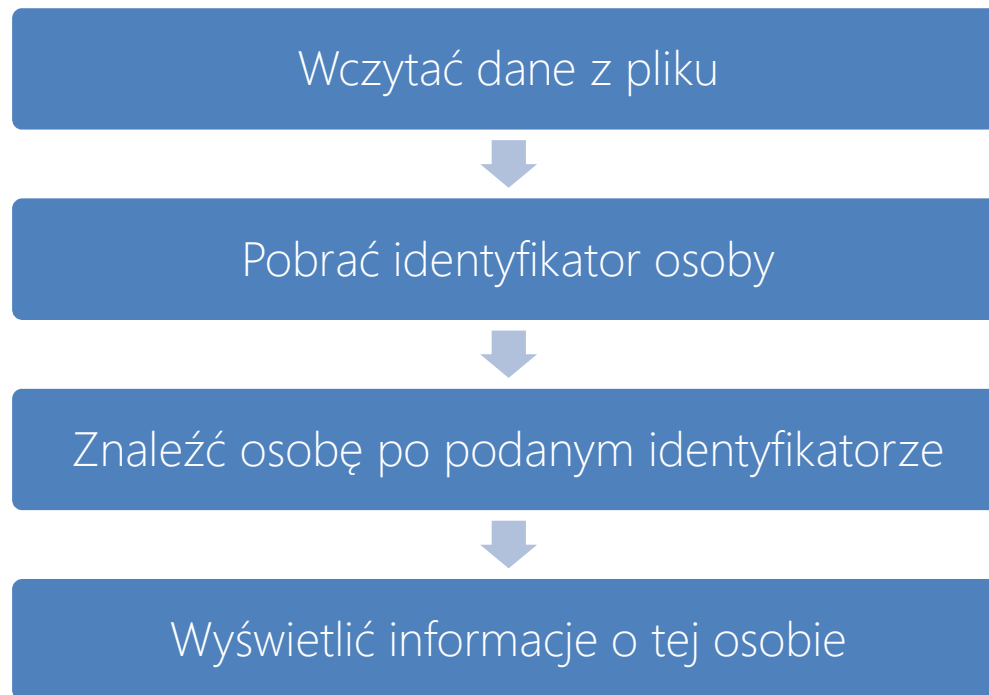


# PODSTAWOWE WZORCE PROGRAMOWANIA FUNKCYJNEGO W C#

# Przykładowa aplikacja

---



# Przykładowa aplikacja

---

```
static void Main(string[] args)
{
    var dane = WczytajDane("dane.json");
    Console.WriteLine("Podaj ID osoby do wyświetlenia");
    string str = Console.ReadLine();
    int id = int.Parse(str);
    Osoba osoba = ZnajdzOsobePoId(dane, id);
    WyświetlDane(osoba);
    Console.ReadKey();
}
```

## Przykładowa aplikacja

---

```
private static List<Osoba> WczytajDane(string nazwaPliku)
{
    var linie = System.IO.File.ReadAllText(nazwaPliku);
    return JsonConvert.DeserializeObject<List<Osoba>>(linie);
}

private static void WyswietlDane(Osoba osoba)
{
    Console.WriteLine($"Id: {osoba.Id}");
    Console.WriteLine($"Imie: {osoba.Imie}");
    Console.WriteLine($"Nazwisko: {osoba.Nazwisko}");
}
```

## Przykładowa aplikacja

---

```
private static Osoba ZnajdzOsobePoId(List<Osoba> dane, int id)
{
    foreach(var osoba in dane)
    {
        if (osoba.Id == id)
            return osoba;
    }
    return null;
}
```

# Co w tej aplikacji może pójść źle?

---



```
static void Main(string[] args)
{
    var dane = WczytajDane("dane.json");
    Console.WriteLine("Podaj ID osoby do wyświetlenia");
    string str = Console.ReadLine();
    int id = int.Parse(str);
    Osoba osoba = ZnajdzOsobePoId(dane, id);
    WyświetlDane(osoba);
    Console.ReadKey();
}
```

# Co w tej aplikacji może pójść źle?

---

```
static void Main(string[] args)
{
    var dane = WczytajDane("dane.json");
    Console.WriteLine("Podaj ID osoby do wyświetlenia");
    string str = Console.ReadLine();
    int id = int.Parse(str);
    Osoba osoba = GetOsoba(id);
    WyswietlDane(osoba);
    Console.ReadLine();
}
```



Może rzucić 9 różnych typów wyjątków

Może rzucić wyjątek `JsonReaderException`

```
private static List<Osoba> WczytajDane(string nazwaPliku)
{
    var linie = System.IO.File.ReadAllText(nazwaPliku);
    return JsonConvert.DeserializeObject<List<Osoba>>(linie);
}
```

## Co w tej aplikacji może pójść źle?

---

```
static void Main(string[] args)
{
    var dane = WczytajDane("dane.json");
    Console.WriteLine("Podaj ID osoby do wyświetlenia");
    string str = Console.ReadLine();
    int id = int.Parse(str);
    Osoba osoba = ZnajdzOsobePoId(dane, id);
    WyswietlDane(osoba);
    Console.ReadKey();
}
```

Mogą być rzucone wyjątki, ale są mało prawdopodobne



## Co w tej aplikacji może pójść źle?

---

```
static void Main(string[] args)
{
    var dane = WczytajDane("dane.json");
    Console.WriteLine("Podaj ID osoby do wyświetlenia");
    string str = Console.ReadLine();
    int id = int.Parse(str);
    Osoba osoba = ZnajdzOsobePoId(dane, id);
    WyswietlDane(osoba);
    Console.ReadKey();
}
```

