

FUNKCJE WYŻSZYCH RZĘDÓW

Filtrowanie listy

Problem: przefiltrować listę



```
class Program
{
    static void Main(string[] args)
    {
        List<string> przedmioty = new List<string>() {
            "Paradygmaty programowania",
            "Programowanie stron internetowych",
            "Algebra",
            "Logika dla informatyków",
            "Sieci komputerowe"
        };
    }
}
```

Filtrowanie listy

```
class OperacjeNaLiscie {
    public static List<string> TylkoDlugieNazwy(List<string> lista) {
        List<string> rezultat = new List<string>();
        for(int i=0; i<lista.Count; i++)
            if(lista[i].Length > 18)
                rezultat.Add(lista[i]);
        return rezultat;
    }

    public static List<string> TylkoNaLitereP(List<string> lista) {
        List<string> rezultat = new List<string>();
        for(int i=0; i<lista.Count; i++)
            if(lista[i][0] == 'P')
                rezultat.Add(lista[i]);
        return rezultat;
    }
}
```

Filtrowanie listy

```
class Program
{
    static void Main(string[] args)
    {
        List<string> przedmioty = new List<string>() {
            "Paradygmaty programowania",
            ...
        };

        List<string> dlugieNazwy
            = OperacjeNaLiscie.TylkoDlugieNazwy(przedmioty);
        List<string> przedmiotyNaLitereP
            = OperacjeNaLiscie.TylkoNaLitereP(przedmioty);
    }
}
```

Filtrowanie listy



Czy można to rozwiązać
bardziej elegancko?

Filtrowanie listy

```
class OperacjeNaLiscie {  
    public static List<string> TylkoDlugieNazwy(List<string> lista) {  
        List<string> rezultat = new List<string>();  
        for(int i=0; i<lista.Count; i++)  
            if(lista[i].Length > 18)  
                rezultat.Add(lista[i]);  
        return rezultat;  
    }  
  
    public static List<string> TylkoNaLitereP(List<string> lista) {  
        List<string> rezultat = new List<string>();  
        for(int i=0; i<lista.Count; i++)  
            if(lista[i][0] == 'P')  
                rezultat.Add(lista[i]);  
        return rezultat;  
    }  
}
```



Wzorzec Strategii

Czynnościowy wzorzec projektowy, który definiuje rodzinę wymiennych algorytmów i kapsułkuje je w postaci klas.

Wzorzec strategii

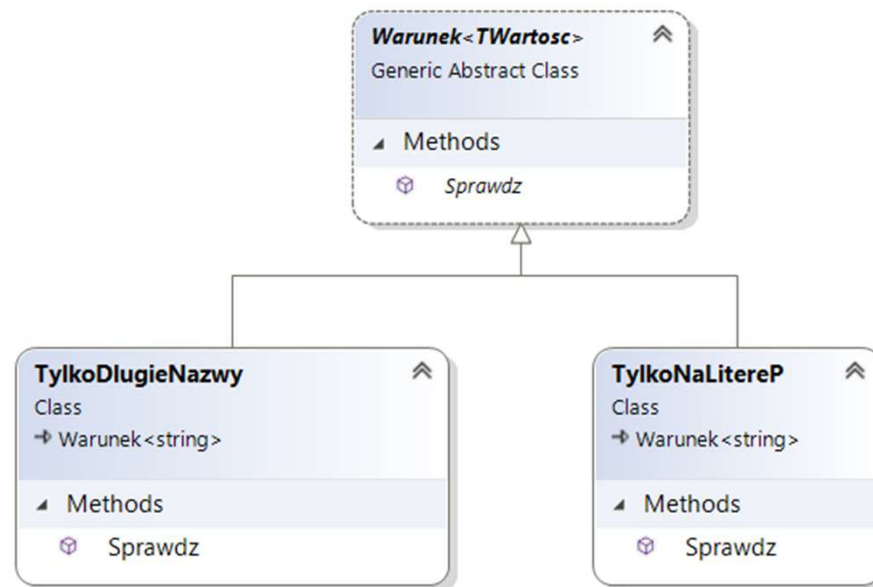
```
abstract class Warunek {  
    public abstract bool Sprawdz(string wartosc);  
}
```

```
abstract class Warunek <TWartosc> {  
    public abstract bool Sprawdz(TWartosc wartosc);  
}
```


Wzorzec strategii

```
class TylkoDlugieNazwy : Warunek<string> {  
    public override bool Sprawdz(string wartosc) {  
        return wartosc.Length > 18;  
    }  
}  
  
class TylkoNaLitereP : Warunek<string> {  
    public override bool Sprawdz(string wartosc) {  
        return wartosc[0] == 'P';  
    }  
}
```

