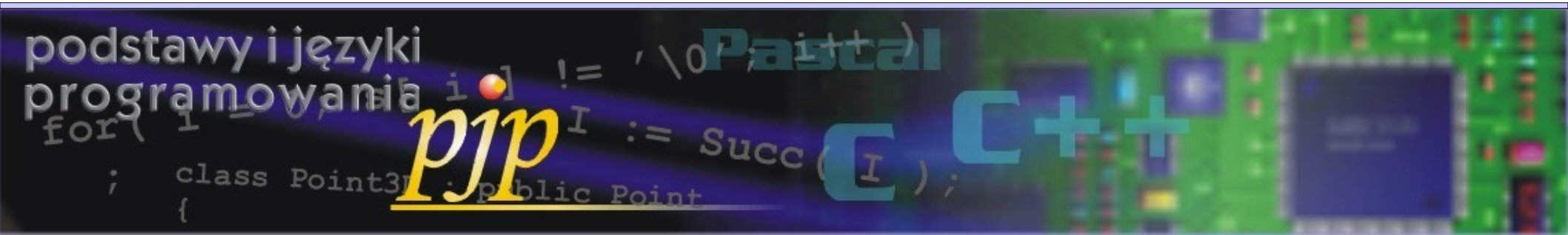
Projektowanie i programowanie obiektowe

Roman Simiński
roman.siminski@us.edu.pl
roman@siminskionline.pl
programowanie.siminskionline.pl

