

Семинар #2: Наследование. Домашнее задание.

Задача 1. Фигуры

В файле `shape.cpp` написаны простейшие классы `Circle`, `Rectangle` и `Triangle`, описывающие геометрические фигуры: круг, прямоугольник и треугольник. При написании этих классов наследование не было использовано, в следствии чего некоторые поля и методы одинаковы у всех классов (поле `mPosition` и методы `getPosition` и `setPosition`).

- Создайте класс `Shape`, который бы описывал абстрактную фигуру и содержал бы поля и методы, общие для всех фигур. Измените код классов `Circle`, `Rectangle` и `Triangle` так, чтобы они наследовались от класса `Shape`. Общие для всех фигур поля и методы должны содержаться только в классе `Shape`, но не в классах-наследниках.
- Добавьте метод `void move(Vector2f change)` в класс `Shape`. Этот метод должен изменять поле `mPosition` на значение `change`. Так как остальные классы наследуются от `Shape`, то этот метод можно будет вызвать у всех объектов дочерних классов. Протестируйте этот метод, изменив положения объектов дочерних классов.

Задача 2. Приведение типов при наследовании

Пусть есть два следующих класса:

```
struct Alice
{
    int x;
    void func() const {std::cout << "Alice " << x << std::endl;}
};

struct Bob : public Alice
{
    int y;
    void func() const {std::cout << "Bob " << x << " " << y << std::endl;}
};
```

Определите, скомпилируется ли следующий код, использующий эти классы, и, если скомпилируется, то что будет напечатано в следующих программах:

1. `Alice a {10};`
`Bob b {20, 30};`
`a = b;`
`a.func();`
2. `Alice a {10};`
`Bob b {20, 30};`
`b = a;`
`b.func();`
3. `Bob b {20, 30};`
`Alice* p = &b;`
`p->func();`
4. `Alice a {10};`
`Bob* p = &a;`
`p->func();`

Пусть есть следующие классы:

```
struct Alice
{
    int x;
    void func() const {std::cout << "Alice " << x << std::endl;}
};

struct Bob : public Alice
{
    int y;
    void func() const {std::cout << "Bob " << x << " " << y << std::endl;}
};

struct Casper : public Bob
{
    int z;
    void func() const {std::cout << "Casper " << x << " " << y << " " << z << std::endl;}
};
```

Определите, скомпилируется ли следующий код, использующий эти классы, и, если скомпилируется, то что будет напечатано в следующих программах:

5. Alice a {10};
Bob b {20, 30};
Casper c {40, 50, 60};
a = c;
a.func();
6. Alice a {10};
Bob b {20, 30};
Casper c {40, 50, 60};
Alice* p = &c;
p->func();
7. Alice a {10};
Bob b {20, 30};
Casper c {40, 50, 60};
Bob* p = &a;
p->func();
8. Alice a {10};
Bob b {20, 30};
Casper c {40, 50, 60};
Bob* p = &c;
p->func();

Для того, чтобы сдать эту задачу нужно создать файл в формате .txt и, используя любой текстовый редактор, записать в него ответы в следующем формате (ответы ниже неверны):

- 1) Error
- 2) Alice 10
- 3) Bob 20 30
- ...

После этого, файл нужно поместить в ваш репозиторий на github.

Задача 3. Изменение цвета

В файле `draggable.cpp` написан класс `Draggable`, который описывает передвигаемый курсором мыши прямоугольник. Ваша задача – написать класс `DraggableWithColorChange` – наследник класса `Draggable`. Новый класс должен также описывать передвигаемый прямоугольник, но, во время передвижения прямоугольника, его цвет должен меняться на другой. Конструктор нового класса будет иметь вид:

```
DraggableWithColorChange(sf::RenderWindow& window, Vector2f position, Vector2f size, Color  
    baseColor, Color dragColor)
```

Где `baseColor` – это основной цвет прямоугольника, а `dragColor` – цвет прямоугольника при перетаскивании. Протестируйте этот класс в функции `main`.

Задача 4. Окна

В файле `window.cpp` написан класс `BaseWindow`, описывающий простейшее окно. Это окно состоит из двух прямоугольников. Первый прямоугольник определяет границы отрисовки окна, а второй определяет границы области, за которую прямоугольник можно перетаскивать.

- Создай класс `MessageWindow`, наследник `BaseWindow`. У объектов этого класса, помимо функционала `BaseWindow` должен быть текст, в котором отображается некоторая строка.
- Создайте классы `ErrorWindow` и `DoneWindow`, наследники класса `MessageWindow`. Эти два окна должны отличаться от окна `MessageWindow` только тем, что окно типа `Error` должно всегда рисоваться оттенком красного цвета, а окно типа `Done` – оттенком зелёного цвета.
- Создай класс `QuestionWindow`, наследник `BaseWindow`. У объектов этого класса, помимо функционала `BaseWindow` должен быть текст, в котором отображается некоторая строка. А также две кнопки внизу: `Ok` и `Cancel`. При нажатии на кнопку `Ok` в консоль должно выводиться строка `Ok`, а при нажатии на кнопку `Cancel` – строка `Cancel`. Используйте класс `Button` из файла `button.hpp` для создания кнопок.