runtime** to perform the cast it sees that animal **is not really** a Dog.
- **Upcasting** is always allowed, but **Downcasting NOT**

Chap8_Question5

5. Consider a graphics system that has classes for various figures—say, rectangles, boxes, triangles, circles, and so on. For example, a rectangle might have data members' height, width, and center point, while a box and circle might have only a center point and an edge length or radius, respectively. In a well-designed system, these would be derived from a common class, `Figure`. You are to implement such a system.

   The class `Figure` is the base class. You should add only `Rectangle` and `Triangle` classes derived from `Figure`. Each class has stubs for methods `erase` and `draw`. Each of these methods outputs a message telling the name of the class and what method has been called. Because these are just stubs, they do nothing more than output this message. The method `center` calls the `erase` and `draw` methods to erase and redraw the figure at the center. Because you have only stubs for `erase` and `draw`, center will not do any "centering" but will call the methods `erase` and `draw`, which will allow you to see which versions of `draw` and `center` it calls. Also, add an output message in the method `center` that announces that `center` is being called. The methods should take no arguments. Also, define a demonstration program for your classes.

   For a real example, you would have to replace the definition of each of these methods with code to do the actual drawing. You will be asked to do this in Programming Project 8.6.

Chap8_Question6

6. Flesh out Programming Project 8.5. Give new definitions for the various constructors and methods `center`, `draw`, and `erase` of the class `Figure`; `draw` and `erase` of the class `Triangle`; and `draw` and `erase` of the class `Rectangle`. Use character graphics; that is, the various `draw` methods will place regular keyboard characters on the screen in the desired shape. Use the character `'*'` for all the character graphics. That way, the `draw` methods actually draw figures on the screen by placing the character `'*'` at suitable locations on the screen. For the `erase` methods, you can simply clear the screen (by outputting blank lines or by doing something more sophisticated). There are a lot of details in this project, and you will have to decide on some of them on your own.

Chap8_Question7

7. Define a class named `MultiItemSale` that represents a sale of multiple items of type `Sale` given in Display 8.1 (or of the types of any of its descendent classes). The class `MultiItemSale` will have an instance variable whose type is `Sale[]`, which will be used as a partially filled array. There will also be another instance variable of type `int` that keeps track of how much of this array is currently used. The exact details on methods and other instance variables, if any, are up to you. Use this class in a program that obtains information for items of type `Sale` and of type `DiscountSale` (Display 8.2) and that computes the total bill for the list of items sold.

Reminder

- No Tutorial Next week – Mid Sem Break
- Next Tutorial: **Oct 13th/14th**
- Notice the **Daylight Saving Time Change** on Oct 4th
  - If you are not in Melbourne, recalculate the corresponding time in your time zone