Wzorce projektowe

Wybrane wzorce strukturalne: Dekorator



Dekorator
Decorator Pattern

Obiekty GUI

`Window`

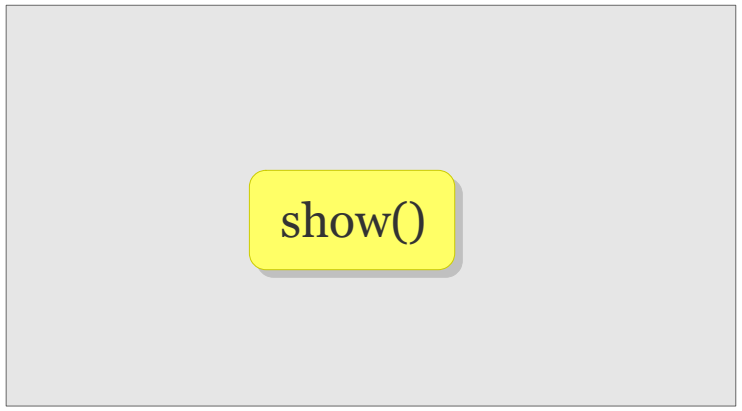
`WindowWithTitle`

`WindowWithIcon`

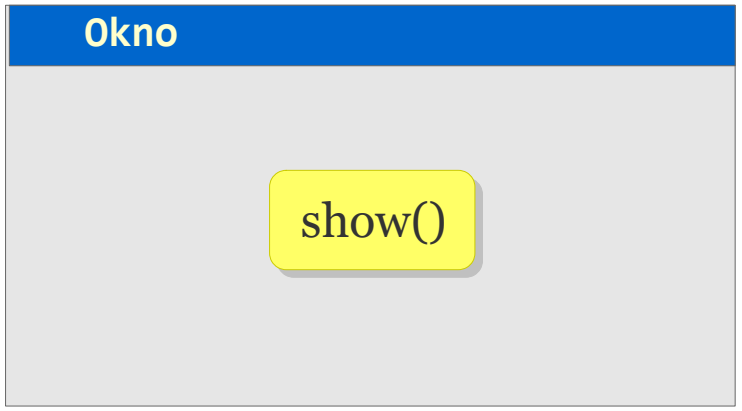
`WindowWithScrollBars`

Obiekty GUI

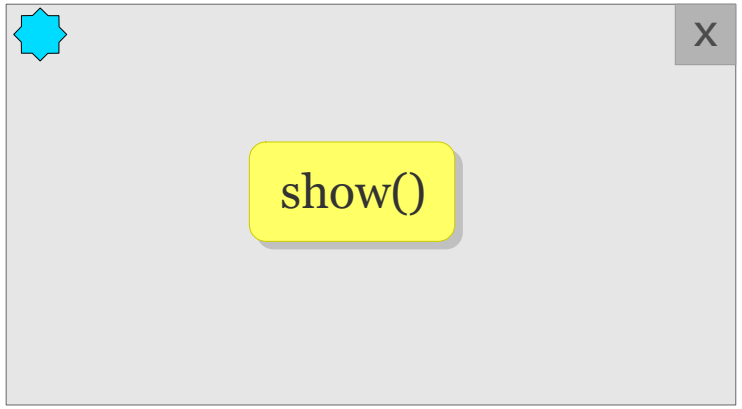
`Window`



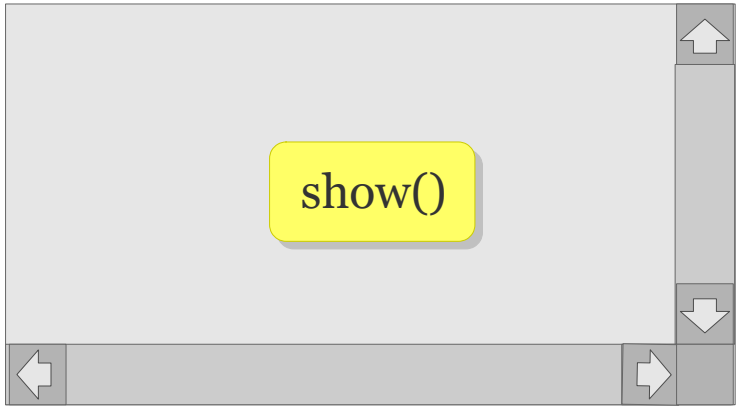
`WindowWithTitle`



`WindowWithIcon`

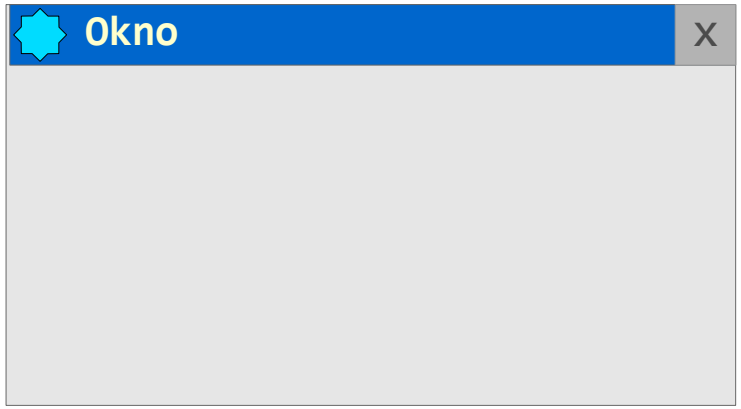


`WindowWithScrollBars`

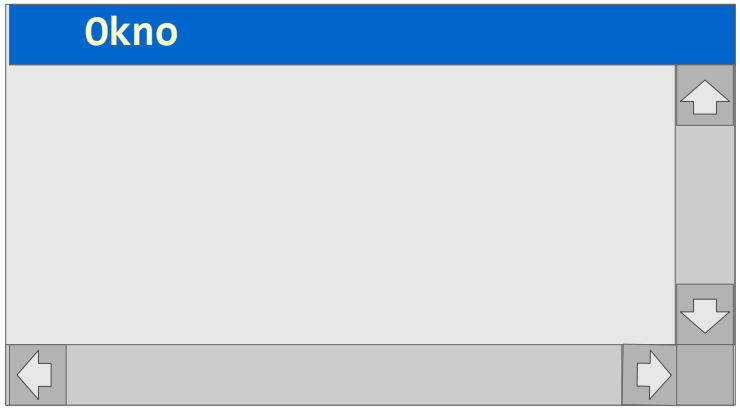


Obiekty GUI

`WindowWithTitleAndIcons`

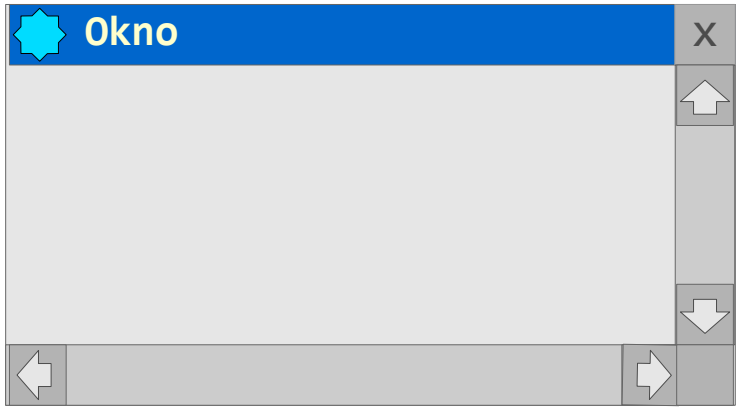


`WindowWithTitleAndScrollBars`



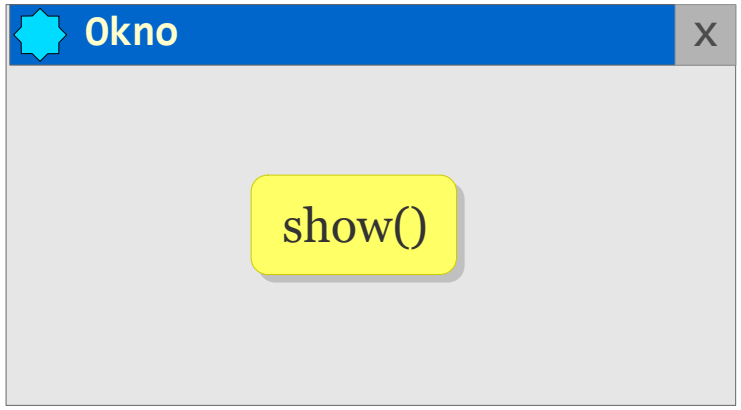
