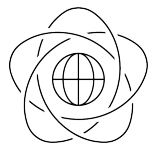
Programming 3 Advanced

Type Testing, Functional Programming, Events, Pattern Matching

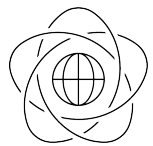
Tomasz Herman

Faculty of Mathematics and Information Science
Warsaw University of Technology

Lecture 6, 14 listopada 2024



Outline

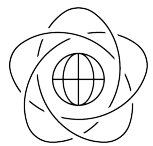


Type Testing

- `is` operator
- `as` operator
- Cast expression
- `typeof` operator

Documentation

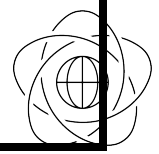
`https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/type-testing-and-cast`



is operator

Type pattern

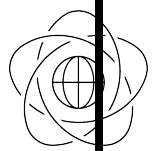
```
1  public class Base { }
2  public class Derived : Base { }
3
4  public static class IsOperatorExample
5  {
6      public static void Main()
7      {
8          object b = new Base();
9          Console.WriteLine(b is Base);  // output: True
10         Console.WriteLine(b is Derived);  // output: False
11
12         object d = new Derived();
13         Console.WriteLine(d is Base);  // output: True
14         Console.WriteLine(d is Derived); // output: True
15     }
16 }
```



is operator

Declaration pattern

```
1  public class Base { }
2  public class Derived : Base
3  {
4      int Foo() => 42;
5  }
6
7  public static class IsOperatorExample
8  {
9      public static void Main()
10     {
11         object b = new Base();
12         if (b is Derived d)
13         {
14             Console.WriteLine($"Foo: {d.Foo()}");
15         }
16     }
17 }
```



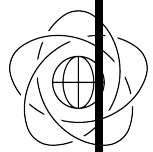
as operator

The as operator

The as operator explicitly converts the result of an expression to a given reference or nullable value type. If the conversion isn't possible, the as operator returns null. Unlike a cast expression, the as operator never throws an exception.

```
E as T
// Is equivalent to:
E is T ? (T)(E) : (T)null
```

```
IEnumerable<int> numbers = new List<int>(){10, 20, 30};
IList<int>? list = numbers as IList<int>;
if (list != null)
{
    Console.WriteLine(list[0] + list[^1]);
}   // output: 40
```

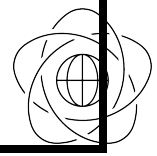


Cast expression

The as operator

A cast expression of the form (T)E performs an explicit conversion of the result of expression E to type T. If no explicit conversion exists from the type of E to type T, a compile-time error occurs. At run time, an explicit conversion might not succeed and a cast expression might throw an exception.

```
1  double x = 1234.7;
2  int a = (int)x;
3  Console.WriteLine(a);   // output: 1234
4
5  List<int> ints = [10, 20, 30];
6  IEnumerable<int> numbers = ints;
7  IList<int> list = (IList<int>)numbers;
8  Console.WriteLine(list.Count);  // output: 3
9  Console.WriteLine(list[1]);  // output: 20
```

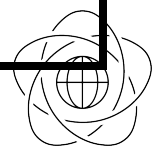


Testing with typeof operator

The typeof operator

Use the typeof operator to check if the run-time type of the expression result exactly matches a given type.

```
1  object b = new Giraffe();
2  Console.WriteLine(b is Animal);  // True
3  Console.WriteLine(b.GetType() == typeof(Animal));  // False
4
5  Console.WriteLine(b is Giraffe);  // True
6  Console.WriteLine(b.GetType() == typeof(Giraffe));  // True
7
8  public class Animal { }
9  public class Giraffe : Animal { }
```



Events Overview

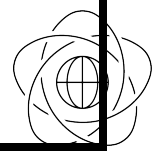
```
1  // Delegate definition
2  public delegate void PriceChangedHandler(decimal oldPrice,
3                                           decimal newPrice);
4
5  public class Broadcaster
6  {
7      // Event declaration
8      public event PriceChangedHandler? PriceChanged;
9  }
```

Documentation

https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/events/

Events

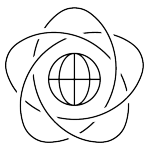
```
1  public class Stock
2  {
3      public event PriceChangedHandler? PriceChanged;
4      private decimal _price;
5      public decimal Price
6      {
7          get => _price;
8          private set
9          {
10             if (_price == value) return;
11             decimal oldPrice = _price;
12             _price = value;
13             if (PriceChanged != null)
14                 PriceChanged(oldPrice, _price);
15         }
16     }
17 }
```



Without the event keyword the code will behave the same way.

