Programming Techniques COSC1284/2010

Tutorial 11

Agenda

- Tutorial/Lab
  - Read chapter 14 from the textbook.
  - Discuss the concepts with your tutor and fellow classmates
    - Complete chapter 14 - Exercises 1 - 2
  - Attempt on your own
    - Complete chapter 14 - Exercises 3 - 4
- Note: Please refer to tutorial 4 for online instructions.

Continuing to developing the card application with the expansion of OOP.

Inheritance

- For this week, we are going to expanding the concepts of object oriented programming, such as inheritance, superclasses, subclasses, overwriting methods, abstract classes and interfaces.
- As the number of classes increases, similar or same behaviours tends to appear.
- Inheritance can solve this problem by moving the same behaviours in a class that is shared among other classes.
- The shared behaviour within that class is known as the superclass and the classes that inheritance such behaviours is known as the subclasses.
- Subclasses can add their own behaviours that is not general, but rather specific to that class.
- Subclasses can also overwrite methods from the superclass(es) such as the toString, equals and other methods.
- In the direction from the superclass to the subclasses can be thought of going from the generic to the specific

Abstract Classes

- With this though in mind, this concept can be taken further as that a superclass can be so generic that it lacks enough detail to bring it into existence (instantiate it into memory).
- This is known as an abstract class (superclass), that doesn't contain enough detail to build itself. A concrete class (subclass) adds that detail for completion.
- For example, if someone is asked, "What is a fruit?" they typically give an example of a fruit (apple, banana, orange, etc). Not the definition of a fruit, a food that contains seeds. The definition only is insufficient when describing a fruit (try to imagine a fruit without thinking about apples, bananas, oranges or any other fruits).
- In the java code, the Fruit class is defined as the abstract superclass where the Apple class is the concrete subclass.
- Note: Abstract classes can still have implementation details, but can not be instantiated directly, only by a concrete subclass.

Interfaces

- This idea can be reframed in a different way, where classes do a generic task, i.e. a mouse click and then create a specific class that writes a file via a button. To join this generic and specific behaviours requires a bridge between the code.
- That is, the button that saves the file is now clickable.
- This bridge is formally is known as an interface.
- This interface is like an 100% abstract class, class that lacks all implementations, though an interface is not class, it's an interface between classes.

Polymorphism

- Definition: The condition of occuring in several different forms
- In OOP, polymorphism promotes code reuse by calling the method in a generic way.
- For instance, a superclass is responsible for emergency services and a generic method within the superclass is named, callOut, as in any emergency service is able to be called out to arrive at a emergency situation.
- The classes that inherit from EmergencyServices class is Police, Fire and Ambulance.
- All these subclasses have the ability to call out, but do it in a different way, i.e. using different vehicles.
- Furthermore, the subclasses have then own ability that are not generic, like the police can arrest, while the others can not, though caution is needed when working with polymorphism.
- Casting and Dynamic binding is useful when it comes to polymorphism.

Exercise 14.1

- Design a better strategy for the Player.play method. For example, if there are multiple cards you can play, and one of them is an eight, you might want to play the eight.

- Think of other ways you can minimize penalty points, such as playing the highest-ranking cards first. Write a new class that extends Player and overrides play to implement your strategy.

Exercise 14.2

- Write a loop that plays the game 100 times and keeps track of how many times each player wins. If you implemented multiple strategies in the previous exercise, you can play them against each other to evaluate which one works best.

- Hint: Design a Genius class that extends Player and overrides the play method, and then replace one of the players with a Genius object.