`WindowWithIconAndScrollBars`

`WindowWithEverything`

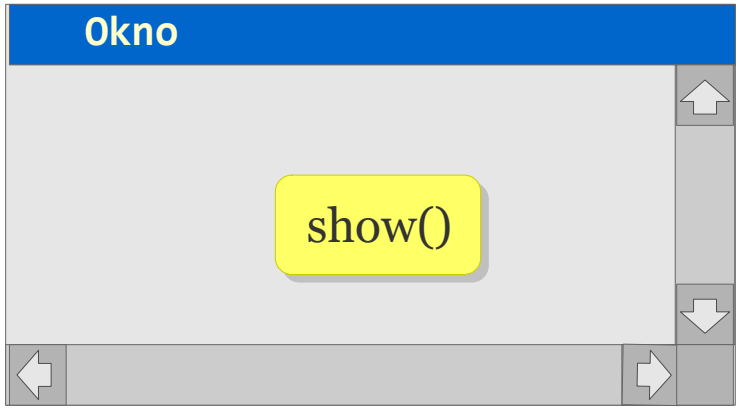


Obiekty GUI

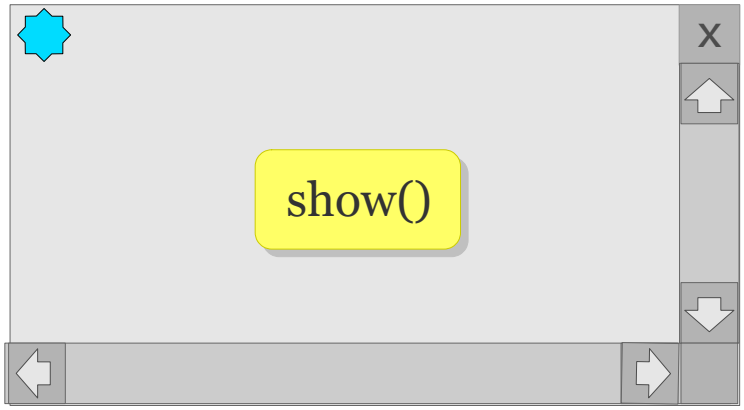
`WindowWithTitleAndIcons`



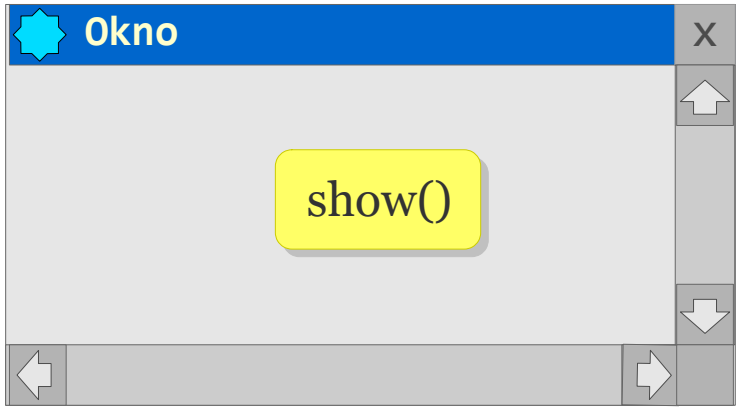
`WindowWithTitleAndScrollBars`



`WindowWithIconAndScrollBars`



`WindowWithEverything`



Jak opisać różne warianty wyświetlania okna?

```
class Window
{
  . . .
  public void show() { . . . }
}

class WindowWithTitle : Window
{
  . . .
  public void show()
  {
    Window.show();
    . . .
  }
}

class WindowWithIcons : Window
{
  . . .
  public void show()
  {
    Window.show();
    . . .
  }
}
```

Jest przecież dziedziczenie...

