day-8-object-oriented-programming

September 6, 2023

1 Day 8: Object oriented programming

1.0.1 Topics

- encapsulation
- static polymorphism
- composition
- dynamic polymorphism
- compiler complaints

1. Encapsulation

a) Write a class `VectorR3` with member variables `x`, `y`, and `z` to represent a vector in three-dimensional space.

b) Use access specifiers to ensure that member variables can only be read and modified via setters and getters, and implement the corresponding methods.

c) Overload the stream operator to print `VectorR3` instances as `(x, y, z)`.

d) Overload the `+` operator for vector addition.

e) Test all implemented functionalities.

[ ]:

2. Static Polymorphism

a) Write a class `Position` via public inheritance from `VectorR3` to describe a position in three-dimensional space.

b) Add a method `distance(const Position& p2)` that returns the distance between `p2` and the `Position` object for which the method has been called (the implicit object parameter).

c) Test all implemented functionalities.

d) Similarly, define a class `Force` via inheritance from `VectorR3` to describe a force in three-dimensional space.

e) Reason whether your class hierarchy fulfills the Liskov Principle.

[ ]:

3. Composition

a) Write a class `Particle` that describes a particle based on its charge and position in three-dimensional space. Make use of the `Position` class developed in Exercise 1.

b) Add a method `force(const Particle& p2)` that returns the Coulomb force between `p2` and the `Particle` object for which the method has been called (the implicit object parameter). Make use of the `Force` class developed in Exercise 2 d).

c) Overload the stream operator to print the position and charge of `Particle` objects.

d) Test all implemented functionalities.

[ ]:

4. Dynamic Polymorphism

a) Define a class `Shape` that can represent plane figures exhibiting the attributes width, height, position in three-dimensional space, and color.

b) Add a method `getColor(...)` that returns the color of a shape. Provide a default implementation that applies for all `Shape` objects.

c) Add a method `shift(...)` that shifts the (center of an) object in x-, y-, and z-direction. Provide a default implementation that applies for all `Shape` objects.

d) Add the abstract methods `area()` and `perimeter()`.

e) Define the derived classes `Rectangle` and `RightTriangle`, and provide implementations for all abstract methods of `Shape`.

f) Create a free-standing function that takes an array of `Shapes` and a color as input, and returns an array that contains only those shapes of the input array that exhibit the specified color.

g) Test all implemented functionalities.

[ ]:

5. Compiler Complaints?

Reason whether or not the below code snippets are compilable. Afterwards, compile the snippets to check your reasoning.

a)

[ ]:
```
class Foo
{
        int a;
};

int main()
{
        Foo foo();
        foo.a;
}
```

b)

[ ]:
```
class Bar
{

public:
        void function(const int& value) const
        {
                attribute1_ = value;
        }

private:
        int attribute1_;
};

int main()
{
        Bar bar();
        bar.function();
}
```