String może być w złym formacie

# Co w tej aplikacji może pójść źle?

---

```
static void Main(string[] args)
{
    var dane = WczytajDane("dane.json");
    Console.WriteLine("Podaj ID osoby do wyświetlenia");
    string str = Console.ReadLine();
    int id = int.Parse(str);
    Osoba osoba = ZnajdzOsobePoId(dane, id);
    WyświetlDane(osoba);
    Console.ReadKey();
}
```

Ten fragment jest  
bezpieczny

# Co w tej aplikacji może pójść źle?

---

```
static void Main(string[] args)
{
    var dane = WczytajDane("dane.json");
    Console.WriteLine("Podaj ID osoby do wyświetlenia");
    string str = Console.ReadLine();
    int id = int.Parse(str);
    Osoba osoba = ZnajdzOsobePoId(dane, id);
    WyswietlDane(osoba);
    Console.ReadKey();
}

private static void WyswietlDane(Osoba osoba)
{
    Console.WriteLine($"Id: {osoba.Id}");
    Console.WriteLine($"Imie: {osoba.Imie}");
    Console.WriteLine($"Nazwisko: {osoba.Nazwisko}");
}
```



Jeżeli osoba będzie null to będzie wyjątek

# Naprawa aplikacji

---

```
static void Main(string[] args)
{
    try {
        var dane = WczytajDane("dane.json");
        int id = 0;
        bool ok;
        do {
            try {
                Console.WriteLine("Podaj ID osoby do wyświetlenia");
                string str = Console.ReadLine();
                id = int.Parse(str);
                ok = true;
            }
        }
    }
}
```

# Naprawa aplikacji

---

```
        catch (Exception e) {
            Console.Clear();
            PokazBlad("To musi być liczba");
            ok = false;
        }
    }
    while (!ok);
    Osoba osoba = ZnajdzOsobePoId(dane, id);
    if (osoba != null)
        WyswietlDane(osoba);
    else
        PokazBlad("Nie ma osoby o takim ID");
}
catch(Exception exception)
{
    PokazBlad("Nie mogę nic z tym błędem zrobić! Kończę pracę!");
}
}
```

# Naprawa aplikacji

---

```
private static void PokazBlad(string wiadomosc)
{
    Console.ForegroundColor = ConsoleColor.Red;
    Console.WriteLine(wiadomosc);
    Console.ResetColor();
}
```

```
static void Main(string[] args) {
    try {
        var dane = WczytajDane("dane.json");
        int id = 0;
        bool ok;
        do {
            try {
                Console.WriteLine("Podaj ID osoby do wyświetlenia");
                string str = Console.ReadLine();
                id = int.Parse(str);
                ok = true;
            }
            catch (Exception e) {
                Console.Clear();
                PokazBlad("To musi być liczba");
                ok = false;
            }
        }
        while (!ok);
        Osoba osoba = ZnajdzOsobePoId(dane, id);
        if (osoba != null)
            WswietlDane(osoba);
        else
            PokazBlad("Nie ma osoby o takim ID");
    }
    catch (Exception exception) {
        PokazBlad("Nie mogę nic z tym błędem zrobić! Kończę pracę!");
    }
}
```

```

static void Main(string[] args) {
    try {
        var dane = WczytajDane("dane.json");
        int id = 0;
        bool ok;
        do {
            try {
                Console.WriteLine("Podaj ID osoby do wyświetlenia");
                string str = Console.ReadLine();
                id = int.Parse(str);
                ok = true;
            }
            catch (Exception e) {
                Console.Clear();
                PokazBlad("To musi być liczba");
                ok = false;
            }
        }
        while (!ok);
        Osoba osoba = ZnajdzOsobePoId(dane, id);
        if (osoba != null)
            WyswietlDane(osoba);
        else
            PokazBlad("Nie ma osoby o takim ID");
    }
    catch (Exception exception) {
        PokazBlad("Nie mogę nic z tym błędem zrobić! Kończę pracę!");
    }
}

```



Czy można to napisać lepiej?

