Politechnika Wrocławska

Programowanie obiektowe

Wykład 8

Metody

Dziedziczenie Polimofrizm

Prowadzący
dr inż. Paweł Maślak

Plan wykładu

- Metody
- Dziedziczenie
- Polimorfizm

Metody

- Metody w języku C# to bloki kodu, które grupują instrukcje wykonywane w celu wykonania określonego zadania. Mogą być wywoływane przez inne części kodu, dzięki czemu kod jest bardziej modularny i łatwiejszy do zrozumienia, a także ułatwia ponowne użycie.

Metody - rodzaje

- **Metody statyczne** – są to metody, które nie wymagają tworzenia instancji klasy, aby je wywołać. Mogą być wywoływane bezpośrednio z klasy, do której należą.
- **Metody niestatyczne** – są to metody, które muszą być wywoływane na instancji klasy. Mogą mieć dostęp do pól i metod instancji klasy.
- **Metody wirtualne** – są to metody, które można przesłaniać w klasach pochodnych. Metoda wirtualna w klasie bazowej może być zastąpiona przez metodę pochodną w klasie dziedziczącej.

Metody - rodzaje

- **Metody abstrakcyjne** – są to metody, które nie posiadają ciała metody. Metody abstrakcyjne są zadeklarowane w klasie abstrakcyjnej i muszą być zaimplementowane w klasach pochodnych.
- **Metody zwracające wartość** – są to metody, które zwracają wartość określonego typu po zakończeniu swojego działania.

Metody - rodzaje

- Metody w klasach w języku C# są szczególnym rodzajem funkcji, które są związane z klasą i jej instancjami.
- Pozwalają one na definiowanie funkcjonalności, która może być wywoływana na instancjach danej klasy.
- Umożliwiają abstrakcyjne definiowanie zachowań, które obiekty danej klasy powinny realizować.

Metody - przykład

```
public class Car
{
  public void StartEngine()
  {
    Console.WriteLine("Engine works!");
  }
}
```

W powyższym przykładzie **klasa Car** posiada **metodę StartEngine**(), która nie przyjmuje żadnych parametrów i nie zwraca żadnej wartości (typ zwracany void). Metoda ta wyświetla w konsoli napis "Engine works!".

Wywoływanie metod na instancjach klasy

- Metody w klasach są związane z instancjami klasy, więc aby wywołać metodę na konkretnej instancji klasy, musimy utworzyć tę instancję i odwołać się do niej w wywołaniu metody.
- Wywołanie metody StartEngine() na instancji klasy Car:

```
Car myCar = new Car();
myCar.StartEngine();
```

Wywoływanie metod na instancjach klasy

- Metody w klasach mogą przyjmować parametry, które są przekazywane do nich podczas wywoływania.
- Przykład metody w **klasie Car** przyjmującej parametr typu **string**:

```
public class Car
{
  public void ShowBrand(string brand)
  {
    Console.WriteLine("Marka samochodu: " + brand);
  }
}
```

Wywoływanie metod na instancjach klasy

- Metody w klasach mogą zwracać wartości, które są przetwarzane wewnątrz bloku instrukcji i zwracane do miejsca, z którego została wywołana metoda.

```
public class Car
{
  int yearOfProduction;
  public Car(int yearOfProduction)
  {
    this.yearOfProduction =
      yearOfProduction;
  }
  public int GetYearOfProduction()
  {
    return this.yearOfProduction;
  }
}
```

Wywoływanie metod na instancjach klasy

- Oczywiście należy utworzyć obiekt klasy Car, a dopiero później można zwróconą wartość przypisać do zmiennej:

```
Car myCar = new Car(2023);
int yearOfProduction = myCar.GetYearOfProduction();
Console.WriteLine("Rok produkcji samochodu: " +
yearOfProduction);
```

Wywoływanie metod na instancjach klasy

```
Car myCar = new Car(2023);
int yearOfProduction = myCar.GetYearOfProduction();
Console.WriteLine("Rok produkcji samochodu: " +
yearOfProduction);
```

W tym przykładzie utworzyliśmy instancję klasy Car o nazwie myCar. Obiekt ten w czasie tworzenia za pomocą konstruktora klasy Car, otrzymał wartość dla parametru yearOfProduction równą 2023. Następnie na tej instancji wywołano metodę GetProductionYear(), a zwrócona przez nią wartość (2023) została zapisana do zmiennej globalnej yearOfProduction, a następnie wyświetlona w konsoli razem z napisem „Rok produkcji samochodu:”.