Jak opisać różne warianty wyświetlania okna?

```
class WindowWithTitleAndIcons : WindowWithTitle
{
  . . .
  public void show()
  {
    WindowWithTitle.show();
    . . .
  }
}

class WindowWithIconsAndScrollBars : WindowWithIcons
{
  . . .
  public void show()
  {
    WindowWithIcons.show();
    . . .
  }
}
```

Jest przecież dziedziczenie...

Jak opisać różne warianty wyświetlania okna?

```
class WindowWithTitleAndIconsAndScrollBars : WindowWithTitleAndIcons
{
  . . .
  public void show()
  {
    WindowWithTitleAndIcons.show();
    . . .
  }
}
```

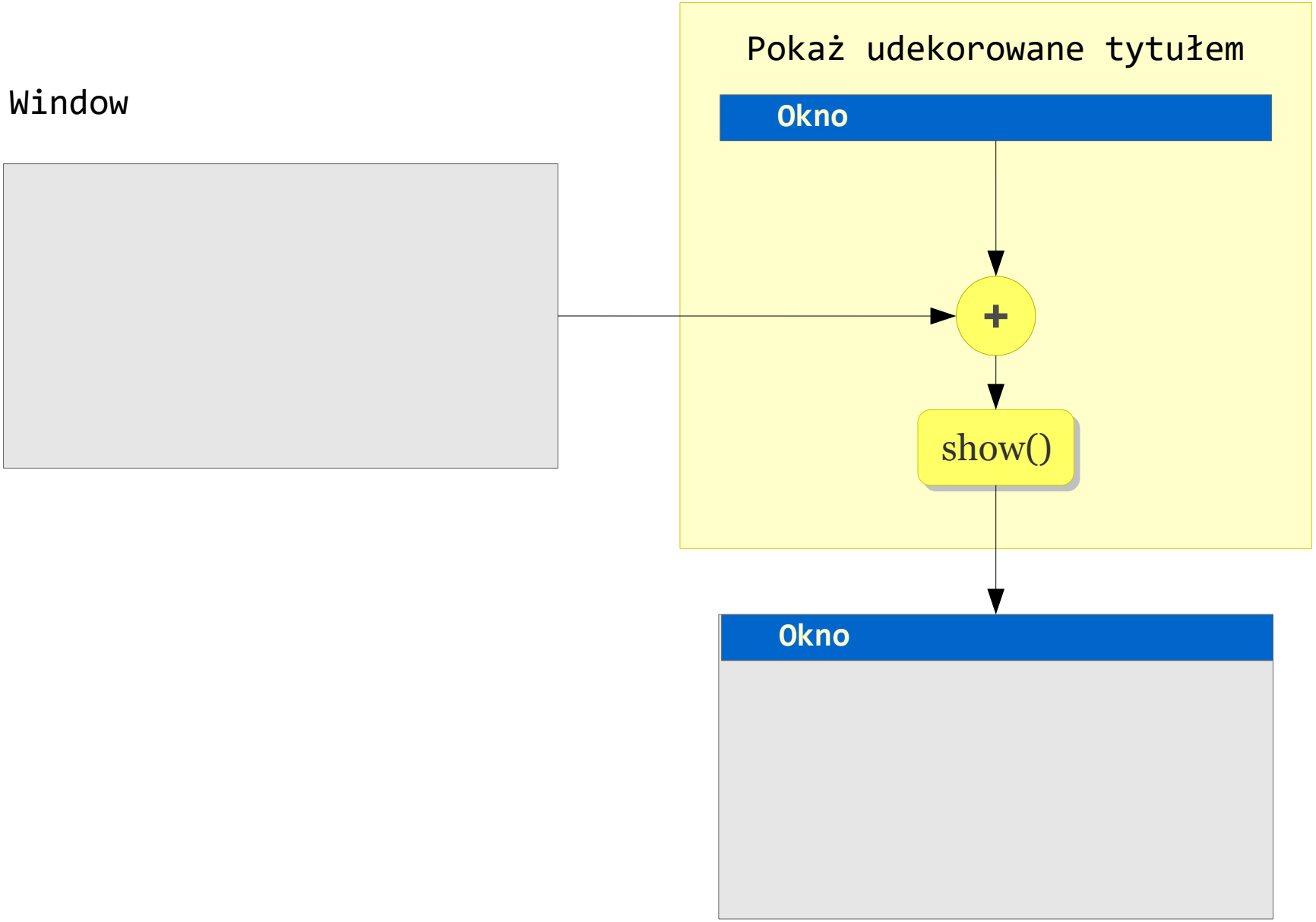
Duuuuużooo klas, a może dziedziczenie wielobazowe?

Jak opisać różne warianty wyświetlania okna?