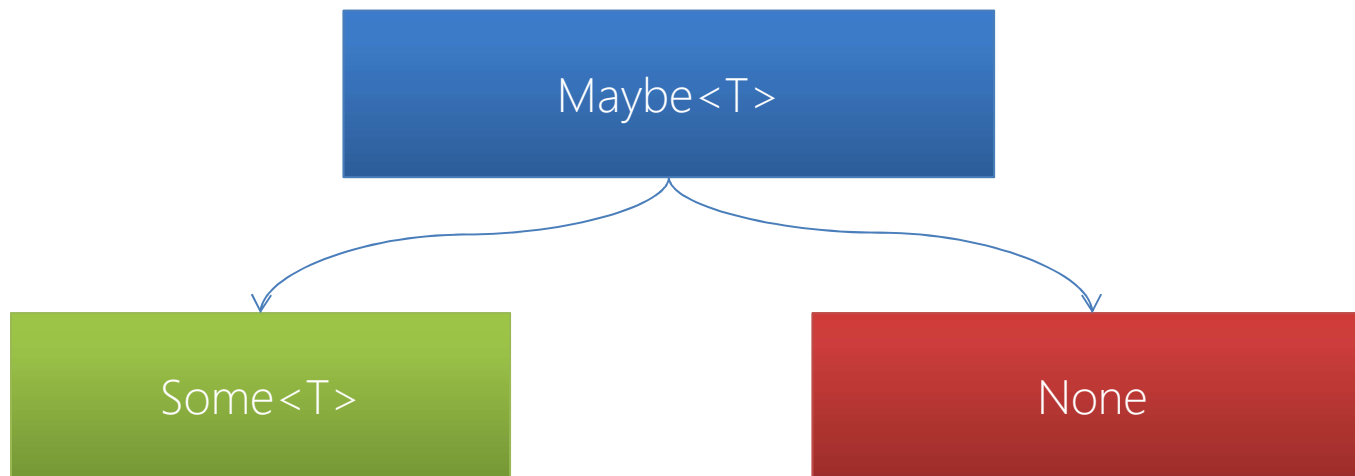


# OBSŁUGA WARTOŚCI NULL

# Typ Maybe/Option

---

Typ Maybe pozwala na określanie czy dana wartość istnieje czy nie



# Typ Maybe

---

```
namespace Maybe {  
  
    public class None {  
        private None() { }  
        public static readonly None Value = new None();  
    }  
  
    public class Some<T> {  
        public readonly T Value;  
  
        public Some(T value) {  
            if (value == null) throw new ArgumentNullException();  
            Value = value;  
        }  
    }  
}
```

# Typ Maybe

---

```
public static class Functional
{
    public static Maybe.None None => Maybe.None.Value;
    public static Maybe.Some<T> Some<T>(T value)
        => new Maybe.Some<T>(value);
}
```

# Typ Maybe

---

```
namespace ConsoleApp5
{
    using static ConsoleApp5.Functional;

    class Program {

        static void Main(string[] args)
        {
            var wiek = Some(12);
            var imie = None;
        }

    }
}
```

# Typ Maybe

---

```
public class Maybe<T> {  
    readonly bool _hasValue;  
    readonly T _value;  
  
    private Maybe(T value) {  
        _hasValue = true;  
        _value = value;  
    }  
  
    private Maybe() { _hasValue = false; }
```

# Typ Maybe

---

```
public static implicit operator Maybe<T>(Maybe.None v)
    => new Maybe<T>();
```

```
public static implicit operator Maybe<T>(Maybe.Some<T> v)
    => new Maybe<T>(v.Value);
```

```
public static implicit operator Maybe<T>(T v)
    => v == null ? (Maybe<T>)None : Some(v);
```

# Typ Maybe

---

```
public TOutput When<TOutput>(
    Func<TOutput> whenNone,
    Func<T, TOutput> whenSome)
    => _hasValue ? whenSome(_value) : whenNone();

public T Reduce(Func<T> whenNone)
    => _hasValue ? _value : whenNone();
}
```



```
static void Main(string[] args) {
    try {
        var dane = WczytajDane("dane.json");
        int id = 0;
        bool ok;
        do {
            try {
                Console.WriteLine("Podaj ID osoby do wyświetlenia");
                string str = Console.ReadLine();
                id = int.Parse(str);
                ok = true;
            }
            catch (Exception e) {
                Console.Clear();
                PokazBlad("To musi być liczba");
                ok = false;
            }
        }
        while (!ok);
        Osoba osoba = ZnajdzOsobePoId(dane, id);
        if (osoba != null)
            WswietlDane(osoba);
        else
            PokazBlad("Nie ma osoby o takim ID");
    }
    catch (Exception exception) {
        PokazBlad("Nie mogę nic z tym błędem zrobić! Kończę pracę!");
    }
}
```

```
private static Maybe<Osoba> ZnajdzOsobePoId(
    List<Osoba> dane, int id)
{
    foreach (var osoba in dane) {
        if (osoba.Id == id)
            return osoba;
    }
    return None;
}
```

```
Osoba osoba = ZnajdzOsobePoId(dane, id);  
    if (osoba != null)  
        WswietlDane(osoba);  
    else  
        PokazBlad("Nie ma osoby o takim ID");
```



```
var osoba = ZnajdzOsobePoId(dane, id );  
osoba.When(  
    some: ...,  
    none: ...  
);
```

```
internal class Osoba
{
    public int Id { get; set; }
    public string Imie { get; set; }
    public string Nazwisko { get; set; }

    public override string ToString() {
        return new StringBuilder()
            .AppendLine($"Id: {Id}")
            .AppendLine($"Imie: {Imie}")
            .AppendLine($"Nazwisko: {Nazwisko}").ToString();
    }
}
```

```
Osoba osoba = ZnajdzOsobePoId(dane, id);  
    if (osoba != null)  
        WswietlDane(osoba);  
    else  
        PokazBlad("Nie ma osoby o takim ID");
```



```
var osoba = ZnajdzOsobePoId(dane, id);  
Console.WriteLine(  
    osoba.When(  
        some: (o) => o.ToString(),  
        none: () => "Nie ma osoby o takim ID"  
    )  
);
```

```
static void Main(string[] args) {
    try {
        var dane = WczytajDane("dane.json");
        int id = 0;
        bool ok;
        do {
            try {
                Console.WriteLine("Podaj ID osoby do wyświetlenia");
                string str = Console.ReadLine();
                id = int.Parse(str);
                ok = true;
            }
            catch (Exception e) {
                Console.Clear();
                PokazBlad("To musi być liczba");
                ok = false;
            }
        }
        while (!ok);
        var osoba = ZnajdzOsobePoId(dane, id);
        Console.WriteLine(
            osoba.When( some: (o) => o.ToString(), none: () => "Nie ma osoby o takim ID" )
        );
    }
    catch(Exception exception) {
        PokazBlad("Nie mogę nic z tym błędem zrobić! Kończę pracę!");
    }
}
```

```
public static class Extensions {  
    public static Maybe<int> ParseToInt(this string str) {  
        try {  
            return int.Parse(str);  
        }  
        catch(Exception) {  
            return None;  
        }  
    }  
}
```

```
Maybe<int> id;
do {
    Console.WriteLine("Podaj ID osoby do wyświetlenia");
    string str = Console.ReadLine();
    id = str.ParseToInt();
}

while (id == (Maybe<int>)None);

var osoba = ZnajdzOsobePoId(dane, id.Reduce(() => 0));
```



```
static void Main(string[] args) {  
    try {  
        var dane = WczytajDane("dane.json");  
        Maybe<int> id;  
  
        do {  
            Console.WriteLine("Podaj ID osoby do wyświetlenia");  
            string str = Console.ReadLine();  
            id = str.ParseToInt();  
        }  
        while (id == (Maybe<int>)None);  
  
        var osoba = ZnajdzOsobePoId(dane, id.Reduce(() => 0) );  
  
        Console.WriteLine(  
            osoba.When( some: (o) => o.ToString(), none: () => "Nie ma osoby o takim ID" )  
        );  
    }  
    catch(Exception exception) {  
        PokazBlad("Nie mogę nic z tym błędem zrobić! Kończę pracę!");  
    }  
}
```

```
private static Maybe<List<Osoba>> WczytajDane(string nazwaPliku)
{
    try {
        var linie = System.IO.File.ReadAllText(nazwaPliku);
        return JsonConvert.DeserializeObject<List<Osoba>>(linie);
    }
    catch(Exception e) {
        return None;
    }
}
```

```
static void Main(string[] args) {
    var wynik = WczytajDane("dane.json")
        .When( some: (dane) =>
            {
                Maybe<int> id;
                do {
                    Console.WriteLine("Podaj ID osoby do wyświetlenia");
                    string str = Console.ReadLine();
                    id = str.ParseToInt();
                }
                while (id == (Maybe<int>)None);
                var osoba = ZnajdzOsobePoId(dane, id.Reduce(() => 0));
                return osoba.When(
                    some: (o) => o.ToString(),
                    none: () => "Nie ma osoby o takim ID"
                );
            },
            none: () => "Nie mogę nic z tym błędem zrobić! Kończę pracę!");
    Console.WriteLine(wynik);
}
```

