Wzorce(szablony)

Jakub Jaroszek
Ewa Czarnożyńska

Wzorce/szablony(ang. templates)

- Szablony to klasa bądź funkcja zaimplementowana dla nieokreślonego typu.
- Typ jest określany w sposób jawny bądź niejawny w miejscu zastosowania szablonu
- Szablony definiuje się umieszczając przed definicją funkcji lub klasy frazę **template** z listą parametrów w nawiasach kątowych. template <typename T, typename D>

```
void poka(int a, double b) {
    cout << a << " , " << b << endl;

}

int main() {

    poka(3, 5.7);
    poka(3.0, 5.0);
    poka(12.123, 4);
    poka(12.123, 4);
```

```
3 , 5.7
3 , 5
12 , 4
12 , 4
3 , 5.7
```

```
template <typename T, typename D>
void poka(T a, D b) {
    cout << a << " , " << b << endl;

}

int main() {

    poka(3, 5.7);
    poka(3.0, 5.0);
    poka(12.123, 4);
    poka(12.123, 4);
```

```
3 , 5.7
3 , 5
12.123 , 4
12.123 , 4
3 , 5.7
```

Ale po co jak można zamienić oba typy na double

To samo? To samo!

```
}
//template <typename T, typename D>
void poka(double a, double b) {
    cout << a << " , " << b << endl;

}

int main() {

    poka(3, 5.7);
    poka(3.0, 5.0);
    poka(12.123, 4);
    poka(12.123, 4);
```

```
3 , 5.7
3 , 5
12.123 , 4
12.123 , 4
3 , 5.7
```

```
template <typename T, typename D>
void poka(T a, D b) {
    cout << a << " , " << b << endl;

}

int main() {

    poka(3, 5.7);
    poka(3.0, 5.0);
    poka(12.123, 4);
    poka(12.123, 4);
```

```
3 , 5.7
3 , 5
12.123 , 4
12.123 , 4
3 , 5.7
```

A co teraz?

```
void poka(double a, double b) {
    cout << a << " , " << b << endl;

}

int main() {

    poka(3, 5.7);
    poka(3.0, 5.0);
    poka(12.123, 4);
    poka(12.123, 4);
    poka("jestem ", " string");
```

Program się wysypie

```
template <typename T, typename D>
void poka( T a, D b) {
    cout << a << " , " << b << endl;

}

int main() {

    poka(3, 5.7);
    poka(3.0, 5.0);
    poka(12.123, 4);
    poka(12.123, 4);
    poka("jestem ", " string");
```

```
3 , 5.7
3 , 5
12.123 , 4
12.123 , 4
jestem  ,  string
```

Przykładowy wzorzec funkcji

```cpp
template <typename T, typename D>
void pokaz(T  a, D b) {

    cout << a << " , " << b << endl;
    cout << " my type is : " << typeid(a).name() << endl;
    cout << " my type is : " << typeid(b).name() << endl;
}


int main() {
    //różne możliwości wywołania funkcji

    // void pokaz< typename T>(T a, D b)

    pokaz(3, 5.7); //wywolanie wzorca pokaz<int> w sposób niejawny (dedukcja)
    pokaz(3.0, 5.0); //wywolanie wzorca pokaz<double> w sposób niejawny (dedukcja)
    pokaz<>(3.0, 5.0); //wywolanie wzorca pokaz<double> w sposób niejawny (dedukcja)
    pokaz<double>(12.123, 4); //wywolanie wzorca pokaz<double> w sposób jawny (bez dedukcji)
    pokaz("jestem ", " string"); //wywolanie wzorca pokaz<string> w sposób niejawny (dedukcja)
    pokaz<double,double>(12.123, 4); //wywolanie wzorca pokaz<double> w sposób jawny (bez dedukcji)
```

```
3 , 5.7
 my type is : int
 my type is : double
3 , 5
 my type is : double
 my type is : double
3 , 5
 my type is : double
 my type is : double
12.123 , 4
 my type is : double
 my type is : int
jestem   ,  string
 my type is : char const *
 my type is : char const *
12.123 , 4
 my type is : double
 my type is : double
```

Wzorce klas

Możliwość napisania ogólnej klasy, parametrem której, może być Typ lub inna klasa.

Przykład wzorca klasy

T, T2 to symbole zastępcze typu zmiennej, możliwe jest użycie wielu typów.
Oprócz typów i klas można użyć stałych, napisów , nazw funkcji

```
template <class T>
class className
{
public:
    T value;
    T function(T arg);
};
```

```
template <class T, class T2, int n>
class Example
{
    T arg1, arg2;
    T2 arg3;
}
```

Jak stworzyć klasę ze wzorca

```
className<dataType> classObject;
```

Na przykład:

```
className<int> classObject;
className<float> classObject;
className<string> classObject;
```

```
struct Figure{ };
className<Figure*> classObject;
```

Prosty przykład

```
template <class T>
class Example
{
private:
    T arg1, arg2;

public:
    Example(T _arg1, T _arg2)
    {
        arg1 = _arg1;
        arg2 = _arg2;
    }

    void displayResult()
    {
        cout << arg1 << ", " << arg2 << endl;
    }
};
```

Prosty przykład cd.

```
int main()
{
    Example<int> intExample(2, 1);
    Example<float> floatExample(2.4, 1.2);
    Example<string> stringExample("name1", "name2");
    cout << "Int:" << endl;
    intExample.displayResult();

    cout << endl << "Float:" << endl;
    floatExample.displayRes   (const char [7])"Float:"

    cout << endl << "String:" << endl;
    stringExample.displayResult();

    getchar();
    return 0;
};
```

```
Int:
2, 1

Float:
2.4, 1.2

String:
name1, name2
```

Wykrywanie błędów

- Wzorce sprawdzane są dopiero w momencie użycia w programie
- Wzorce z błędem mogą zostać poprawnie skompilowane!

Warto sprawdzić poprawność klasy na konkretnym przypadku, dopiero potem zamienić ją na ogólną klasę.

W przypadku modyfikacji wzorca, zostaną zmodyfikowane wszystkie klasy stworzone z niego.Należy zamiast tego stworzyć wzorzec klasy pochodnej.

Wzorzec szczegółowy

```
class Example<double>
{
private:
    double arg1, arg2;

public:
    Example(double _arg1, double _arg2)
    {
        arg1 = _arg1;
        arg2 = _arg2;
    }

    void displayResult()
    {
        cout << arg1 << ", " << arg2 << endl;
    }
};
```

Możliwe jest stworzenie innej wersji wzorca dla konkretnego parametru.