CST209

Object-oriented Programming C++

(Week 8)

Prepared by Dr. Teo Bee Guan

Presented by Mr. Venantius Kumar

Content

- Concept of Polymorphism
- Function Overloading
- Operator Overloading
- Function Overriding
  - Virtual Function

Concept of Polymorphism

- Polymorphism means "many forms“.
- The same entity (function or operator) behaves differently in different scenarios.
- For example:
  - The + operator in C++ is used to perform two specific functions. When it is used with numbers (integers and floating-point numbers), it performs addition.
  - And when we use the + operator with strings, it performs string concatenation.

Concept of Polymorphism

- We can implement polymorphism in C++ using the following ways:
  1. Function overloading
  2. Operator overloading
  3. Function overriding
  4. Virtual functions

Function Overloading

- In C++, we can use two functions having the same name if they have different parameters (either types or number of arguments).

- Depending upon the number/type of arguments, different functions are called.

Practice: Example 1

In-Class Exercise 1

Create three overloaded functions for multiplication.

- Function 1 (Accept two integer arguments)
- Function 2 (Accept two double arguments)
- Function 3 (Accept three integer arguments)

Test each of the overloaded functions in the main function.

Practice: Example 1

Operator overloading

- C++ allows you to redefine how standard operators work when used with class objects.
- For example, the following statement adds five days to the date stored in the today object:

```
today.add(5);
```

- The use of an operator can be more intuitive

```
today += 5;
```

Practice: Example 2, 3

In-Class Exercise 2

Extends the Code Example 3 by overloading the operator minus (-) and multiply (*).

Test your overloaded operators in the main function.

(Simplification of fraction is not required.)

Function Overriding

- In C++ inheritance, we can have the same function in the base class as well as its derived classes.
- When we call the function using an object of the derived class, the function of the derived class is executed instead of the one in the base class.
- Different functions are executed depending on the object calling the function.

Practice: Example 4

Virtual Functions

- From the previous code example, we cannot override functions if we use a pointer of the base class to point to an object of the derived class.

- However, by using **virtual functions** in the base class, we can ensure that the function can be overridden even when a parent object is used to invoke the function.

- Virtual function is a base class member function that we can redefine in a derived class to achieve polymorphism.

Practice: Example 5

In-Class Exercise 3

Write a class named as “Shape” with only a member variable, id. This class should have a virtual function, displayInfo().

Create another class named as “Rectangle” which inherit the Shape class. The Rectangle class has two member variables, width & height. This class should also have a function, displayInfo() that is supposedly override the displayInfo() function from the Shape class.

Write a main function to test the class by showing how displayInfo() function from Rectangle class overrides the one in Shape class. The displayInfo function should be invoked by the Shape class object.

In-Class Exercise 4

Write a class named as “Polygon” with two member variables, width & height. In the class, create a virtual function named as area() that returns the result as double.

Write a second class named as “Rectangle” that inherit the Polygon class. In the class, create a function named as area() to calculate the area of rectangle.

Write a third class named as “Triangle” that inherit the Polygon class. In the class, create a function named as area() to calculate the area of a triangle.

Create a main function to test your class by having a Polygon object invoking the area() function of Rectangle and another Polygon object invoking the area() function of Triangle.

See you next class