LABORATORIUM 7

TREŚCI KSZTAŁCENIA:

- F# - PROGRAMOWANIE Z WYKORZYSTANIEM OBIEKTÓW

Spis treści

F# - programowanie obiektowe

Programowanie obiektowe w F# pozwala na tworzenie klas, interfejsów, dziedziczenia oraz enkapsulacji. Oto kilka zadań i przykładów ilustrujących, jak można wykorzystać obiektowość w F#:

Przykład 1: Klasa z właściwościami i metodami

Stwórz klasę Person, która będzie miała właściwości Name, Age i metodę Introduce, która wypisuje powitanie.

```
type Person(name: string, age: int) =
    // Pole
    let mutable _name = name
    let mutable _age = age

    // Właściwości
    member this.Name
        with get() = _name
        and set(value) = _name <- value

    member this.Age
        with get() = _age
        and set(value) = _age <- value

    // Metoda
    member this.Introduce() =
        printfn "Cześć, jestem %s i mam %d lat." _name _age



[<EntryPoint>]
let main argv =
    let person = Person("Jan", 30)
    person.Introduce()

    0
```

W F#, słowa kluczowe with i and w kontekście definiowania właściwości w klasach mają szczególne znaczenie i pełnią konkretne funkcje w strukturze deklaracji właściwości.

1. Słowo kluczowe with jest używane do definiowania getter (metody do odczytu wartości właściwości) oraz setter (metody do zapisu wartości właściwości). Pozwala to na przypisanie logiki do akcji odczytu i zapisu dla właściwości klasy.

```
member this.Age
    with get() = _age
    and set(value) = _age <- value
```

- with get() = _name oznacza, że właściwość Name jest powiązana z polem _name w klasie. Gdy ktoś odczytuje wartość właściwości Name, zostanie zwrócona wartość pola _name.
- and set(value) = _name <- value oznacza, że wartość przypisana do właściwości Name zostanie zapisana do zmiennej _name.

Słowo with jest więc używane, aby połączyć logikę get i set w ramach jednej właściwości.

2. Słowo kluczowe and w tym kontekście służy do połączenia dwóch członów deklaracji. Zamiast oddzielać get i set dwoma różnymi wyrażeniami member, używa się and, aby obie te części były częścią jednej deklaracji właściwości.

Bez użycia and, trzeba by było zdefiniować dwie oddzielne deklaracje:

```
member this.Name with get() = _name
member this.Name with set(value) = _name <- value
```

Przykład 2: Klasa dziedzicząca

Utwórz klasę Employee, która dziedziczy po klasie Person i dodaje dodatkową właściwość Position oraz metodę Work.

```
type Person(name: string, age: int) =
    member this.Name = name
    member this.Age = age

    member this.Greet() =
        printfn "Cześć, jestem %s i mam %d lat." this.Name this.Age


type Employee(name: string, age: int, position: string) =
    inherit Person(name, age)

    member this.Position = position

    member this.Work() =
        printfn "%s pracuje jako %s." this.Name this.Position

[<EntryPoint>]
let main argv =
    let employee = Employee("Anna", 28, "Programista")
    employee.Greet()
    employee.Work()

    0
```

Przykład 3: Interfejs

Stwórz interfejs IAnimal z metodą Speak, a następnie utwórz klasy Dog i Cat, które implementują ten interfejs.

```
type IAnimal =
    abstract member Speak: unit -> string

type Dog() =
    interface IAnimal with
        member this.Speak() = "Hau hau!"

type Cat() =
    interface IAnimal with
        member this.Speak() = "Miau!"
```

```
[<EntryPoint>]
let main argv =
    let animals: IAnimal list = [ Dog() :> IAnimal; Cat() :> IAnimal ]

    for animal in animals do
        printfn "Zwierzę mówi: %s" (animal.Speak())

    0
```

Przykład 4: Użycie kolekcji obiektów

Utwórz kolekcję obiektów Person i wykonaj operacje, takie jak filtrowanie i wyświetlanie.

```
type Person(name: string, age: int) =
    member this.Name = name
    member this.Age = age

[<EntryPoint>]
let main argv =
    let people = [
        Person("Jan", 30)
        Person("Anna", 25)
        Person("Krzysztof", 35)
    ]

    let filteredPeople = people |> List.filter (fun p -> p.Age > 28)
    printfn "Osoby powyżej 28 roku życia:"
    for person in filteredPeople do
        printfn "Nazwa: %s, Wiek: %d" person.Name person.Age

    0
```

Przykład 5: Konstruktory i przeciążanie metod

Utwórz klasę Rectangle z różnymi konstruktorami i metodami obliczającymi pole oraz obwód.

```
type Rectangle(width: float, height: float) =
    member this.Width = width
    member this.Height = height

    member this.Area() = this.Width * this.Height
    member this.Perimeter() = 2.0 * (this.Width + this.Height)

[<EntryPoint>]
let main argv =
    let rect = Rectangle(5.0, 10.0)
    printfn "Pole prostokąta: %.2f" (rect.Area())
    printfn "Obwód prostokąta: %.2f" (rect.Perimeter())

    0
```

Zadania do samodzielnego rozwiązania

Rozwiązania w postaci niezbędnych plików źródłowych należy przesłać do utworzonego zadania na platformie e-learningowej zgodnie ze zdefiniowanymi instrukcjami oraz w nieprzekraczalnym wyznaczonym terminie.

Zadanie 1. System do zarządzania biblioteką

Zaproponuj system do zarządzania biblioteką, który będzie zwierał klasy do reprezentowania książek, użytkowników biblioteki oraz samej biblioteki, z funkcjonalnością do dodawania i usuwania książek oraz wypożyczania ich użytkownikom. Program powinien zawierać:

- Klasa Book: Reprezentuje książkę z tytułem, autorem i liczbą stron. Posiada metodę GetInfo() zwracającą informacje o książce.
- Klasa User: Reprezentuje użytkownika biblioteki, który może wypożyczać i zwracać książki. Posiada metody do wypożyczania (BorrowBook) oraz zwracania (ReturnBook) książek.
- Klasa Library: Reprezentuje bibliotekę, która zarządza kolekcją książek. Posiada metody do dodawania (AddBook), usuwania (RemoveBook) oraz listowania (ListBooks) książek.
- Program główny: Tworzy instancję biblioteki i użytkownika, dodaje książki do biblioteki, wypożycza i zwraca książki przez użytkownika oraz wyświetla aktualny stan biblioteki.

Zadanie 2. System BankAccount

Zaproponuj system do zarządzania kontami bankowymi, który będzie implementował operacje CRUD. W systemie należy zaimplementować:

Klasa BankAccount: Reprezentuje konto bankowe. Posiada:

- Właściwość AccountNumber do przechowywania numeru konta.
- Właściwość Balance do odczytu aktualnego salda.
- Metody Deposit do wpłat i Withdraw do wypłat, z odpowiednimi sprawdzeniami.

Klasa Bank: Zarządza kontami bankowymi. Posiada metody do:

- Tworzenia kont (CreateAccount).
- Odczytu kont (GetAccount).
- Aktualizacji salda konta (UpdateAccount).
- Usuwania kont (DeleteAccount).

Program główny:

- Tworzy instancję banku i wykonuje operacje CRUD na kontach bankowych.
- Demonstracja działań obejmuje tworzenie kont, wpłaty, wypłaty, aktualizację salda oraz usuwanie kont.