METODY PROGRAMOWANIA – PROJEKT

PROGRAM 01

Pracujesz nad aplikacją, która dostarcza bibliotek graficznych do wyświetlania skomplikowanych obiektów. Kluczowe jest, aby tworzone rozwiązanie było elastyczne, pozwalało na łatwe dodawanie nowych bibliotek, oraz umożliwiało dynamiczną zmianę biblioteki dla aktualnie rysowanego obiektu. W obecnym momencie jesteś na początku projektu, więc praca skupia się na stworzeniu szkieletów potrzebnych klas oraz implementacją wyświetlania prostych kształtów.

Poszukiwane rozwiązanie jest przykładem odseparowania abstrakcji reprezentującej kształty od implementacji stanowiącej sposób wyświetlania. Oba byty istnieją równolegle bez wzajemnej ingerencji (większej niż jest to konieczne).

Dla testów wystarczy nam wyświetlanie 3 kształtów: koła, trójkąta oraz równoległoboku (ewentualnie kształtów skomponowanych z tych 3 składowych). Na razie chcemy dostarczyć 2 sposobów wyświetlania (bibliotek) – tekstowej oraz graficznej. Poniżej została opisana struktura klas oraz ich publiczne metody.

1. Podstawową klasą jest interfejs **Display** deklarujący metody wyświetlania poszczególnych kształtów:

   void drawTriangle(int a1, int a2, int b1, int b2);

   wyświetla trójkąt oparty na wektorach **[a1, a2]** oraz **[b1, b2]**. Dokładniejsze działanie metody zależy od konkretnej implementacji w klasie pochodnej.

   void drawCircle(int r);

   wyświetla koło o promieniu **r**. Dokładniejsze działanie metody zależy od konkretnej implementacji w klasie pochodnej.

   void drawParallelogram(int a1, int a2, int b1, int b2);

   wyświetla równoległobok oparty na wektorach **[a1, a2]** oraz **[b1, b2]**. Dokładniejsze działanie metody zależy od konkretnej implementacji w klasie pochodnej.

   Dwa sposoby wyświetlania reprezentowane są przez dwie klasy implementujące podany interfejs: **TextDisplay** oraz **GraphicalDisplay**. One też konkretyzują działanie powyższych metod.

2. **TextDisplay** służy do tekstowego wyświetlania obiektów. W ramach wywołania kolejnych metod powinien zostać wypisany tekst (zamiast nazw zmiennych powinny zostać wypisane ich wartości) oraz nastąpić przejście do nowej linii.

   void drawTriangle(int a1, int a2, int b1, int b2);

   wypisuje tekst: *Drawing a triangle from vectors (a1, a2), (b1, b2).*

   void drawCircle(int r);

   wypisuje tekst: *Drawing a circle with radius r.*

   void drawParallelogram(int a1, int a2, int b1, int b2);

   wypisuje tekst: *Drawing a parallelogram from vectors (a1, a2), (b1, b2).*

3. **GraphicalDisplay** służy do graficznego wyświetlania obiektów. Ogólnie rzecz biorąc, figury są rysowane w konsoli poprzez postawienie znaku „t” (dla trójkąta), „c” (dla koła), „p” (dla równoległoboku) w każdym punkcie należącym do figury (wnętrza bądź brzegu). Zakładamy, że jednej jednostce na osi poziomej odpowiada jeden znak, na osi pionowej natomiast jedna linia. Każda figura jest wyrównana do lewej strony ekranu oraz jest wyświetlana bezpośrednio pod wcześniej wyświetlonym obiektem.