# MAPOWANIE, FUNKTORY I MONADY

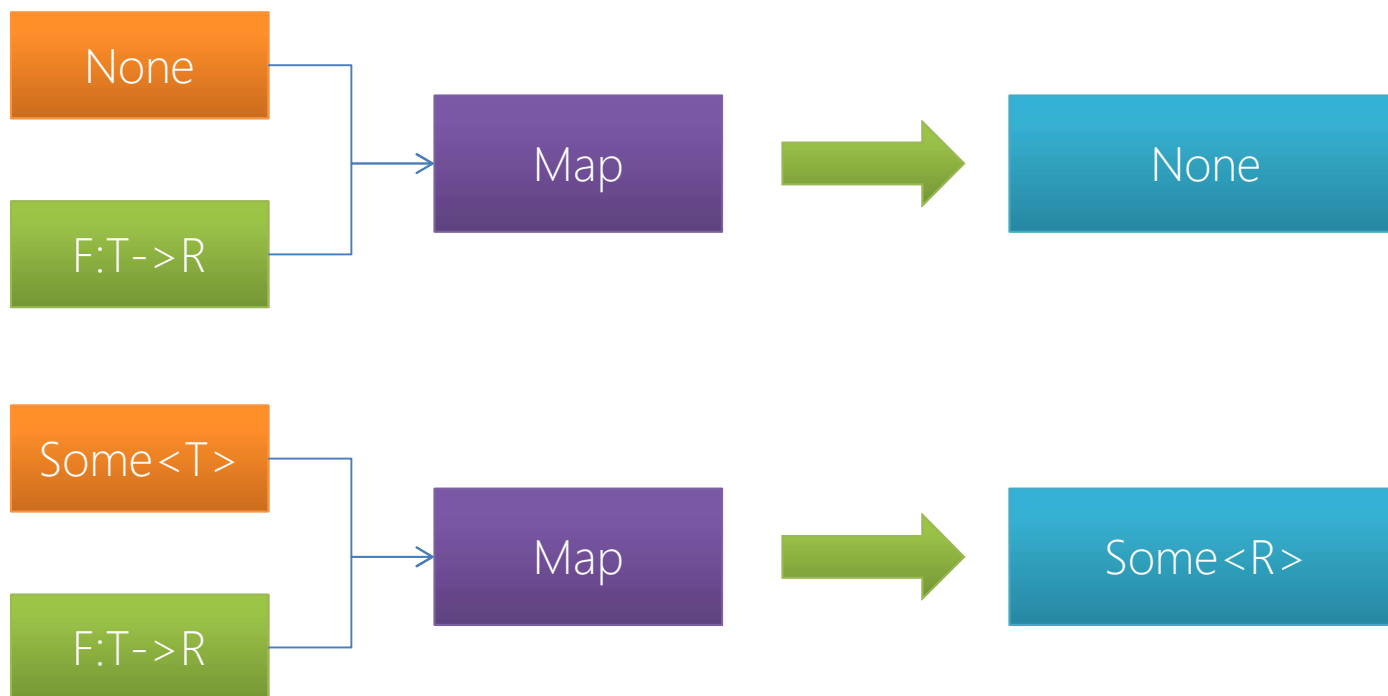
## Mapowanie

---

$$\text{Map: } (M < T >, (T \rightarrow R)) \rightarrow M < R >$$
$$\text{Map: } (\text{Maybe } < T >, (T \rightarrow R)) \rightarrow \text{Maybe } < R >$$