Wywoływanie metod na instancjach klasy

- Połączenie danych w klasie

```
public class Car
{
  public void ShowData(string brand)
  {
    Console.WriteLine("Marka samochodu: " + brand);
  }

  public void ShowData(string brand, int yearOfProduction)
  {
    Console.WriteLine("Marka samochodu: " + brand + ", Rok produkcji: " +
yearOfProduction);
  }
  static void Main(string[] args)
  {
    Car myCar = new Car();
    myCar.ShowData("Toyota");
```

Wywoływanie metod na instancjach klasy

```
public class Car
{
  public void ShowData(string brand)
  {
    Console.WriteLine("Marka samochodu: " + brand);
  }

  public void ShowData(string brand, int yearOfProduction)
  {
    Console.WriteLine("Marka samochodu: " + brand + ", Rok produkcji: " +
yearOfProduction);
  }
  static void Main(string[] args)
  {
    Car myCar = new Car();
    myCar.ShowData("Toyota");
  }
}
```

```
Marka samochodu: Toyota

Process finished with exit code 0
```

Wywoływanie metod na instancjach klasy

```
public class Car
{
  public void ShowData(string brand)
  {
    Console.WriteLine("Marka samochodu: " + brand);
  }

  public void ShowData(string brand, int yearOfProduction)
  {
    Console.WriteLine("Marka samochodu: " + brand + ", Rok produkcji: " +
yearOfProduction);
  }
  static void Main(string[] args)
  {
    Car myCar = new Car();
    myCar.ShowData("Toyota", 2023);
  }
}
```

```
Marka samochodu: Toyota, Rok prod
ukcji: 2023

Process finished with exit code 0
```

Przeciążanie metody

```
class Calculator
{
  public static int Add(int x, int y) { return x + y; }
  public static double Add(double x, double y) { return x + y; }
  public static string Add(string x, string y) { return x + y; }

  static void Main(string[] args)
  {
    int x = 5;
    int y = 7;
    Console.WriteLine(Add(x, y));
  }
}
```

```
12

Process finished with exit code 0
```

Przeciążanie metody

```
class Calculator
{
  public static int Add(int x, int y) { return x + y; }
  public static double Add(double x, double y) { return x + y; }
  public static string Add(string x, string y) { return x + y; }

  static void Main(string[] args)
  {
    double x = 5.4;
    double y = 7.32;
    Console.WriteLine(Add(x, y));
  }
}
```

```
12,72

Process finished with exit code 0
.
```

Przeciążanie metody

```
class Calculator
{
  public static int Add(int x, int y) { return x + y; }
  public static double Add(double x, double y) { return x + y; }
  public static string Add(string x, string y) { return x + y; }

  static void Main(string[] args)
  {
    string x = "5";
    string y = "7";
    Console.WriteLine(Add(x, y));
  }
}
```

```
57

Process finished with exit code 0
```

Dziedziczenie

- Dziedziczenie to proces, w którym jedna klasa dziedziczy pola i metody z innej klasy. Klasa, która dziedziczy, nazywana jest podklasą, a klasa, z której dziedziczy, nazywana jest nadklasą lub klasą bazową.
- Dziedziczenie w C# pozwala na tworzenie bardziej specjalizowanych klas na podstawie istniejących klas. Podklasy dziedziczą pola i metody z nadklasy, co umożliwia programistom ponowne wykorzystanie kodu.

Dziedziczenie

- Dziedziczenie pozwala nam na zdefiniowanie klasy w uogólnieniu innej klasy, co pozwala na łatwiejsze tworzenie i zarządzanie aplikacją. Daje również możliwość ponowego wykorzystania kodu i przyśpiesza czas jego implementacji.
- Podczas tworzenia nowej klasy, zamiast pisać zupełnie od nowa wszystkie składowe tej klasy, programista może powiedzieć, że nowa klasa ma dziedziczyć z istniejącej już klasy. Istniejąca klasa nazywana jest klasą bazową a nowa klasa dziedzicząca po klasie bazowej nosi nazwę klasy pochodnej.

