



Daniel Krasnokucki

LEKSYKON KIESZONKOWY

# Wzorce projektowe

**ODKRYJ WZORCE PROJEKTOWE  
I SPOSOBY ICH STOSOWANIA!**

- Poznaj najlepsze wzorce projektowe
- Naucz się je stosować w praktyce
- Dowiedz się, jak wybrać właściwy wzorec

**Helion** 

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Studio Gravite / Olsztyn  
Obarek, Pokoński, Pazdrijowski, Zaprucki

Grafika na okładce została wykorzystana za zgodą Shutterstock.com

Wydawnictwo HELION  
ul. Kościuszki 1c, 44-100 GLIWICE  
tel. 32 231 22 19, 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/wzoplk>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-3880-7

Copyright © Helion 2017

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

<b>Wstęp</b>	<b>7</b>
<b>Wprowadzenie</b>	<b>9</b>
Relacje pomiędzy wzorcami	10
<b>Wzorce konstrukcyjne (kreacyjne)</b>	<b>11</b>
Budowniczy	13
Fabryka abstrakcyjna	20
Metoda wytwórcza (metoda fabrykująca)	30
Prototyp	35
Singleton	40
<b>Wzorce strukturalne</b>	<b>43</b>
Adapter (wrapper)	45
Dekorator	48
Fasada	52
Kompozyt	55
Most	59
Pełnomocnik (proxy, substytut)	63
Pyłek	66
<b>Wzorce operacyjne (czynnościowe)</b>	<b>73</b>
Interpreter	75
Iterator (kursor)	82
Łańcuch zobowiązań	87
Mediator	92
Metoda szablonowa	98
Obserwator	101
Odwiedzający (wizytator)	106
Pamiętka (znacznik)	112
Polecenie	117
Stan	121
Strategia (polityka)	125



# Wzorce operacyjne (czynnościowe)

INTERPRETER

ITERATOR (KURSOR)

ŁAŃCUCH ZOBOWIĄZAŃ

MEDIATOR

METODA SZABLONOWA

OBSERWATOR

ODWIEDZAJĄCY (WIZYTATOR)

PAMIĄTKA (ZNACZNIK)

POLECENIE

STAN

STRATEGIA (POLITYKA)

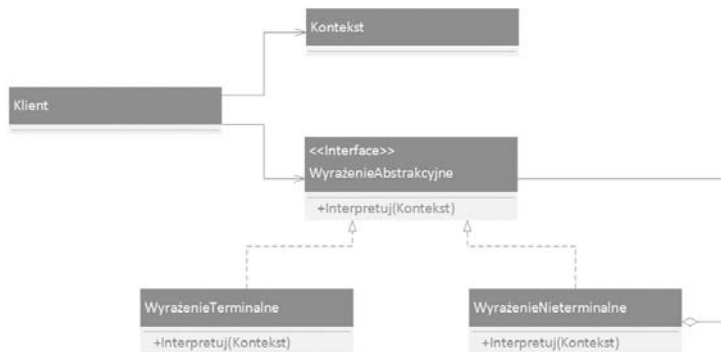
Wzorce operacyjne (czynnościowe) opisują zachowanie obiektów, komunikację pomiędzy nimi i ich odpowiedzialność.



# Interpreter

## Przeznaczenie

Jego zadaniem jest określenie opisu gramatyki języka oraz stworzenie interpretera, który będzie wykorzystywał ten opis do interpretowania zdań z tego języka.



Rysunek 14. Diagram UML — interpreter

## Implementacja

1. Jeden z obiektów odpowiedzialny jest za przetrzymywanie danych do interpretacji.
2. Tworzymy abstrakcję, która interpretuje polecenia.
3. Dla poszczególnych przypadków tworzymy konkretne obiekty interpretujące dane i zgodne z naszą abstrakcją.
4. Abstrakcja wraz z obiektami konkretnych implementacji tworzy wzorzec metody szablonowej.

## Przykłady zastosowania

- interpretacja języka.

## Przykładowy kod w C++

```
using namespace std;

class Tysiace;
class Setki;
class Dziesiatki;
class Jednosci;

class InterpreterLiczbrzyskich
{
```

```

public:
    InterpreterLiczbyRzymskich();
    explicit InterpreterLiczbyRzymskich(int) {}
    int interpretuj(string);

    virtual void interpretuj(string &wejscie, int &wynik)
    {
        int index;
        index = 0;
        string doPorownania = wejscie.substr(0, 2);
        if (doPorownania == dziewiec())
        {
            wynik += 9 * mnoznik();
            index += 2;
        }
        else if (doPorownania == cztery())
        {
            wynik += 4 * mnoznik();
            index += 2;
        }
        else
        {
            if (wejscie[0] == piec())
            {
                wynik += 5 * mnoznik();
                index = 1;
            }
            else
            { index = 0; }
            for (int end = index + 3; index < end; index++)
            {
                if (wejscie[index] == jeden())
                {
                    wynik += 1 * mnoznik();
                }
                else
                {
                    break;
                }
            }
            int length = wejscie.length() - index;
            wejscie = wejscie.substr(index, length);
        }
        virtual ~InterpreterLiczbyRzymskich(){}
protected:
    virtual char jeden()
    {
        return 0;
    }

    virtual string cztery()
    {
        return nullptr;
    }
}