# Mapowanie

---



# Mapowanie

---

```
public static class Functional
{
    public static Maybe<R> Map<T, R>(
        this Maybe<T> maybe, Func<T, R> map)
        => maybe.When(
            () => (Maybe<R>)None,
            (v) => Some(map(v))
        );
}
```

```
static void Main(string[] args) {  
    var wynik =  
        WczytajDane("dane.json")  
        .Map((dane) => {  
            Maybe<int> id;  
            do  
                id = WczytajId();  
            while (id == (Maybe<int>)None);  
            var osoba = ZnajdzOsobePoId(dane, id.Reduce(() => 0));  
            return osoba.When(  
                none: ()=>"Nie ma osoby o takim ID",  
                some: o => o.ToString());  
        });  
    Console.WriteLine(wynik.Reduce(()=>""));  
    Console.ReadKey();  
}
```



# Funktory

---

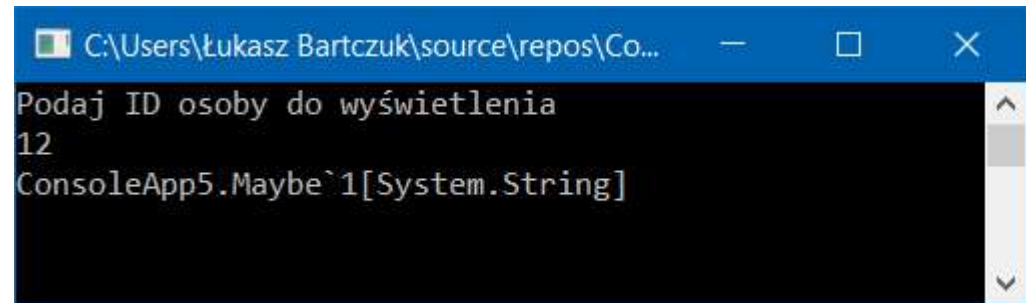
$$\text{Map}: (M < T >, (T \rightarrow R)) \rightarrow M < R >$$

W programowaniu funkcyjnym dowolny typ wyposażony w funkcję Map nazywamy funktorem

# Mapowania

---

```
var wynik =  
    WczytajDane("dane.json")  
        .Map((dane) => {  
            Maybe<int> id;  
            do  
                id = WczytajId();  
            while (id == (Maybe<int>)None);  
            var osoba = ZnajdzOsobePoId(dane, id.Reduce(() => 0));  
            return osoba.Map(o => o.ToString());  
        });  
Console.WriteLine(wynik.Reduce(() => ""));  
Console.ReadKey();
```



# Mapowania

---

```
var wynik =  
    WczytajDane("dane.json")  
    .Map((dane) => {  
        Maybe<int> id;  
        do  
            id = WczytajId();  
            while (id == (Maybe<int>)None);  
            return ZnajdzOsobePoId(dane, id.Reduce(() => 0));  
    })  
    .Map(o=>o.ToString());  
Console.WriteLine(wynik.Reduce(()=>""));  
Console.ReadKey();
```

## Bindowanie

---

Bind:  $(M < T >, (T \rightarrow M < R >)) \rightarrow M < R >$

Bind:  $(\text{Maybe} < T >, (T \rightarrow \text{Maybe} < R >)) \rightarrow \text{Maybe} < R >$

# Monady

---

$\text{Return}: T \rightarrow M < T >$

$\text{Return}: (T) \rightarrow \text{Maybe} < T >$

W programowaniu funkcyjnym dowolny typ wyposażony w funkcje Bind i Return nazywamy monadą

# Bindowanie

---

```
public static class Functional
{
    public static Maybe<R> Map<T, R>(
        this Maybe<T> maybe,
        Func<T, R> map)
        => maybe.When(() => (Maybe<R>)None,
            (v) => Some(map(v)));

    public static Maybe<R> Bind<T, R>(
        this Maybe<T> maybe,
        Func<T, Maybe<R>> map)
        => maybe.When(() => (Maybe<R>)None,
            (v) => map(v));
}
```

# Bindowanie

---

```
public static class Functional
{
    public static Maybe<R> Map<T, R>(
        this Maybe<T> maybe,
        Func<T, R> map)
        => maybe.When(() => (Maybe<R>)None,
            (v) => Some(map(v)));

    public static Maybe<R> Bind<T, R>(
        this Maybe<T> maybe,
        Func<T, Maybe<R>> map)
        => maybe.When(() => (Maybe<R>)None,
            (v) => map(v));
}
```

# Bindowanie

---

```
var wynik =  
    WczytajDane("dane.json")  
        .Bind((dane) => {  
            Maybe<int> id;  
            do  
                id = WczytajId();  
            while (id == (Maybe<int>)None);  
            return ZnajdzOsobePoId(dane, id.Reduce(() => 0));  
        })  
        .Map(o=>o.ToString());  
Console.WriteLine(wynik.Reduce(()=>""));  
Console.ReadKey();
```



# Bindowanie

---

```
var wynik =  
    WczytajDane("dane.json")  
        .Bind((dane) => {  
            Maybe<int> id;  
            do  
                id = WczytajId();  
            while (id == (Maybe<int>)None);  
            return id.Map((i) => (dane:dane, id:i));  
        })  
        .Bind((t)=>ZnajdzOsobePoId(t.dane,t.id ))  
        .Map(o=>o.ToString());  
Console.WriteLine(wynik.Reduce(()=>""));  
Console.ReadKey();
```

# Bindowanie

---

```
private static Maybe<(List<Osoba> dane, int id)>
    WczytajID(List<Osoba> dane)
{
    Maybe<int> id;
    do
        id = WczytajId();
    while (id == (Maybe<int>)None);
    return id.Map((i) => (dane: dane, id: i));
}
```