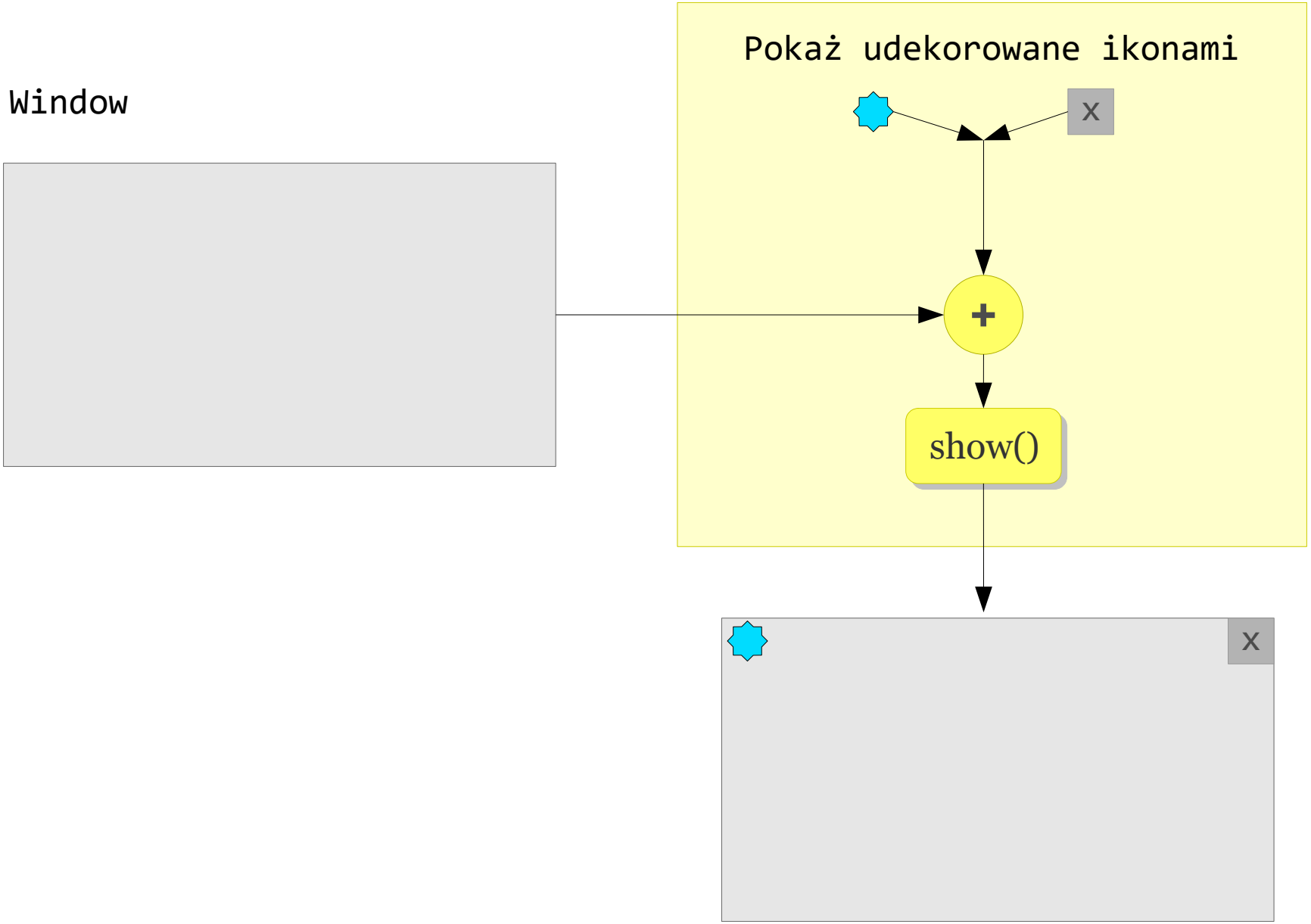
```
class WindowWithTitleAndIconsAndScrollBars
: WindowWithTitle, WindowWithIcons, WindowWithScrollBars
{
  . . .
  public void show()
  {
    WindowWithTitleAndIcons.show();
    WindowWithIcons.show();
    WindowWithscrollBars.show();
    . . .
  }
}
```

Dalej dużo klas, zależności pomiędzy nimi są statycznie definiowane via dziedziczenie.
No i język musi wspierać dziedziczenie wielobazowe...