Wzorzec strategii



Wzorzec strategii

```
class OperacjeNaLiscie {  
    public static List<string> Filtruj(  
        List<string> lista,  
        Warunek<string> warunek)  
    {  
        List<string> rezultat = new List<T>();  
        for(int i=0; i<lista.Count; i++)  
            if(warunek.Sprawdz(lista[i]))  
                rezultat.Add(lista[i]);  
        return rezultat;  
    }  
}
```

Wzorzec strategii

```
class OperacjeNaLiscie {  
    public static List<T> Filtruj<T>(  
        List<T> lista,  
        Warunek<T> warunek)  
    {  
        List<T> rezultat = new List<T>();  
        for(int i=0; i<lista.Count; i++)  
            if(warunek.Sprawdz(list[i]))  
                rezultat.Add(lista[i]);  
        return rezultat;  
    }  
}
```

Wzorzec strategii

```
class Program {
    static void Main(string[] args) {
        List<string> przedmioty = new List<string>() {
            "Paradygmaty programowania",
            ...
        };

        List<string> dlugieNazwy
            = OperacjeNaLiscie.Filtruj(przedmioty, new TylkoDlugieNazwy());
        List<string> przedmiotyNaLitereP
            = OperacjeNaLiscie.Filtruj(przedmioty, new TylkoNaLitereP());
    }
}
```

Funkcje wyższych rzędów

Funkcje wyższych rzędów są to funkcje, które przyjmują inne funkcje jako parametry, lub funkcje zwracają.

Typy funkcyjne w C#

Typy do opisu funkcji

```
Func<out TResult>  
Func<in T, out TResult>  
Func<in T1, in T2, out TResult>  
Func<in T1,...,in T16, out TResult>
```

Typy do opisu akcji

```
Action<in T>  
Action<in T1, in T2>  
Action<in T1,...,in T16>
```

Funkcje wyższych rzędów

```
class OperacjeNaLiscie {  
    public static List<T> Filtruj<T>(  
        List<T> lista,  
        Func<T, bool> warunek)  
    {  
        List<T> rezultat = new List<T>();  
        for(int i=0; i<lista.Count; i++)  
            if(warunek(lista[i]))  
                rezultat.Add(lista[i]);  
        return rezultat;  
    }  
}
```


Funkcje wyższych rzędów

```
class Program {  
  
    static bool TylkoDlugie(string wartosc) => wartosc.Length > 18;  
    static bool TylkoNaP(string wartosc) => wartosc[0] == 'P';  
    static void Main(string[] args) {  
  
        List<string> przedmioty = new List<string>() {  
            "Paradygmaty programowania",  
            ...  
        };  
  
        List<string> dlugieNazwy  
            = OperacjeNaLiscie.Filtruj(przedmioty, TylkoDlugie);  
        List<string> przedmiotyNaLitereP  
            = OperacjeNaLiscie.Filtruj(przedmioty, TylkoNaP);  
    }  
}
```

Funkcje wyższych rzędów

```
class Program {  
  
    static bool TylkoDlugie(string wartosc) => wartosc.Length > 18;  
  
    static void Main(string[] args) {  
  
        List<string> przedmioty = new List<string>() {  
            "Paradygmaty programowania",  
            ...  
        };  
  
        List<string> dlugieNazwy  
            = OperacjeNaLiscie.Filtruj(przedmioty, (nazwa)=>nazwa.Length>18);  
    }  
}
```

Wzorzec strategii

```
class TylkoNaLitere : Warunek<string> {  
  
    char _litera;  
    public TylkoNaLitere(char litera) {  
        _litera = litera;  
    }  
  
    public bool Sprawdz(string wartosc) {  
        return wartosc[0] == _litera;  
    }  
  
}
```

Funkcje wyższych rzędów

```
class Program {  
  
    static bool TylkoNaLitere(string nazwa, char litera) => nazwa[0] == litera;  
  
    static Func<string, bool> TylkoNaLitere(char litera) {  
        return (nazwa) => TylkoNaLitere(nazwa, litera);  
    }  
  
    static void Main(string[] args) {  
  
        List<string> przedmioty = new List<string>() {  
            "Paradygmaty programowania", ...  
        };  
  
        List<string> dlugieNazwy  
            = OperacjeNaLiscie.Filtruj(przedmioty, TylkoNaLitere('S'));  
    }  
}
```

Funkcje wyższych rzędów

```
class Program {  
  
    static Func<string, bool> TylkoNaLitere(char litera) {  
        return (nazwa) => nazwa[0] == litera;  
    }  
  
    static void Main(string[] args) {  
  
        List<string> przedmioty = new List<string>() {  
            "Paradygmaty programowania", ...  
        };  
  
        List<string> dlugieNazwy  
            = OperacjeNaLiscie.Filtruj(przedmioty, TylkoNaLitere('S'));  
    }  
}
```

Mierzenie czasu wykonania funkcji

```
static double ZmierzCzas(Action akcja) {  
    var stoper = new Stopwatch();  
    stoper.Start();  
    akcja();  
    stoper.Stop();  
    return stoper.Elapsed.TotalMilliseconds;  
}
```