# Bindowanie

---

```
static void Main(string[] args)
{
    var wynik =
        WczytajDane("dane.json")
        .Bind((dane)=>WczytajID(dane))
        .Bind((t) => ZnajdzOsobePoId(t.dane, t.id))
        .Map(o => o.ToString());
    Console.WriteLine(wynik.Reduce(() => ""));
    Console.ReadKey();
}
```

MONADA EITHER

# Either

---

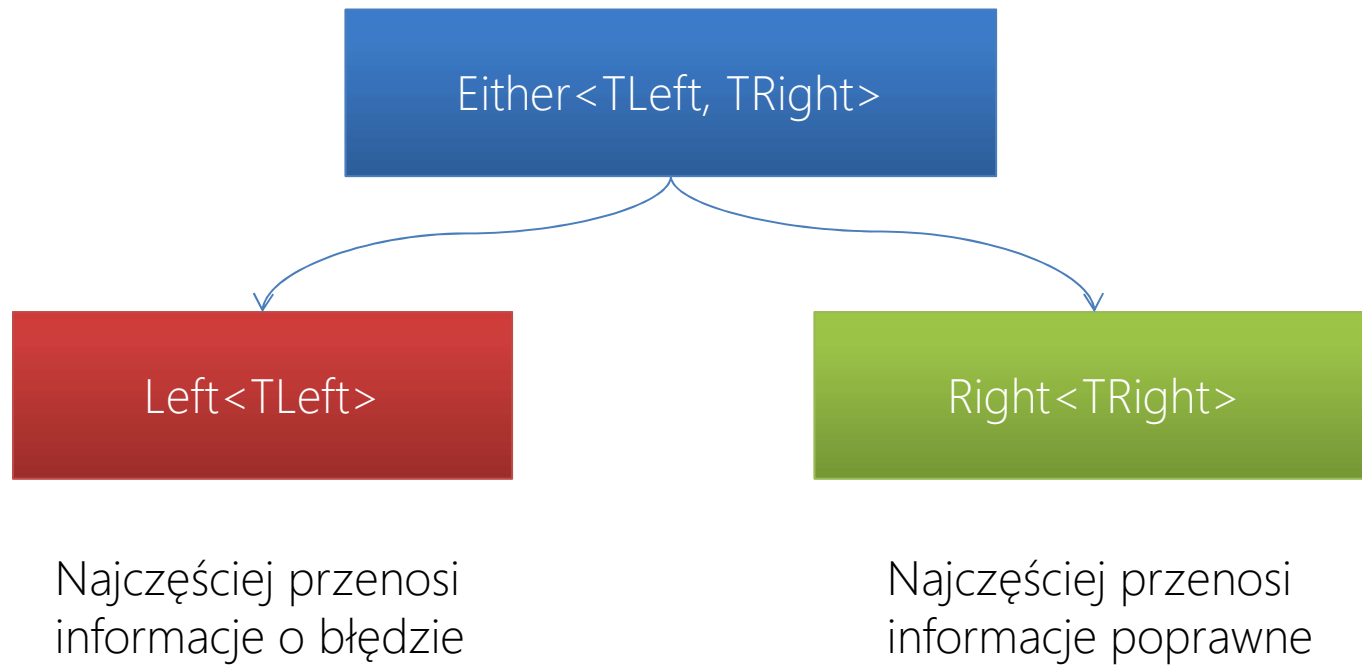
Monada Either jest funkcyjną metodą pozwalającą na rozwiązanie problemu obsługi błędów

Realizacja Railway Oriented Programming



# Monada Either

---



# Monada Either

---

```
namespace Either {
```

```
    public class Left<TLeft>{  
        public readonly TLeft Value;
```

```
        public Left(TLeft value) {  
            if (value == null)  
                throw new ArgumentNullException();  
            Value = value;  
        }  
    }
```

```
    public class Right<TRight> { ... }  
}
```



Implementacja taka  
jak Left

# Monada Either

---

```
public class Either<TLeft, TRight> {  
    TLeft _left;  
    TRight _right;  
    public readonly bool IsRight;  
    public readonly bool IsLeft;  
  
    private Either(TLeft value) {  
        _left = value;  
        IsLeft = true; IsRight = false;  
    }  
  
    private Either(TRight value)  
        _right = value;  
        IsLeft = false; IsRight = true;  
    }
```



# Monada Either

---

```
public TResult Match<TResult>(
    Func<TLeft, TResult> leftFunction,
    Func<TRight, TResult> rightFunction)
    => IsLeft ? leftFunction(this._left) :
        rightFunction(this._right);
```

```
public void Do(Action<TLeft> leftAction,
               Action<TRight> rightAction) {
    if (IsLeft)
        leftAction(this._left);
    else
        rightAction(this._right);
}
```

```
public void WhenLeft(Action<TLeft> a) { if(IsLeft) a(_left); }
```

# Monada Either

---

```
public static implicit operator Either<TL, TR>(Either.Left<TL> v)  
    => new Either<TL, TR>(v.Value);
```

```
public static implicit operator Either<TL, TR>(Either.Right<TR> v)  
    => new Either<TL, TR>(v.Value);
```

```
public static implicit operator Either<TL, TR>(TL v)  
    => new Either<TL, TR>(v);
```

```
public static implicit operator Either<TL, TR>(TR v)  
    => new Either<TL, TR>(v);
```

```
}
```

# Monada Either

---

```
public static class Functional {  
  
    public static Either<L> Left<L>(L v)  
        => new Either.Left<L>(v);  
    public static Either<R> Right<R>(R v)  
        => new Either.Right<R>(v);  
  
    public static Either<TNL, TNR> Map<TL, TR, TNL, TNR>  
        (this Either<TL, TR> either,  
         Func<TL, TNL> mapLeft,  
         Func<TR, TNR> mapRight)  
  
        => either.Match<Either<TNL, TNR>>  
            (l => Left(mapLeft(l)), r => Right(mapRight(r)));  
}
```

