Informatik für Mathematiker und Physiker - AS18

Exercise 14: Inheritance & Polymorphism

Handout: 18. Dez. 2018 06:00

Due: 3. Jan. 2019 23:59

Task 1: Polymorphic Functions

Open Task (/solve/AZkG2zWs3RYMzEySo)

Default

This task is a text based task. You do not need to write any program/C++ file: the answer should be written in main.txt *(and might include code fragments if questions ask for them).*

Task

Given is the following class hierarchy:

```
#include <iostream>
#include <string>

class A {
public:
    std::string name;
    A(std::string _name) : name(_name) {}
    virtual void say_hello() { std::cout << "A says hi to " << name <<
    void say_bye() { std::cout << "A says bye to " << name << "\n"; }
};

class B : public A {
public:
    B(std::string _name) : A(_name) {}
    void say_hello() { std::cout << "B greets " << name << "\n"; }
};

class C : public A {
public:
    C(std::string _name) : A(_name) {}
    void say_bye() { std::cout << "C say goodbye to " << name << "\n"
};
```

For each of the following functions, specify for each call whether it is polymorphic, i.e., the dynamic type determines the function to be called, or not.

1. ```
   void f() {
       A x("Jane");
       x.say_hello(); // call 1
       B y("John");
       y.say_hello(); // call 2
       x = y;
       x.say_hello(); // call 3
   }
   ```
2. ```
   void g() {
       A* x = new A("Jane");
       (*x).say_hello();  // call 1
       B* y = new B("John");
       (*y).say_hello();  // call 2
       x = y;
       (*x).say_hello();  // call 3
   }
   ```
3.

```
void h() {
    A* x = new A("Jane");
    (*x).say_bye();  // call 1
    C* y = new C("John");
    (*y).say_bye();  // call 2
    x = y;
    (*x).say_bye();  // call 3
}
```

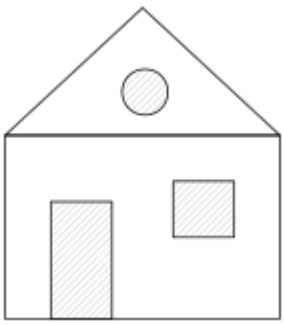
Task 2: House Painting

Open Task (/solve/iotjcan5DdttkxKQw)

Default

Task

Mr. Brush is hired to paint a house facade. To make an offer, he needs to know the area that requires painting. He looks at the house facade and sees that it can be approximated by using different shapes: Triangle (given by width and height), Rectangle (given by width and height) and Circle (given by radius). Holes in shapes can be accounted for by subtracting the non-paintable areas, e.g. a door or a window.



Help Mr. Brush by implementing the three required shapes: `Rectangle`, `Triangle`, and `Circle`. Each shape is implemented with its own class that inherits from class `Shape` and overrides the virtual member function `get_area` that returns the area defined by the shape.

Steps:

1. Complete declarations of class `Rectangle`, `Triangle` and `Circle` in file `house_shapes.h`.
2. Implement their member functions in file `house_shapes.cpp`.

Testing: The test input is a list of shape objects provided in a textual representation. The template includes a parser for the test input to avoid a lengthy specification. You do not have to implement it, but you may want to take a look at it to understand how it works.

Also you can use it to test your implementation manually.

Example: `+ rectangle 4 3 - circle 1.5 end` is a rectangular facade with a hole.
The following EBNF defines the input:

House facade description EBNF:

```
facade    = area { area } .
area      = op { triangle | rectangle | circle }
op        = "+" | "-"                 // add (+) or subtract (-) are
triangle  = "triangle"  double double // width height
rectangle = "rectangle" double double // width height
circle    = "circle"    double        // radius
double    = C++ double value
```