

Zadanie 1

Sprawdź działanie polimorfizmu na przykładzie tablicy figur. * Utwórz klasę bazową **Shape**. Klasa ta powinna posiadać metodę służącą do drukowania nazwy obiektu np.:

```
void PrintName()
{
    cout << "class Shape\n";
}
```

- Utwórz klasy **Circle**, **Square**, **Triangle** będące obiektami potomnymi klasy **Shape**. Obiekty te powinny być opisane przez wierzchołki (można wykorzystać klasę z zajęć poprzednich **Wektor2D**) i posiadać metody do obliczania pola figury.
- Zmodyfikuj klasę **Shape** przez dodanie abstrakcyjnej metody do liczenia pola figury np.:

```
virtual double Area() = 0;
```

- Sprawdź, czy możesz teraz utworzyć obiekt typu **Shape** lub np. **Circle**.
- Zmodyfikuj metody **Area()** obiektów **Circle**, **Square**, **Triangle** tak aby były one wirtualne oraz dodaj do nich wirtualne metody **PrintName()**.
- Utwórz tablicę wskaźników do obiektów typu **Shape** i zainicjalizuj je obiektami **Circle**, **Square**, **Triangle** np.:

```
Shape *tabp[3];
tabp[0] = new Circle(...);
tabp[1] = new Triangle(...);
tabp[2] = new Square(...);
for ( int i=0; i<3; ++i)
{
    tabp[i]->PrintName();
    cout << tabp[i]->Area() << endl;
}
```

- Sprawdź co się stanie gdy usunie się modyfikator **virtual** przy metodach obiektów **Circle**, **Square**, **Triangle**.

- Sprawdź działanie funkcji **dynamic_cast<T*>()** np. do policzenia ile obiektów w danej kolekcji jest typu **Circle** (pamiętaj o włączeniu opcji kompilatora RTTI):

```
if( dynamic_cast<Circle*>( tabp[i] ) )
{
    ++count;
}
```

- Czy wiesz dlaczego w punkcie poprzednim używaliśmy **dynamic_cast** a nie **static_cast**?

Zadanie 2

Zmodyfikuj program tak aby każda klasa była umieszczona w oddzielnym pliku .h i .cpp