LEKCJA 2 – OOP

Przypomnienie OOP w Javie

Pytanie: Czym jest klasa?

Odpowiedź:
Klasa to szablon obiektu. Określa jego pola i metody.

Pytanie: Co to enkapsulacja?

Odpowiedź:
Ukrywanie danych i udostępnianie ich przez metody (gettery/settery).

Pytanie: Jak działa dziedziczenie w Javie?

Odpowiedź:
Używamy `extends`, a klasa potomna dziedziczy pola i metody klasy bazowej.

Pytanie: Co robi `@Override`?

Odpowiedź:
Oznacza, że metoda nadpisuje metodę z klasy bazowej.

Przykład z Javy

```
class Animal {
   public void makeSound() {
      System.out.println("Sound");
   }
}

class Dog extends Animal {
   @Override
   public void makeSound() {
      System.out.println("Woof");
   }
}
```

Klasy i Properties (duża różnica)

Klasa w C#

```
public class Animal
{
   public string Name { get; set; }

   public void MakeSound()
   {
      Console.WriteLine("Sound");
   }
}
```

Pytanie: Co oznacza `{ get; set; }`?

Odpowiedź:
To skrócona składnia właściwości (property). Kompilator automatycznie tworzy getter i setter.

Pytanie: Dlaczego to jest ważne?

Odpowiedź:
Bo w Javie musimy pisać gettery i settery ręcznie.
W C# to jest część języka, a nie tylko konwencja.

Java dla porównania:

```
private String name;

public String getName() {
   return name;
}

public void setName(String name) {
   this.name = name;
}
```

C#:

```
public int Age
      {
         get { return age; } // odczyt
         set
```

```
        {
            if (value >= 0)
            {
                age = value;
            }
            else
            {
                Console.WriteLine("Wiek nie może być ujemny!");
            }
        }
    }
```

Pytanie: Czy property to metoda?

Odpowiedź:
Nie. Składniowo wygląda jak pole, ale wewnętrznie wywołuje metody get/set.

Ograniczenie dostępu

public string Name { get; private set; }

Pytanie: Co to oznacza?
Odpowiedź:
Odczyt publiczny, ale modyfikacja tylko wewnątrz klasy.

Dziedziczenie i virtual / override

C#

```
public class Animal
{
  public virtual void MakeSound()
  {
     Console.WriteLine("Sound");
  }
}

public class Dog : Animal
{
  public override void MakeSound()
```

```
  {
      Console.WriteLine("Woof");
  }
}
```

Pytanie: Dlaczego jest `virtual`?

Odpowiedź:
Bo w C# metoda musi być jawnie oznaczona jako możliwa do nadpisania.

Pytanie: Co się stanie bez `virtual`?

Odpowiedź:
Nie będzie można użyć `override` w klasie potomnej.

Pytanie: Jak jest w Javie?

Odpowiedź:
W Javie każda metoda (poza `final`) jest domyślnie wirtualna.

Wniosek:

C# daje większą kontrolę – musisz świadomie zdecydować, że metoda może być nadpisana.

---

struct vs class (duża różnica koncepcyjna)

W C# mamy dwa typy:

- class → typ referencyjny
- struct → typ wartościowy

Przykład struct:

```
public struct Point
{
   public int X;
   public int Y;
}
  Point p1 = new Point();
            p1.X = 10;
            p1.Y = 20;
```

Pytanie: Gdzie trafia struct?

Odpowiedź:
Zazwyczaj na stos, jako typ wartościowy.

Pytanie: Gdzie trafia class?

Odpowiedź:
Na stertę, jako typ referencyjny.

Pytanie: Czy Java ma odpowiednik struct?

Odpowiedź:
Nie. W Javie wszystko oprócz prymitywów jest obiektem.

Wniosek:

C# daje większą kontrolę nad pamięcią i wydajnością.

Delegaty

Definicja:Delegat w C# to typ danych, który przechowuje referencję do metody i pozwala ją wywołać w późniejszym czasie. Umożliwia przekazywanie metod jako argumentów, dzięki czemu można traktować funkcje jak dane i tworzyć bardziej elastyczny kod.

Pytanie: Czym jest delegat?

Odpowiedź:
To typ, który przechowuje referencję do metody.

Pytanie: Jak wygląda odpowiednik w Javie?

Odpowiedź:
Używamy interfejsów funkcyjnych.

Wbudowane delegaty

W C# istnieją gotowe, uniwersalne delegaty, które pokrywają większość przypadków użycia.
Najczęściej używane to:

- `Action`
- `Func`
- dodatkowo: `Predicate`

Definicja

- Action to wbudowany delegat reprezentujący metodę, która nie zwraca żadnej wartości, ale może przyjmować parametry.
- Func to wbudowany delegat reprezentujący metodę, która przyjmuje parametry i zwraca wartość.

Action:

Charakterystyka

- Zawsze zwraca `void`
- Może mieć od 0 do 16 parametrów
- Jest skrótem, który zastępuje własną definicję delegata

Kiedy używamy Action?

- do przekazywania metod jako argumentów
- do obsługi zdarzeń (eventów)
- w metodach typu `ForEach`
- gdy chcemy wykonać jakąś operację bez zwracania wyniku

Func

Definicja

Func to wbudowany delegat reprezentujący metodę, która przyjmuje parametry i zwraca wartość.

Charakterystyka

- Może mieć parametry
- Zawsze zwraca wartość
- Ostatni typ w definicji oznacza typ zwracany

Kiedy używamy Func?

- do obliczeń
- w LINQ
- gdy chcemy przekazać metodę, która coś zwraca
- w programowaniu funkcyjnym

Cecha	Action	Func
Zwraca wartość	❌ Nie	✅ Tak
Może mieć parametry	✅ Tak	✅ Tak
Typ zwracany	zawsze void	określony jako ostatni typ

Pytanie: Dlaczego delegaty są ważne?

Odpowiedź:

- Bo są podstawą LINQ,
- są używane w całym ekosystemie .NET,
- pozwalają pisać bardziej nowoczesny i elastyczny kod.

Podsumowanie

Tabela do omówienia:

Temat	Java	C#
Klasy	class	class
Konstruktor	metoda o nazwie klasy	metoda o nazwie klasy
Pola i enkapsulacja	prywatne + get/set	private + properties
Dziedziczenie	extends	:
Override	@Override	virtual + override
Polimorfizm	działa domyślnie	działa na metodach virtual

Typy wartościowe	tylko prymitywy	struct
Funkcje jako typ	interfejsy funkcyjne	delegaty

Ostatnie pytanie: Czy C# to tylko kopia Javy?

Odpowiedź:
Nie. Składnia jest podobna, ale C# daje większą kontrolę nad dziedziczeniem, pamięcią i funkcjami jako typami.