# Monada Either

---

```
public static Either<TL, TNR>  
  Map<TL, TR, TNR>  
    (this Either<TL, TR> either,  
      Func<TR, TNR> map)  
    => either.Match<Either<TL, TNR>>(l => Left(l), r => Right(map(r)));
```

```
public static Either<TL, TNR>  
  Bind<TL, TR, TNR>  
    (this Either<TL, TR> either,  
      Func<TR, Either<TL, TNR>> map)  
    => either.Match(l => Left(l), r => map(r));  
}
```

## Przykładowy program

---

```
static void Main(string[] args)
{
    var wynik =
        WczytajDane("dane.json")
        .Bind((dane)=>WczytajID(dane))
        .Bind((t) => ZnajdzOsobePoId(t.dane, t.id))
        .Map(o => o.ToString());
    Console.WriteLine(wynik.Reduce(() => ""));
    Console.ReadKey();
}
```

# Monada Either

---

```
private static Either<string, List<Osoba>>
    WczytajDane(string nazwaPliku)
{
    try {
        var linie = File.ReadAllText(nazwaPliku);
        return JsonConvert.DeserializeObject<List<Osoba>>(linie);
    }
    catch (Exception e) {
        return e.Message;
    }
}
```

# Monada Either

---

```
private static Either<string, (List<Osoba> dane, int id)>
    WczytajID(List<Osoba> dane) {
    Either<string, int> id;
    do {
        id = WczytajId();
        id.WhenLeft(PokazBlad);
    }
    while ( id.IsLeft );
    return id.Map((s)=>s, (i) => (dane, i));
}

private static Either<string, int> WczytajId() {
    Console.WriteLine("Podaj ID osoby do wyświetlenia");
    string str = Console.ReadLine();
    return str.ParseToInt();
}
```

# Monada Either

---

```
public static class Extensions {  
    public static Either<string,int> ParseToInt(this string str)  
    {  
        try {  
            return int.Parse(str);  
        }  
        catch (Exception e) {  
            return e.Message;  
        }  
    }  
}
```



# Monada Either

---

```
class Program {  
  
    public static Either<string, Maybe<Osoba>>  
        ZnajdzOsobe((List<Osoba> dane, int id) t)  
        => ZnajdzOsobePoId(t.dane, t.id);  
  
    public static Either<string, string> Redukuj(Maybe<Osoba> o)  
        => Right(o.Map(os => os.ToString())  
            .Reduce(() => "Brak osoby o takim id"));
```

# Monada Either

---

```
static void Main(string[] args) {  
    WczytajDane("dane.json")  
    .Bind((dane) => WczytajID(dane))  
    .Bind(ZnajdzOsobe)  
    .Bind(Redukuj)  
    .Do(  
        right: Console.WriteLine,  
        left: PokazBlad  
    );  
    Console.ReadKey();  
}
```

## SelectMany (11)

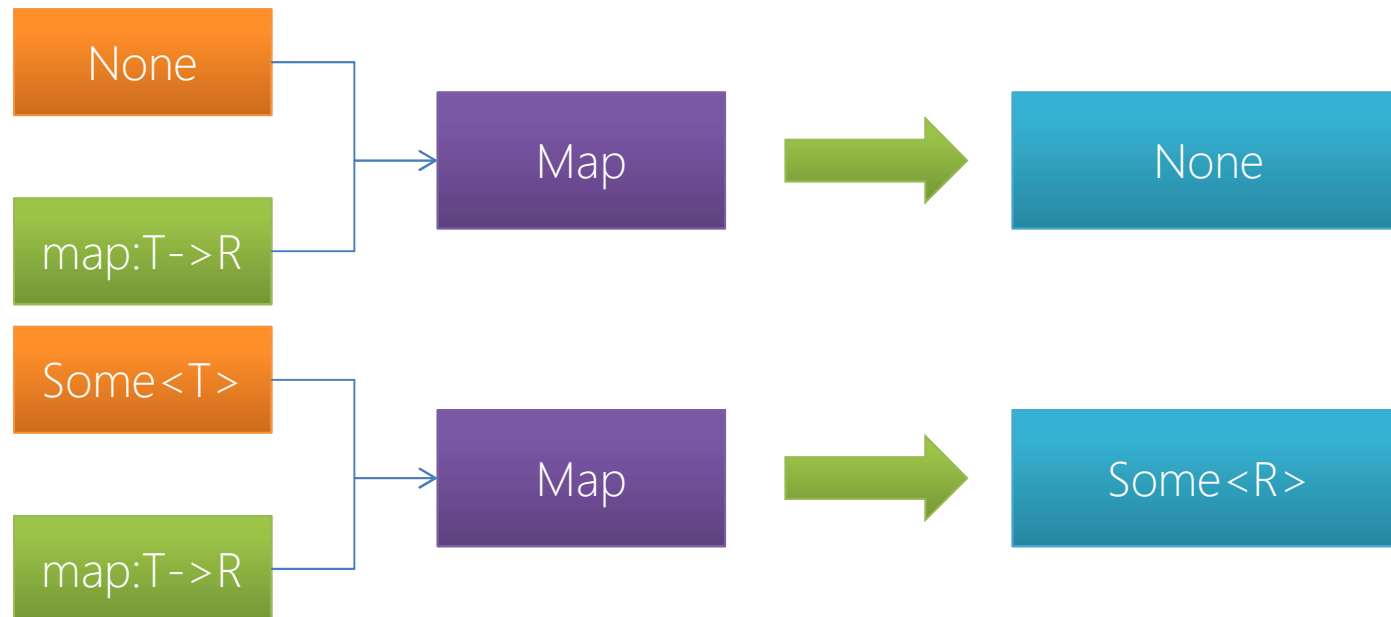
---

```
public static class Functional {  
  
    public static Maybe<R> Map<T, R>(  
        this Maybe<T> maybe, Func<T, R> map)  
        => maybe.When(() => (Maybe<R>)None, (v) => Some(map(v)));  
  
    public static Maybe<R> Bind<T, R>(  
        this Maybe<T> maybe, Func<T, Maybe<R>> map)  
        => maybe.When(() => (Maybe<R>)None, (v) => map(v));  
  
}
```