```



```

    virtual char piec()
    {
        return 0;
    }

    virtual string dziewiec()
    {
        return nullptr;
    }

    virtual int mnoznik()
    {
        return 0;
    }

private:
    InterpreterLiczBRzymskich *tysiace;
    InterpreterLiczBRzymskich *setki;
    InterpreterLiczBRzymskich *dziesiatki;
    InterpreterLiczBRzymskich *jednosci;
};

class Tysiace : public InterpreterLiczBRzymskich
{
public:
    explicit Tysiace(int) : InterpreterLiczBRzymskich(1) {}
protected:
    char jeden() override
    {
        return 'M';
    }
    string cztery() override
    {
        return "";
    }
    char piec() override
    {
        return '\0';
    }
    string dziewiec() override
    {
        return "";
    }
    int mnoznik() override
    {
        return 1000;
    }
};

class Setki : public InterpreterLiczBRzymskich
{
public:
    explicit Setki(int) : InterpreterLiczBRzymskich(1) {}
protected:
    char jeden() override

```

```

    {
        return 'C';
    }
    string cztery() override
    {
        return "CD";
    }
    char piec() override
    {
        return 'D';
    }
    string dziewiec() override
    {
        return "CM";
    }
    int mnoznik() override
    {
        return 100;
    }
};

class Dziesiatki : public InterpreterLiczBRzyskich
{
public:
    explicit Dziesiatki(int) : InterpreterLiczBRzyskich(1) {}
protected:
    char jeden() override
    {
        return 'X';
    }
    string cztery() override
    {
        return "XL";
    }
    char piec() override
    {
        return 'L';
    }
    string dziewiec() override
    {
        return "XC";
    }
    int mnoznik() override
    {
        return 10;
    }
};

class Jednosci : public InterpreterLiczBRzyskich
{
public:
    Jednosci(int) : InterpreterLiczBRzyskich(1) {}
protected:
    char jeden() override
    {
        return 'I';
    }
};

```

```

    }
    string cztery() override
    {
        return "IV";
    }
    char piec() override
    {
        return 'V';
    }
    string dziewiec() override
    {
        return "IX";
    }
    int mnoznik() override
    {
        return 1;
    }
};

```

```

InterpreterLiczBRzymskich::InterpreterLiczBRzymskich()
{
    tysiace = new Tysiace(1);
    setki = new Setki(1);
    dziesiatki = new Dziesiatki(1);
    jednosci = new Jednosci(1);
}

```

```

int InterpreterLiczBRzymskich::interpretuj(string liczba)
{
    if (liczba.empty())
    {
        return 0;
    }
    int total;
    total = 0;
    tysiace->interpretuj(liczba, total);
    setki->interpretuj(liczba, total);
    dziesiatki->interpretuj(liczba, total);
    jednosci->interpretuj(liczba, total);

    return total;
}

```

```

int main()
{
    InterpreterLiczBRzymskich interpreter;
    string liczba = "MMCXLVI"; // przykładowa liczba rzymska
    cout << "Odpowiednik dziesiętny " << liczba << " to: " <<
    interpreter.interpretuj(liczba);
    system("pause");
}

```

## Przykładowy kod w C#

```
// Aplikacja rozpoznająca liczby rzymskie.
class Kontekst
{
    public Kontekst(string input)
    {
        Wejscie = input;
    }

    public string Wejscie { get; set; }

    public int Wyjscie { get; set; }
}

abstract class Wyrazenie
{
    public void Interpretuj(Kontekst kontekst)
    {
        if (kontekst.Wejscie.Length == 0)
            return;

        if (kontekst.Wejscie.StartsWith(Dziewiec()))
        {
            kontekst.Wyjscie += (9 * Mnoznik());
            kontekst.Wejscie = kontekst.Wejscie.Substring(2);
        }
        else if (kontekst.Wejscie.StartsWith(Cztery()))
        {
            kontekst.Wyjscie += (4 * Mnoznik());
            kontekst.Wejscie = kontekst.Wejscie.Substring(2);
        }
        else if (kontekst.Wejscie.StartsWith(Piec()))
        {
            kontekst.Wyjscie += (5 * Mnoznik());
            kontekst.Wejscie = kontekst.Wejscie.Substring(1);
        }

        while (kontekst.Wejscie.StartsWith(Jeden()))
        {
            kontekst.Wyjscie += (1 * Mnoznik());
            kontekst.Wejscie = kontekst.Wejscie.Substring(1);
        }

        protected abstract string Jeden();
        protected abstract string Cztery();
        protected abstract string Piec();
        protected abstract string Dziewiec();
        protected abstract int Mnoznik();
    }
}

// Wyrażenia terminalne:
class Tysiace : Wyrazenie
{