Dziedziczenie

- Klasa może dziedziczyć z jednej klasy, ale może implementować wiele interfejsów.
- Składnia dziedziczenia:

```
    modyfikator_dostepu class klasa_bazowa
    {
…
    }
    class klasa_pochodna : klasa_bazowa
    {
…
    }
```

Dziedziczenie

```
using System;
namespace Dziedziczenie
{
 class Program
 {
  static void Main(string[] args)
  {
   Prostokat pr = new Prostokat();
   pr.UstawSzerokosc(4);
   pr.UstawWysokosc(5);
   // Obliczenie powierzchni
   Console.WriteLine("Powierzchnia prostokąta: {0}", pr.ObliczPowirzchnie());
   Console.ReadKey(); // Wynik działania programu Pow. prostokata: 20
  }
 }

 // klasa bazowa
 class Ksztalt
 {
```

Dziedziczenie

```
// klasa bazowa
class Ksztalt
{
  // modyfikator dostepu protected
  // pola dostepne sa dla klasy oraz klas, której po niej dziedziczą
  // gdybyśmy zastosowali modyfikator dostępu private
  // pole byłoby dostępne tylko dla tej klasy
  protected int szerokosc;
  protected int wysokosc;

  public void UstawWysokosc(int w)
  {
    wysokosc = w;
  }
  public void UstawSzerokosc(int s)
  {
    szerokosc = s;
  }
}
```

Dziedziczenie

```
// klasa pochodna
class Prostokat:Ksztalt
{
  public int ObliczPowirzchnie()
  {
    return wysokosc * szerokosc;
          // mamy dostęp do pól z klasy bazowej
  }
}
}
```

Dziedziczenie - cechy

- Dziedziczenie jest jednokierunkowe – klasa pochodna dziedziczy po klasie bazowej, ale klasa bazowa nie dziedziczy po klasie pochodnej.
- Dziedziczenie umożliwia dostęp do pól i metod klasy bazowej, ale nie pozwala na modyfikację ich wartości w klasie pochodnej.
- Klasa pochodna może przesłaniać (ang. override) metody klasy bazowej, co pozwala na zmianę ich zachowania w kontekście klasy pochodnej.

Dziedziczenie - cechy

- Dziedziczenie umożliwia tworzenie hierarchii klas, co ułatwia organizację kodu i zwiększa jego czytelność.
- Dzięki dziedziczeniu można uniknąć powtarzania się kodu - klasy pochodne mogą korzystać z funkcjonalności klasy bazowej.

Dziedziczenie - przykład

```
public class Animal
{
 public string Name;
 public int Age;

 public virtual void MakeSound()
 {
  Console.WriteLine("The animal makes a sound.");
 }
}
 //klasa pochodna
public class Dog : Animal
{
 public override void MakeSound()
 {
  Console.WriteLine("The dog barks.");
 }
}
 //klasa pochodna
public class Cat : Animal{
 public override void MakeSound()
 {
  Console.WriteLine("The cat meows.");
 }
}
```

Dziedziczenie z kilku klas bazowych

```
public class Person
{
  public string Name;
  public int Age;
}

public class Employee : Person
{
  public int EmployeeId;
  public double Salary;
}

public class Manager : Employee
{
  public List<Employee> Subordinates;
}
```

Polimorfizm

- Polimorfizm to zdolność do wykorzystywania jednej nazwy metody do realizacji różnych zadań. W C# polimorfizm pozwala na przesłanianie metod, czyli definiowanie nowej implementacji metody w podklasie, która zastępuje implementację metody w nadklasie.
- Polimorfizm w C# pozwala na bardziej elastyczne i rozszerzalne kodowanie i definiowanie zachowania obiektów w zależności od kontekstu.