# Map

---

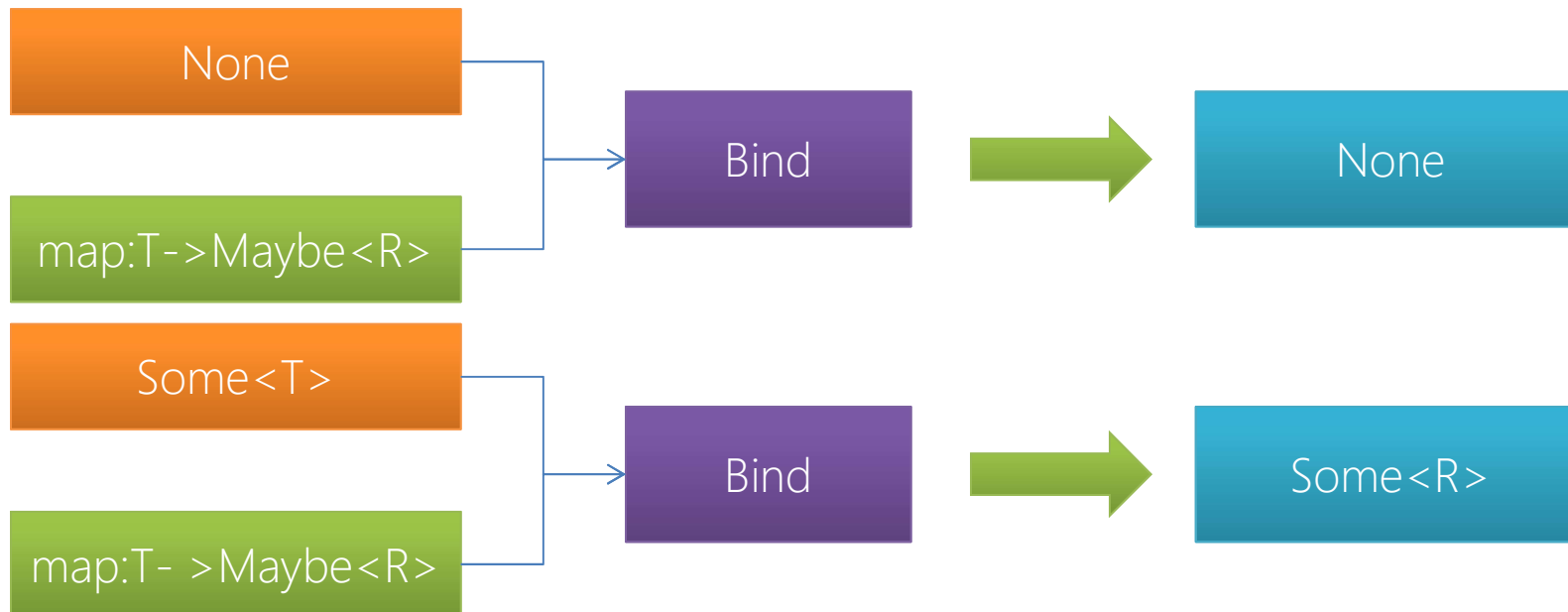
```
public static Maybe<R> Map<T, R>(
    this Maybe<T> maybe, Func<T, R> map)
    => maybe.When(() => (Maybe<R>)None, (v) => Some(map(v)));
```



# Bind

---

```
public static Maybe<R> Bind<T, R>(
    this Maybe<T> maybe, Func<T, Maybe<R>> map)
    => maybe.When(() => (Maybe<R>)None, (v) => map(v));
```



## SelectMany (13)

---

```
var wynik =  
    WczytajDane("dane.json")  
    .Bind((dane) => WczytajID(dane))  
    .Bind((t) => ZnajdzOsobePoId(t.dane, t.id))  
    .Select(o => o.ToString());  
Console.WriteLine(wynik.Reduce(() => ""));
```

## SelectMany (14)

---

```
public static class Functional {  
  
    public static Maybe<S> SelectMany<T, R, S>(  
        this Maybe<T> maybe,  
        Func<T, Maybe<R>> map,  
        Func<T, R, S> resultSelector)  
        => maybe.Bind(  
            outer => map(outer).Bind(  
                inner => (Maybe<S>)Some(  
                    resultSelector(outer, inner)))));  
  
}
```

## SelectMany (15)

---

```
var osoba = WczytajDane("dane.json")
    .SelectMany(dane => WczytajID(),
        (dane, id) => (dane: dane, id: id))
    .SelectMany(d => ZnajdzOsobePoId(d.dane, d.id),
        (d, o) => o.ToString());

Console.WriteLine(osoba.Reduce(() => ""));
```



## SelectMany (16)

---

```
var osoba =  
    from dane in WczytajDane("dane.json")  
    from id in WczytajID()  
    from osoba in ZnajdzOsobePoId(dane, id)  
    select osoba.ToString();  
  
Console.WriteLine(osoba.Reduce(() => ""));
```