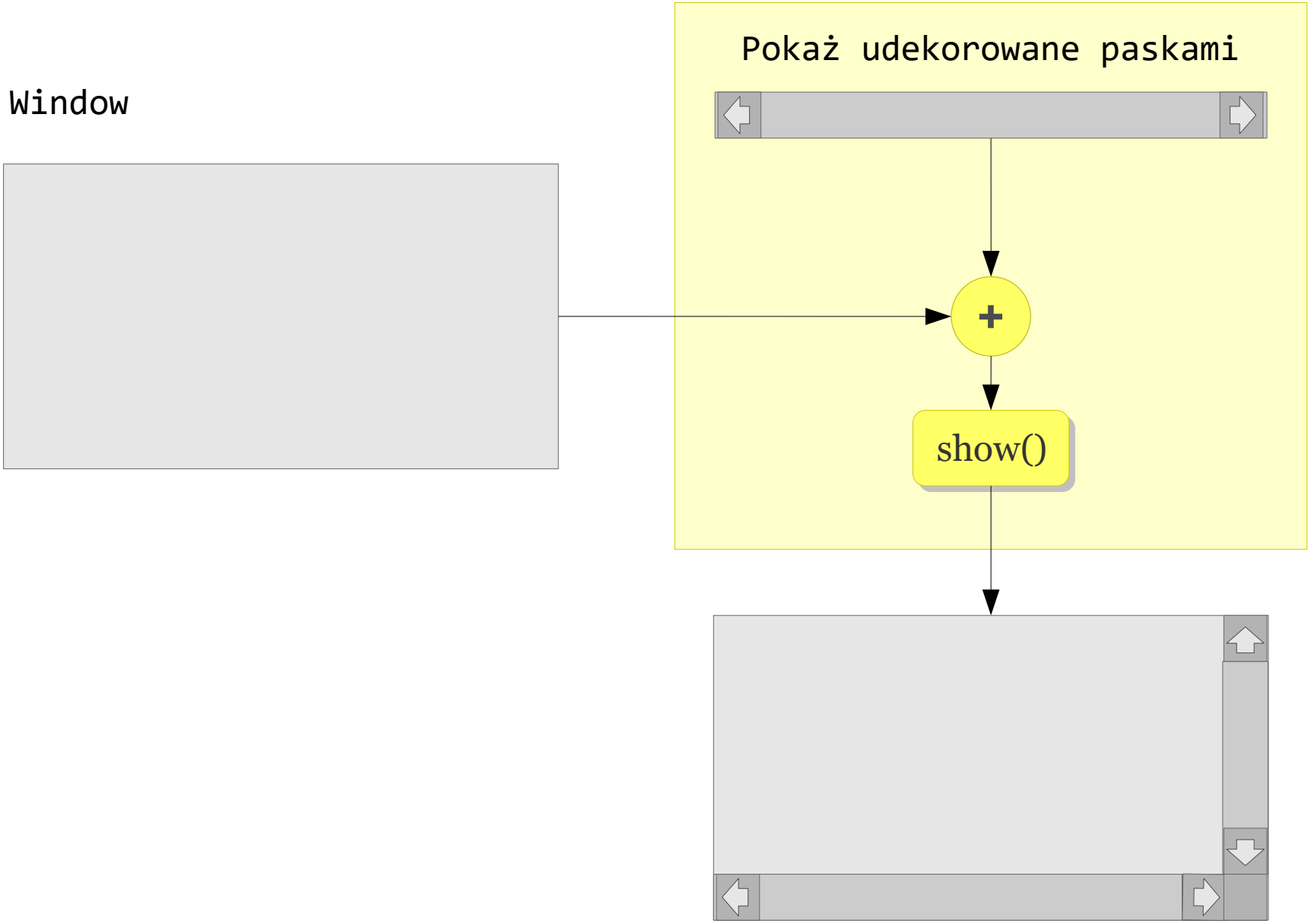
Udekoruj tytułem i pokaż



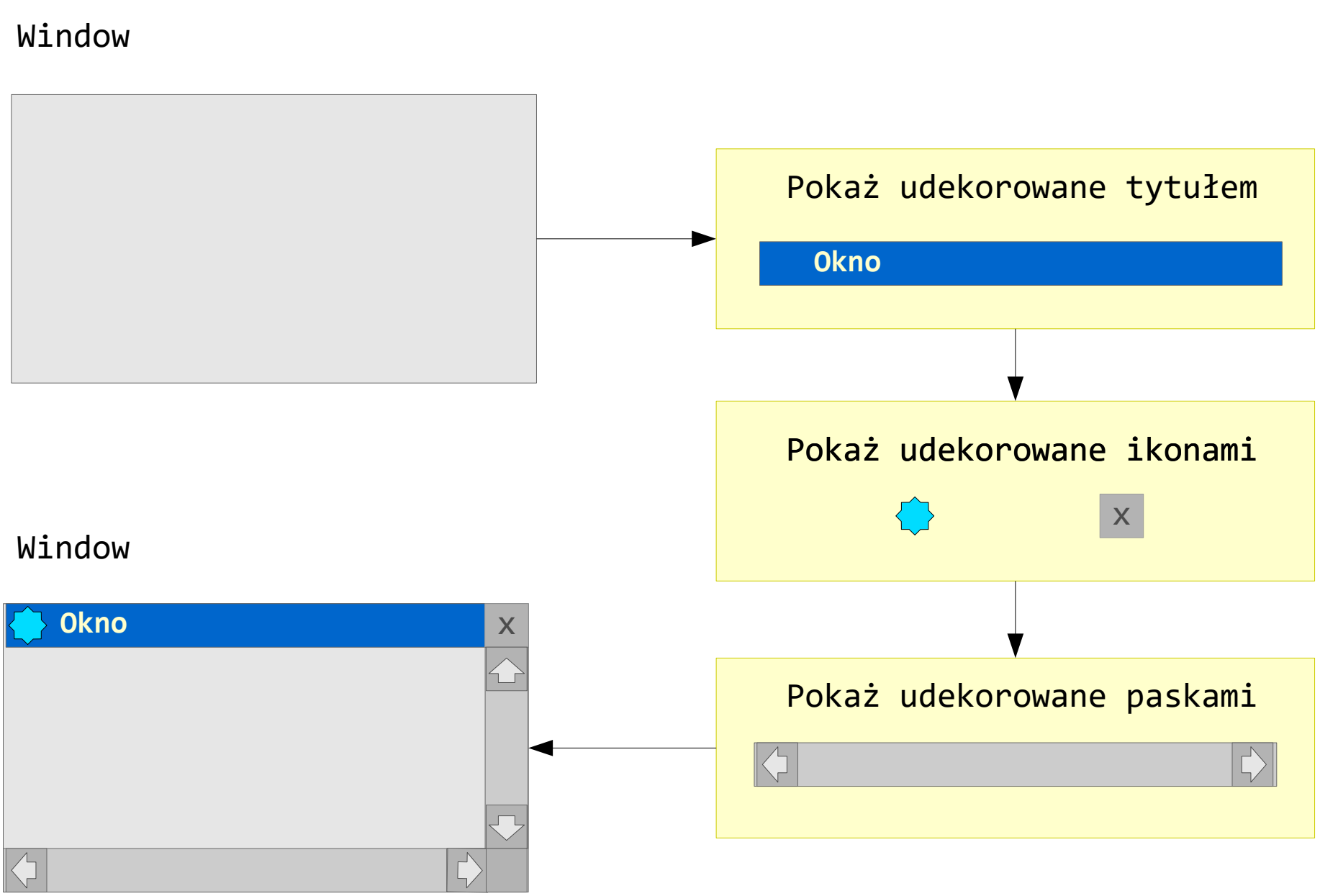
Udekoruj ikonami i pokaż



Udekoruj paskami przewijania i pokaż



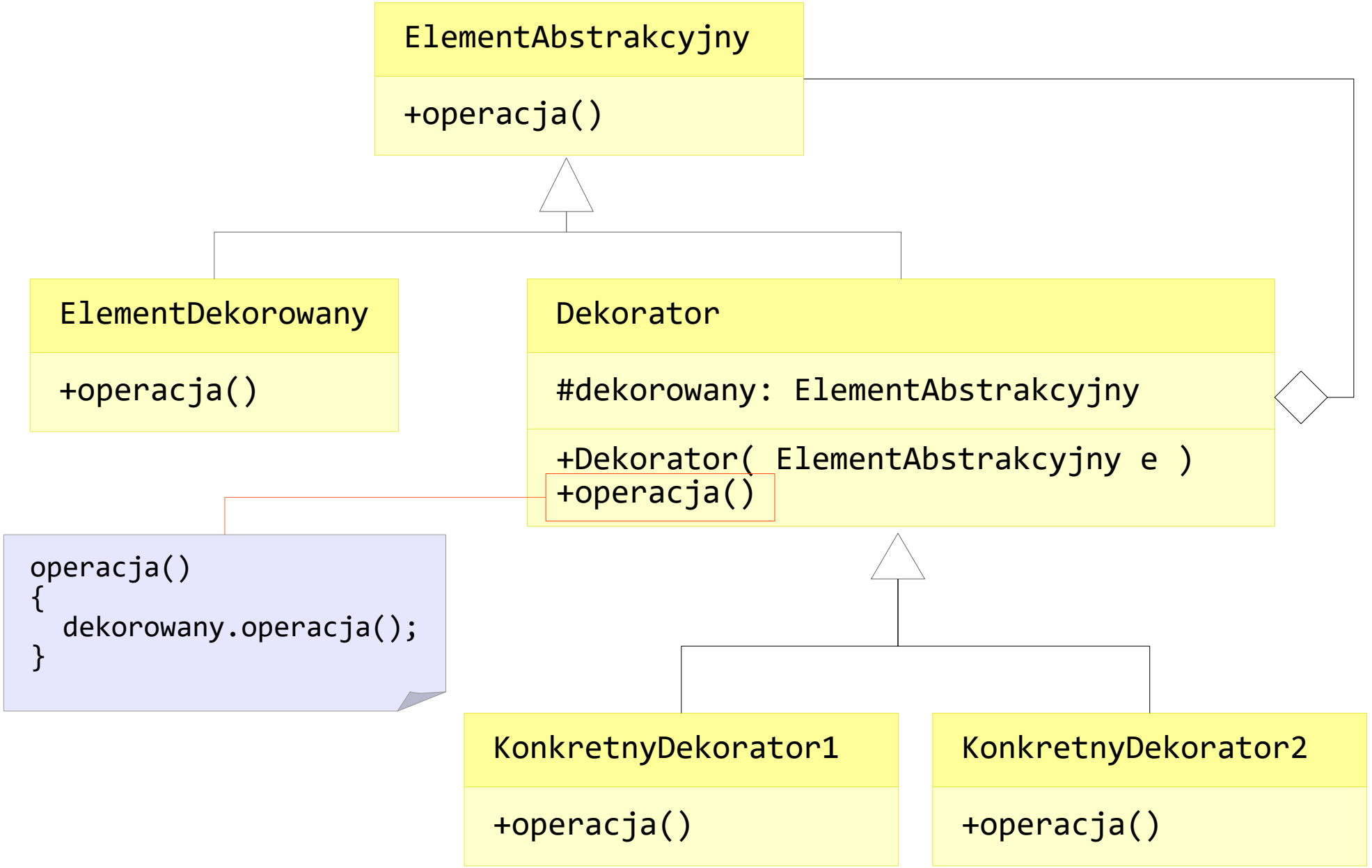
Udekoruj wszystkim i pokaż



Dekorator

- Wzorzec ***Dekorator*** pozwala nam na modyfikowanie zachowania obiektu.
- Modyfikowanie oznacza możliwość zmiany działania funkcji składowej (metody), oznacza to zazwyczaj rozszerzenie zakresu operacji wykonywanych przez funkcje składowe.
- Obiekt jest przekazywany do *dekoratora*, który ów obiekt opakowuje, dostarczając operacji modyfikującej/rozszerzającej działanie obiektu.
- Dekorator wspiera SOLID — realizuje zasadę Open/Close, pozwala pisać klasy otwarte na rozbudowę a zamknięte na modyfikacje.
- Najważniejszą zaletą jest możliwość dynamicznej modyfikacji zachowania bez wykorzystania dziedziczenia.