```

```

    protected override string Jeden() { return "M"; }
    protected override string Cztery() { return " "; }
    protected override string Piec() { return " "; }
    protected override string Dziewiec() { return " "; }
    protected override int Mnoznik() { return 1000; }
}

class Setki : Wyrazenie
{
    protected override string Jeden() { return "C"; }
    protected override string Cztery() { return "CD"; }
    protected override string Piec() { return "D"; }
    protected override string Dziewiec() { return "CM"; }
    protected override int Mnoznik() { return 100; }
}

class Dziesiatki : Wyrazenie
{
    protected override string Jeden() { return "X"; }
    protected override string Cztery() { return "XL"; }
    protected override string Piec() { return "L"; }
    protected override string Dziewiec() { return "XC"; }
    protected override int Mnoznik() { return 10; }
}

class Jednosci : Wyrazenie
{
    protected override string Jeden() { return "I"; }
    protected override string Cztery() { return "IV"; }
    protected override string Piec() { return "V"; }
    protected override string Dziewiec() { return "IX"; }
    protected override int Mnoznik() { return 1; }
}

static class Program
{
    static void Main()
    {
        const string rzymskie = "MMCXLVI";
        var kontekst = new Kontekst(rzymskie);
        // Budujemy drzewo, które będzie parsowane:
        var drzewo = new List<Wyrazenie> { new Tysiace(), new Setki(),
        ↪ new Dziesiatki(), new Jednosci() };
        // Interpretuj
        foreach (var wyrazenie in drzewo)
        {
            wyrazenie.Interpretuj(kontekst);
        }
        Console.WriteLine($"{rzymskie} = {kontekst.Wyjscie}");
        Console.ReadKey();
    }
}

```



# PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW  
w działający bankomat!

**Dowiedz się więcej i dołącz już dzisiaj!**

<http://program-partnerski.helion.pl>

# Wzorce projektowe

**Gdy stoisz przed nowym problemem programistycznym**, nierzadko łamiesz sobie głowę nad właściwym rozwiązaniem. Całymi godzinami starasz się wybrać najlepszą drogę. Zupełnie niepotrzebnie, bo ktoś niemal na pewno zrobił to już wcześniej, przetestował swoje rozwiązanie i wyeliminował ewentualne błędy, a odkryty przez niego sposób stał się obowiązującym wzorcem projektowym, wykorzystywanym z powodzeniem przez rzesze programistów.

**Jeśli chcesz korzystać ze sprawdzonych wzorców**, sięgnij po ten leksykon kieszonkowy. W telegraficznym skrócie prezentuje on najlepsze i najczęściej używane wzorce projektowe, sposoby ich implementacji i przykłady stosowania, a jeśli to Ci nie wystarczy, możesz też zaznajomić się z praktyczną realizacją tych wzorców w językach C++ i C#. Niezależnie od tego, czy temat jest dla Ciebie nowy, czy chcesz tylko uporządkować swoją wiedzę, trafisz na odpowiednią książkę!

- Różne rodzaje wzorców i zależności występujące między nimi
- Wzorce konstrukcyjne, strukturalne i operacyjne
- Przeznaczenie i implementacja poszczególnych wzorców
- Praktyczne zastosowanie wzorców projektowych
- Przykładowe kody w C++ i C#

**Nie wyważaj otwartych drzwi — sięgnij po odpowiedni wzorzec!**

**Helion**

księgarnia internetowa



<http://helion.pl>

zamówienia telefoniczne



**0 801 339900**



**0 601 339900**

Helion SA  
ul. Kościuszki 1c, 44-100 Gliwice  
tel.: 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
<http://helion.pl>

Sprawdź najnowsze promocje:  
● <http://helion.pl/promocje>  
Książki najchętniej czytane:  
● <http://helion.pl/bestsellery>  
Zamów informacje o nowościach:  
● <http://helion.pl/nowosci>

sięgnij po **WIĘCEJ**



**KOD KORZYŚCI**

ISBN 978-83-283-3880-7



9 788328 338807

Informatyka w najlepszym wydaniu

cena: 24,90 zł