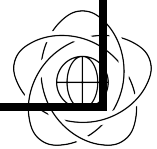
`event` keyword

- Prevents reassigning of the delegate form outside of the class
- Prevents invoking the delegate from outside of the class
- Outside of the class the only option to interact with the event is through += and −= operators.



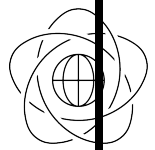
Standard event Pattern

```
1  public class PriceChangedEventArgs : System.EventArgs
2  {
3      public decimal LastPrice { get; }
4      public decimal NewPrice { get; }
5
6      public PriceChangedEventArgs(decimal lastPrice,
7                                   decimal newPrice)
8      {
9          LastPrice = lastPrice;
10         NewPrice = newPrice;
11     }
12 }
13
14 // System EventHandler
15 // delegate void EventHandler<TEventArgs>(object source,
16 //                                        TEventArgs e);
```



Standard event Pattern

```
1  public event EventHandler<PriceChangedEventArgs>? PriceChanged;
2
3  private decimal _price = price;
4  public decimal Price
5  {
6      get => _price;
7      set
8      {
9          if (_price == value) return;
10         decimal oldPrice = _price;
11         _price = value;
12         OnPriceChanged(new PriceChangedEventArgs(oldPrice, _price));
13     }
14 }
15
16 protected virtual void OnPriceChanged(PriceChangedEventArgs e)
17 {
18     if (PriceChanged != null) PriceChanged(this, e);
19 }
```



Pattern Matching

Context

- `is` expression
- `switch` statement
- `switch` expression

Overview

`https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/functional/pattern-matching`

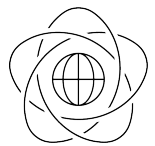
Documentation

`https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/patterns`

Pattern Matching

Types

- Declaration pattern
- Type pattern
- Constant pattern
- Relational patterns
- Logical patterns
- Property pattern
- Positional pattern
- var pattern
- Discard pattern
- List patterns (C# 11)

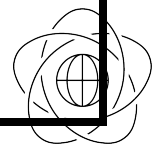


Pattern Matching

is expression

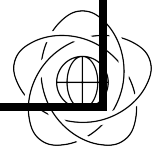
```
variable is pattern
```

```
bool IsWorkingDay(object date)
{
    if (date is null) throw new ArgumentNullException();
    if (date is DateTime dateTime)
    {
        if (dateTime.DayOfWeek is DayOfWeek.Saturday or
                                  DayOfWeek.Sunday)
            return true;
        return false;
    }
    return date is DateOnly {DayOfWeek:
            not (DayOfWeek.Saturday or DayOfWeek.Sunday)};
}
```



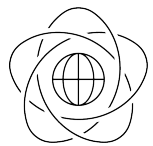
Declaration and Type pattern

```
1  object greeting = "Hello, World!";
2  if (greeting is string message)
3      Console.WriteLine(message.ToLower());
4  public abstract class Vehicle {}
5  public class Car : Vehicle {}
6  public class Truck : Vehicle { public float Load; }
7  public static class TollCalculator
8  {
9      public static decimal CalculateToll(Vehicle vehicle) =>
10     vehicle switch
11     {
12         Car => 2.00m,
13         Truck truck => truck.Load > 100 ? 17.50m : 7.50m,
14         _ => throw new ArgumentException(),
15     };
16 }
```



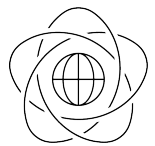
Constant pattern

```
1  string? input = null;
2  if (input is null) { }
3  static decimal GetGroupTicketPrice(int visitorCount) =>
4  visitorCount switch
5  {
6      1 => 12.0m,
7      2 => 20.0m,
8      3 => 27.0m,
9      4 => 32.0m,
10     0 => 0.0m,
11     _ => throw new ArgumentException(),
12 };
```



Relational pattern

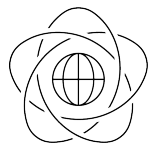
```
1  static string GetCalendarSeason(DateTime date) =>
2  date.Month switch
3  {
4      >= 3 and < 6 => "spring",
5      >= 6 and < 9 => "summer",
6      >= 9 and < 12 => "autumn",
7      12 or (>= 1 and < 3) => "winter",
8      _ => throw new ArgumentOutOfRangeException()
9  };
```