Dekorator, schemat UML



Przykładowa implementacja dekoratora w języku Java

Element abstrakcyjny i konkretny, dekorowany

```
abstract class Window
{
  abstract public void show();
}


class DialogWindow extends Window
{
    public void show()
    {
        System.out.println( "DialogWindow.show()" );
    }
}
```

Klasa bazowego, abstrakcyjnego dekoratora

```
abstract class WindowDecorator extends Window
{
    WindowDecorator( Window window )
    {
        this.window = window;
    }

    @Override
    public void show()
    {
        window.show();
    }
    protected Window window;
}
```

Klasa konkretnego dekoratora, tytuł

```
class TitleDecorator extends WindowDecorator
{
    public TitleDecorator( Window window )
    {
        super( window );
    }

    public void show()
    {
        super.show();
        System.out.println( "+TitleDecorator.show()" );

    }
}
```

Klasa konkretnego dekoratora, ikony

```
class IconsDecorator extends WindowDecorator
{
    public IconsDecorator( Window window )
    {
        super( window );
    }
    public void show()
    {
        super.show();
        System.out.println( "+IconsDecorator.show()" );

    }
}
```

Wykorzystanie dekoratorów

```
Window emptyWindow = new DialogWindow();
emptyWindow.show();

Window windowWithIcons = new IconsDecorator( new DialogWindow() );
windowWithIcons.show();

Window windowWithTitle = new TitleDecorator( new DialogWindow() );
windowWithTitle.show();

Window myWindow = new IconsDecorator( new TitleDecorator( new DialogWindow() ) );
myWindow.show();
```

```
DialogWindow.show()
DialogWindow.show()
+IconsDecorator.show()
DialogWindow.show()
+TitleDecorator.show()
DialogWindow.show()
+TitleDecorator.show()
+IconsDecorator.show()
```