   Poszczególne metody rysują kształty w następujący sposób:

   void drawTriangle(int a1, int a2, int b1, int b2);

   Zakładamy, że **a1, a2, b1** to całkowite liczby dodatnie, a **b2** to całkowita liczba ujemna. Rysuje trójkąt oparty na wektorach **[a1,a2]** oraz **[b1,b2]** (zaczepionych we wspólnym punkcie). Innymi słowy wierzchołkami trójkąta mogą być punkty **(0,0), (a1,a2), (b1, b2)**. Tak jak wcześniej napisano, kształty są wyrównywane do lewej strony ekranu oraz rysowane jeden pod drugim bez nachodzenia na siebie i bez dodatkowych linii. Dla przykładu wywołanie:

   drawTriangle(3,2,2,-2);

   narysuje trójkąt o wierzchołkach w punktach **(0,0), (3,2), (2,-2)** tak jak na rysunku poniżej:

```
   t
  t
 ttt
  tt
  t
```

void drawCircle(int r);

Zakładamy, że **r** to całkowita liczba dodatnia. Rysuje koło o promieniu **r** (oraz ustalonym środku). Dla przykładu wywołanie:

drawCircle(3);

narysuje koło:

```
   c
 ccccc
 ccccc
ccccccc
 ccccc
 ccccc
   c
```

void drawParallelogram(int a1, int a2, int b1, int b2);

Zakładamy, że **a1, a2, b1** to całkowite liczby dodatnie, a **b2** to całkowita liczba ujemna. Rysuje równoległobok oparty na wektorach **[a1,a2]** oraz **[b1,b2]** (zaczepionych we wspólnym punkcie)**.** Innymi słowy wierzchołkami równoległoboku mogą być punkty **(0,0), (a1,a2), (a1+b1, a2+b2), (b1, b2)**. Dla przykładu wywołanie:

drawParallelogram(3, 2, 2, -2);

narysuje równoległobok o wierzchołkach w punktach **(0,0), (3,2), (5, 0), (2,-2)** tak jak na rysunku poniżej:

```
   p
  ppp
pppppp
 ppp
  p
```

4. Dla kształtów bazową klasą jest klasa **Shape**. Posiada ona 3 metody:
   a. Konstruktor:
      Shape(Display *w);

   b. Abstrakcyjną metodę:
      void draw();

   Jest to metoda pozwalająca danemu kształtowi wyświetlić się. Kształt zostaje wyświetlony poprzez bibliotekę graficzną, która została podana jako argument konstruktora.

   c. Metodę pozwalającą kształtowi zmienić sposób wyświetlania się poprzez zmianę używanej biblioteki graficznej:

   void changeDisplay(Display *newDisplay)

   Po wywołaniu tej metody obiekt może zostać wyświetlony za pomocą metody **draw()** przy pomocy nowo załadowanej biblioteki graficznej.
   Cztery klasy pochodne dziedziczą z klasy **Shape**. Trzy z nich reprezentują proste kształty: **Circle**, **Parallelogram**, **Triangle**, a jedna **ComplexShape** jest kształtem złożonym.

5. Klasa **Circle** posiada konstruktor:
   Circle(Display *w, int r)
   gdzie **r** to promień koła. Implementuje również metodę rysującą **draw().**

6. Klasa **Triangle** posiada konstruktor:
   Triangle(Display *w, int a1, int a2, int b1, int b2)
   gdzie **[a1, a2]** to wektor, na którym oparty jest pierwszy bok trójkąta, a **[b1, b2]** to wektor, na którym oparty jest drugi bok trójkąta. Implementuje również metodę rysującą **draw().**

7. Klasa **Parallelogram** posiada konstruktor:
   Parallelogram(Display *w, int a1, int a2, int b1, int b2)
   gdzie **[a1, a2]** to wektor, na którym oparty jest pierwszy bok równoległoboku, a **[b1, b2]** to wektor, na którym oparty jest drugi bok równoległoboku. Implementuje również metodę rysującą **draw().**

8. Klasa **ComplexShape** posiada konstruktor:
   ComplexShape(Display* w)
   oraz metodę

   bool add(Shape *o)

   pozwalającą na dodawanie do kolekcji jednego ze zdefiniowanych wcześniej kształtów (trójkąta, koła bądź równoległoboku). Maksymalnie można dodać 5 kształtów. Kolejne próby nie będą powodowały zmiany kolekcji kształtów. Jeśli kształt udało się dodać metoda zwraca wartość prawda, w przeciwnym przypadku wartość fałsz.