Polimorfizm

- Polimorfizm umożliwia wykorzystanie obiektów różnych klas w sposób jednolity, bez konieczności szczegółowego określania ich typu.
- Można to osiągnąć poprzez dziedziczenie i definiowanie metod wirtualnych w klasach bazowych, które mogą być zastąpione przez metody w klasach pochodnych - programista nie musi przewidywać wszystkich możliwych typów obiektów, z którymi będzie miał do czynienia w trakcie działania aplikacji.

Polimorfizm -przykład

```
public class Shape
{
    public virtual double Area()
    {
        return 0;
    }
}
```

```
public class Circle : Shape
{
    int r;
    public Circle(int r)
    {
        this.r = r;
    }
    public override double Area()
    {
        return Math.PI * r * r;
    }
```

```
}

public class Rectangle : Shape
{
    int a, b;
    public Rectangle(int a, int b)
    {
        this.a = a;
        this.b = b;
    }
    public override double Area()
    {
        return a * b;
    }
}
.....
```

Polimorfizm -przykład

```
public class Shape
{
    public virtual double Area()
    {
        return 0;
    }
}
……
```

```
public class Triangle : Shape
{
    int a, h;
    public Triangle(int a, int h)
    {
        this.a = a;
        this.h = h;
    }
    public override double Area()
    {
        return 0.5 * a * h;
    }
}
```

Polimorfizm – przykład

```
public static class Program
{
    public static void Main(string[] args)
    {
        var shapes = new List<Shape>()
        {
            new Rectangle(2,3),
            new Circle(3),
            new Triangle(2,3)
        };
        foreach(Shape shape in shapes)
        {
            Console.WriteLine(shape.Area());
        }
    }
}
```

```
6
28,274333882308138
3

Process finished with exit code 0
```

Polimorfizm – przykład

Za pomocą zmiennej typu Shape – będącego typem klasy bazowej, zostały utworzone obiekty klas pochodnych – Circle i Rectangle

```
public static class Program2
{
    public static void Main(string[] args)
    {
        Shape circle = new Circle(5);
        Shape rectangle = new Rectangle(5, 5);

        Console.WriteLine(circle.Area());
        Console.WriteLine(rectangle.Area());
    }
}
```

Polimorfizm

- Metody wirtualne i właściwości umożliwiają klasom pochodnym rozszerzanie klasy bazowej bez konieczności używania implementacji metod klasy bazowej.
- Jeśli chcielibyśmy, aby klasa pochodna miała składową o takiej samej nazwie jak składowa w klasie bazowej, można użyć słowa kluczowego **new**, co spowoduje ukrycie składowej klasy bazowej.

Polimorfizm – *new i override*


```
public class Shape
{
   public virtual double Area()
   {
      return 0;
   }
}
public class Rectangle : Shape
{
   int a, b;
   public Rectangle(int a, int b)
   {
      this.a = a;
      this.b = b;
   }
   public override double Area()
   {
      return a * b;
   }
}
public class Square : Rectangle
{
   int a;
   public Square(int a) : base(a,a)
   {
      this.a = a;
   }
   public new double Area()
   {
      return a * a ;
   }
}
```

Polimorfizm – wywołanie

```
public static class Program
{
    public static void Main(string[] args)
    {
        Rectangle square = new Square(4);
        Console.WriteLine(square.Area());
        Square square1 = new Square(3);
        Console.WriteLine(square1.Area());
    }
}
```

```
16
9

Process finished with exit code 0
```

Polimorfizm

Klasa pochodna, która zastąpiła metodę lub właściwość, nadal może uzyskać dostęp do metody lub właściwości w klasie bazowej przy użyciu słowa kluczowego base.

Polimorfizm

```
public class Base
{
    public virtual void PrintText()
    {
        Console.WriteLine("Hello World!");
    }
}
public class Derived : Base
{
    public override void PrintText()
    {
        base.PrintText();
    }
}
```

```
public static class Program
{
    public static void Main(String[] args)
    {
        Base b = new Base();
        b.PrintText();

        Derived d = new Derived();
        d.PrintText();
    }
}
```

```
Hello World!
Hello World!

Process finished with exit code 0
```

Politechnika Wrocławska

Dziękuje bardzo za uwagę

Dr inż. Paweł Maślak