Logical pattern

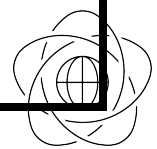
not, and, or

```
1  string? input = null;
2  if (input is not null) { /**/ }
3  static bool IsLetter(char c) =>
4      c is (>= 'a' and <= 'z') or (>= 'A' and <= 'Z');
```



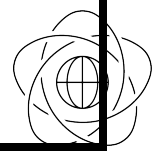
Property pattern

```
1  static bool IsConferenceDay(DateTime date) =>
2  date is { Year: 2020, Month: 5, Day: 19 or 20 or 21 };
3
4  static string TakeFive(object input) => input switch
5  {
6      string { Length: >= 5 } s => s.Substring(0, 5),
7      string s => s,
8
9      ICollection<char> { Count: >= 5 } symbols =>
10         new string(symbols.Take(5).ToArray()),
11     ICollection<char> symbols =>
12         new string(symbols.ToArray()),
13
14     null => throw new ArgumentNullException(),
15     _ => throw new ArgumentException(),
16 };
```



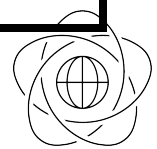
Positional pattern

```
1  public readonly struct Point
2  {
3      public int X { get; }
4      public int Y { get; }
5
6      public Point(int x, int y) => (X, Y) = (x, y);
7
8      public void Deconstruct(out int x, out int y) => (x, y) = (
9  }
10
11 static string Classify(Point point) => point switch
12 {
13     (0, 0) => "Origin",
14     (> 0, 0) => "positive X basis end",
15     (0, > 0) => "positive Y basis end",
16     _ => "Just a point",
17 };
```



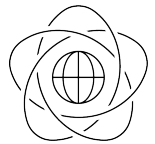
var pattern

```
1  static bool IsAcceptable(int count, int absLimit) =>
2      SimulateDataFetch(count) is var results
3      && results.Min() >= -absLimit
4      && results.Max() <= absLimit;
5
6  static int[] SimulateDataFetch(int count)
7  {
8      var rand = new Random();
9      return Enumerable
10             .Range(start: 0, count: count)
11             .Select(s => rand.Next(minValue: -10,
12                                    maxValue: 11))
13             .ToArray();
14 }
```



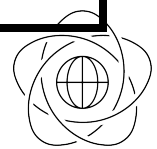
Discard pattern

```
1  static decimal GetDiscountInPercent(DayOfWeek? dayOfWeek) =>
2  dayOfWeek switch
3  {
4      DayOfWeek.Monday => 0.5m,
5      DayOfWeek.Tuesday => 12.5m,
6      DayOfWeek.Wednesday => 7.5m,
7      DayOfWeek.Thursday => 12.5m,
8      DayOfWeek.Friday => 5.0m,
9      DayOfWeek.Saturday => 2.5m,
10     DayOfWeek.Sunday => 2.0m,
11     _ => 0.0m,
12 };
```



List patterns

```
1  int[] numbers = { 1, 2, 3 };
2
3  Console.WriteLine(numbers is [1, 2, 3]);
4  Console.WriteLine(numbers is [1, 2, 4]);
5  Console.WriteLine(numbers is [1, 2, 3, 4]);
6  Console.WriteLine(numbers is [0 or 1, <= 2, >= 3]);
7
8  List<int> list = new() { 1, 2, 3 };
9
10 if (list is [var first, _, _])
11 {
12     Console.WriteLine(
13     $"The first element of a three-item list is {first}.");
14 }
```



List patterns

slice pattern . .

```
1  Console.WriteLine(new[]{ 1, 2, 3, 4 } is [>= 0, .., 2 or 4]);
2  Console.WriteLine(new[]{ 1, 0, 0, 1 } is [1, 0, .., 0, 1]);
3  Console.WriteLine(new[]{ 1, 0, 1 } is [1, 0, .., 0, 1]);
4  string greeting = "Hello world";
5  if (greeting is ['H', .. var rest])
6      Console.WriteLine(rest);
7
8  string Validate(int[] numbers)
9  {
10     return numbers is
11         [< 0, .. { Length: 2 or 4 }, > 0] ?
12             "valid" : "not valid";
13 }
14
15 Console.WriteLine(Validate(new[] { -1, 0, 1 }));
16 Console.WriteLine(Validate(new[] { -1, 0, 0, 1 }));
```