   Metoda **draw()** wyświetla kolejno wszystkie obiekty dodane do kolekcji metodą **add(Shape * o)** jeden pod drugim w kolejności takiej jak zostawały dodawane. Do wyświetlania używana jest taka biblioteka rysująca: **TextDisplay** lub **GraphicalDisplay**, która została załadowana do konkretnego obiektu kolekcji (a nie głównej klasy **ComplexShape**).

Użycie:

Użycie stworzonych klas powinno wyglądać w ten sposób, że na początku tworzona jest biblioteka wyświetlająca, a następnie kształt przy użyciu tej biblioteki. Aby wyświetlić kształt należy wywołać metodę **draw()** dla utworzonego kształtu jak poniżej:

Display *w = new GraphicalDisplay();

Shape * o1 = new Triangle(w, 1, 1, 1, -1);

o1->draw();

Rozwiązanie:

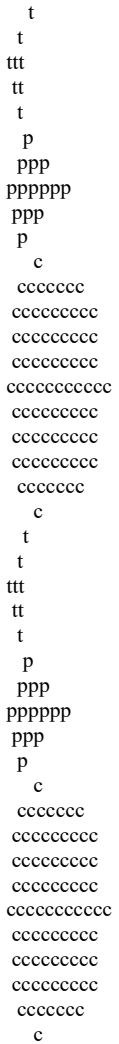
W rozwiązaniu umieść każdą klasę w osobnych plikach z rozszerzeniami .h oraz .cpp. Na platformę elf2.pk.edu.pl do folderu: ***Programy – projekt*** prześlij komplet spakowanych plików w folder o nazwie ***Nazwisko1_Nazwisko_2_projekt***:

display.h, display.cpp, graphicalDisplay.h, graphicalDisplay.cpp, textDisplay.h, textDisplay.cpp, shape.h, shape.cpp, circle.h, circle.cpp, triangle.h, triangle.cpp, parallelogram.h, parallelogram.cpp, complexShape.h, complexShape.cpp oraz *main.cpp*

Test:

Przykład (funkcja main oraz wyjście) znajduje się w osobnych plikach dołączonych do zadania.

```
#include <iostream>
#include "graphicalDisplay.h"
#include "textDisplay.h"
#include "circle.h"
#include "triangle.h"
#include "parallelogram.h"
#include "complexShape.h"
#include "display.h"
#include "shape.h"

using namespace std;

int main(){

	Display *w1 = new GraphicalDisplay();
	Display *w2 = new TextDisplay();
	Shape * o1 = new Triangle(w1, 3, 2, 2,-2);
	Shape * o2 = new Parallelogram(w1, 3, 2, 2,-2);
	Shape * o3 = new Circle(w1, 5);
	ComplexShape * o4 = new ComplexShape(w1);
	o4->add(o1);
	o4->add(o2);
	o4->add(o3);
	o4->draw();
	o1->draw();
	o2->draw();
	o3->draw();

	o1->changeDisplay(w2);
	o2->changeDisplay(w2);
	o3->changeDisplay(w2);
	o4->changeDisplay(w2);
	o4->draw();
	o1->draw();
	o2->draw();
	o3->draw();

	delete o1;
	delete o2;
	delete o3;
	delete o4;
	delete w2;
	delete w1;

	return 0;
}
```



Drawing a triangle from vectors (3, 2), (2, -2).
Drawing a parallelogram from vectors (3, 2), (2, -2).
Drawing a circle with radius 5.
Drawing a triangle from vectors (3, 2), (2, -2).
Drawing a parallelogram from vectors (3, 2), (2, -2).
Drawing a circle with radius 5.