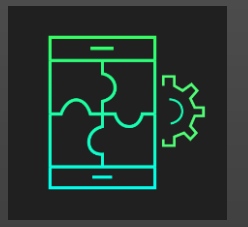
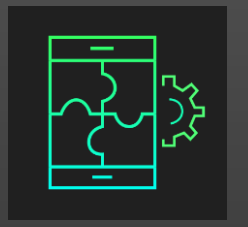
Switch case



Конструкция switch case в C# — это управляющая структура, которая позволяет выполнять различные действия в зависимости от значения переменной (выполнить требуемый блок кода). В некоторых случаях она более удобная и читабельная, служит альтернативой множественному условному выражению if else.

```
switch (одежда.ToLower()) // Использование switch для обработки названия одежды в нижнем регистре
            {
                case "рубашка": // Если 'одежда' равна "рубашка"
                    полка = "Полка для верхней одежды"; // Устанавливаем название полки
                    break; // Выход из switch

                case "брюки": // Если 'одежда' равна "брюки"
                    return "Полка для нижней одежды"; // Возвращаем название полки

                case "платье":
                    return "Полка для женской одежды";

                case "аксессуар":
                    return "Полка для аксессуаров";

                default: // Если 'одежда' не соответствует ни одному из перечисленных случаев
                    throw new ArgumentException("Некорректное название одежды");
                     // Генерация исключения с сообщением об ошибке
            }
            return полка;
}
```

Switch case



break— Оператор **break** используется для немедленного выхода из конструкции **switch-case**. Когда **break** выполняется внутри блока case, выполнение передаётся на следующую строку после всего блока **switch**.

```
case "рубашка": // Если 'одежда' равна "рубашка"
                    полка = "Полка для верхней одежды"; // Устанавливаем название полки
                    break; // Выход из switch
```

default—используется в конструкции **switch-case** для обработки случаев, когда ни один из заданных case не совпадает с выражением. Это функция "по умолчанию" для тех значений, которые не были явно обработаны.

```
default: // Если 'одежда' не соответствует ни одному из перечисленных случаев
                    throw new ArgumentException("Некорректное название одежды"); //
Генерация исключения с сообщением об ошибке
```

goto case—позволяет передавать управление к другому **case** внутри одного и того же блока **switch**. Это позволяет группировать несколько **case** в их обработке, избегая дублирования кода.

```
case "шапка":
  goto case "аксессуар";
   // В случае если юзер набирает слово "шапка",
   //будет переход в case "аксессуар"

case "аксессуар":
  return "Полка для аксессуаров";
```

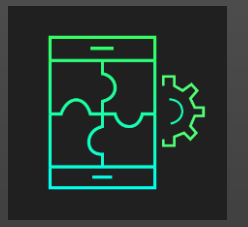
return— Оператор return используется для завершения выполнения метода и возврата значения из него. Если он используется внутри блока **switch**, он может завершить весь метод, в отличие от **break**, который завершает только текущий блок **switch**.

```
return "Полка для нижней одежды";
// Возвращаем название полки
```

throw— Оператор **throw** используется для вызова исключений. В контексте **switch-case**, он может использоваться в **default** или в любом другом case для обработки ситуаций, когда необходимо уведомить о том, что значение не было корректным или не соответствовало ожиданиям.

```
default: // Если 'одежда' не соответствует ни одному из
перечисленных случаев
   throw new ArgumentException("Некорректное название
одежды"); // Генерация исключения с сообщением об ошибке
```

nullable types (?)



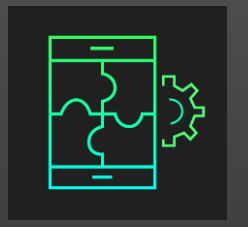
```
string? Димон; // Может быть null
string неДимон; // Не может быть null
```

Nullable Reference Types (NRT) — это новая функция в C#, введённая в версии C# 8.0, которая позволяет разработчикам указывать, может ли ссылка указывать на null или нет. Эта функция помогает улучшить безопасность кода, уменьшая вероятность возникновения ошибок, связанных с null-ссылками (исключение NullReferenceException).

Для определения ссылочного типа как допускающего null, добавляется вопросительный знак “?” после типа, например, string?. Это помогает компилятору отслеживать, где потенциально могут возникнуть проблемы из-за null.

При включении NRT компилятор выдает предупреждения, если есть возможность, что код может вызвать NullReferenceException. Это требует от разработчика больше аккуратности при работе с потенциальными null значениями.

null-forgiving operator (!)



```
string? приз = ПолучитьПриз(); // Возможно, что приз равен null
string вдругОбман = приз!; // Уверен, приз не может быть null
```

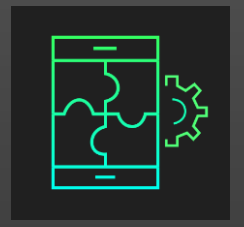
Null-forgiving operator (оператор уверенности в не-null) — это оператор, представленный в C# 8.0, который обозначается символом “!”. Он используется для указания компилятору, что переменная, которая может иметь значение null, в данный момент не является null, и предотвращает предупреждение компилятора о возможном null значении.

Когда вы уверены, что переменная действительно не будет null в определённый момент времени, вы можете использовать оператор “!”, чтобы сообщить компилятору игнорировать предупреждения о возможном null.

Оператор “!” позволяет "сокрыть" предупреждения компилятора о том, что переменная не гарантированно содержит значение, тем самым предоставляя разработчику больше контроля своего кода.

В случае, если метод возвращает null, будет выброшено исключение NullReferenceException во время выполнения. Использование “!” позволяет избавиться от предупреждения компиляции.

Метод ToLower ()



```
switch (одежда.ToLower()) // Использование switch для обработки названия одежды в нижнем регистре
{
    case "рубашка": // Если 'одежда' равна "рубашка"
    полка = "Полка для верхней одежды"; // Задаём название полки
    break; // Выход из switch
```

Метод ToLower в C# — это метод класса String, который используется для преобразования всех символов в строке к нижнему регистру. Он является частью пространства имён System и позволяет обработать строки более удобно, обеспечивая единообразие при сравнении текстов и при вводе данных с учетом регистра.

Метод ToLower() возвращает новую строку, в которой все символы преобразованы к нижнему регистру. Оригинальная строка остаётся неизменной, так как строки в C# являются неизменяемыми (immutable).