Ранее было рассмотрено нисходящее преобразование типов, которое можно свести к трем вариантам: проверка с помощью ключевого слова **is**, приведение через оператор **as** и отлов исключения при прямом преобразовании.

Например, пусть у нас есть следующие классы:

```
class Employee
{
    public virtual void Work()
    {
        Console.WriteLine("Да работаю я, работаю");
    }
}

class Manager : Employee
{
    public override void Work()
    {
        Console.WriteLine("Отлично, работаем дальше");
    }
    public bool IsOnVacation { get; set; }
}
```

В этом случае преобразование от типа Employee к Manager могло бы выглядеть так:

```
static void UseEmployee1(Employee emp)
{
    Manager manager = emp as Manager;
    if (manager != null && manager.IsOnVacation==false)
    {
        manager.Work();
    }
    else
    {
        Console.WriteLine("Преобразование прошло неудачно");
    }
}
static void UseEmployee2(Employee emp)
{
    try
    {
        Manager manager = (Manager)emp;
        if (!manager.IsOnVacation)
            manager.Work();
    }
    catch (InvalidCastException ex)
    {
        Console.WriteLine(ex.Message);
    }
}
static void UseEmployee3(Employee emp)
{
    if (emp is Manager)
    {
        Manager manager = (Manager)emp;
        if(!manager.IsOnVacation)
            manager.Work();
    }
    else
    {
```

```
        Console.WriteLine("Преобразование не допустимо");
    }
}
```

Все эти методы фактически выполняют одно и то же действие: если переданный в метод объект Employee является объектом Manager и его свойство IsOnVacation равно false, то выполняем его метод Work.

Несмотря на то, что любой из трех случаев довольно прост в использовании, однако функциональность **pattern matching** позволяет нам сократить объем кода:

```
static void Main(string[] args)
{
    Employee emp = new Manager(); //Employee();
    UseEmployee(emp);

    Console.Read();
}

static void UseEmployee( Employee emp)
{
    if (emp is Manager manager && manager.IsOnVacation==false)
    {
        manager.Work();
    }
    else
    {
        Console.WriteLine("Преобразование не допустимо");
    }
}
```

Pattern matching фактически выполняет сопоставление с некоторым шаблоном. Здесь выполняется сопоставление с типом Manager. То есть в данном случае речь идет о **type pattern** - в качестве шаблона выступает тип Manager. Если сопоставление прошло успешно, в переменной manager оказывается объект emp. И далее мы можем вызвать у него методы и свойства.

Также мы можем использовать **constant pattern** - сопоставление с некоторой константой. Например, можно проверить, имеет ли ссылка значение:

```
if (!(emp is null))
{
    emp.Work();
}
```

Кроме выражения if pattern matching может применяться в конструкции switch:

```
static void UseEmployee( Employee emp)
{
    switch (emp)
    {
        case Manager manager:
            manager.Work();
            break;
        case null:
            Console.WriteLine("Объект не задан");
            break;
        default:
            Console.WriteLine("Объект не менеджер");
            break;
```

```
    }
}
```

С помощью выражения **when** можно вводить дополнительные условия в конструкцию case:

```
static void UseEmployee( Employee emp)
{
    switch (emp)
    {
        case Manager manager when manager.IsOnVacation==false:
            manager.Work();
            break;
        case null:
            Console.WriteLine("Объект не задан");
            break;
        default:
            Console.WriteLine("Объект не менеджер");
            break;
    }
}
```

В этом случае опять же преобразуем объект emp в объект типа Manager и в случае удачного преобразования смотрим на значение свойства IsOnVacation: если оно равно false, то выполняется данный блок case.