



Инструменты рефакторинга

Подготовил
студент гр. 750503
Гук Вячеслав

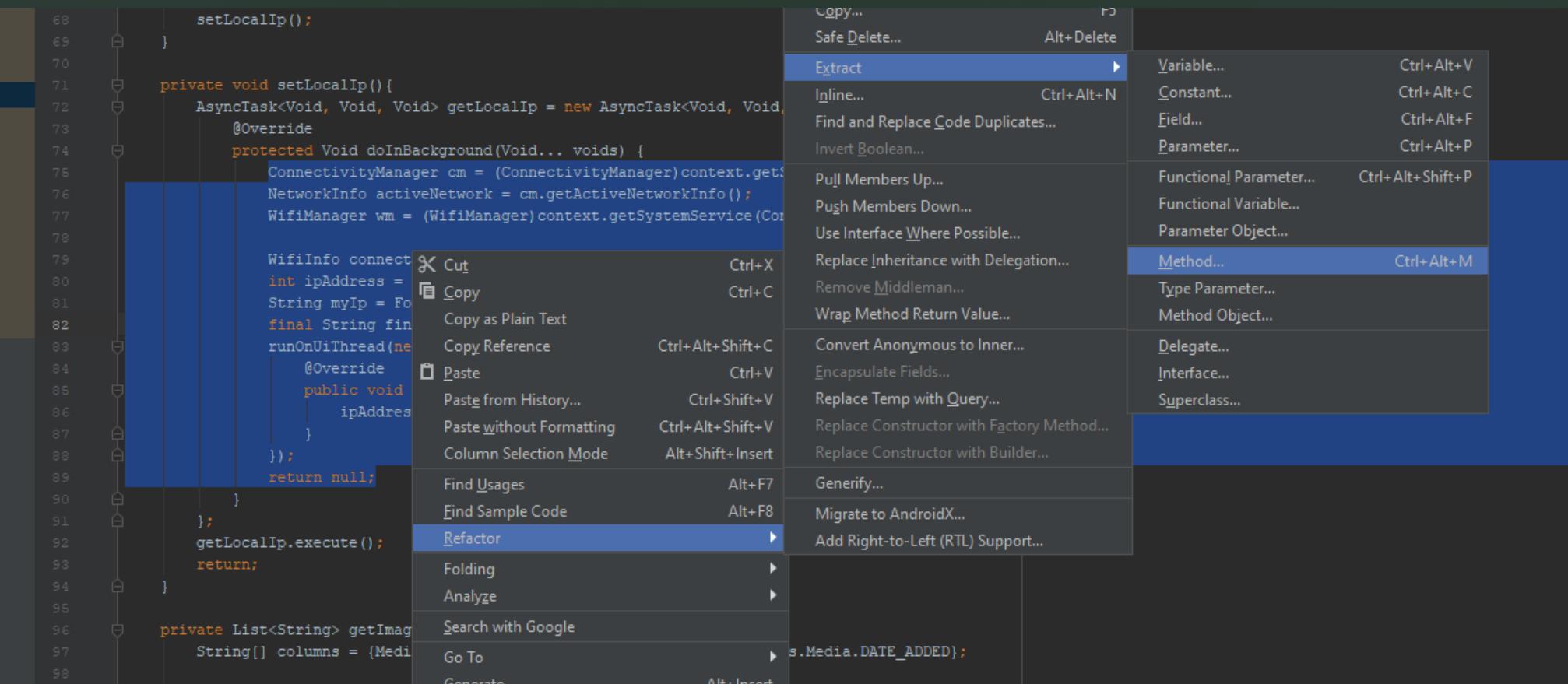
IntelliJ IDEA

IntelliJ IDEA — интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains.

Она предоставляет очень удобные инструменты рефакторинга.

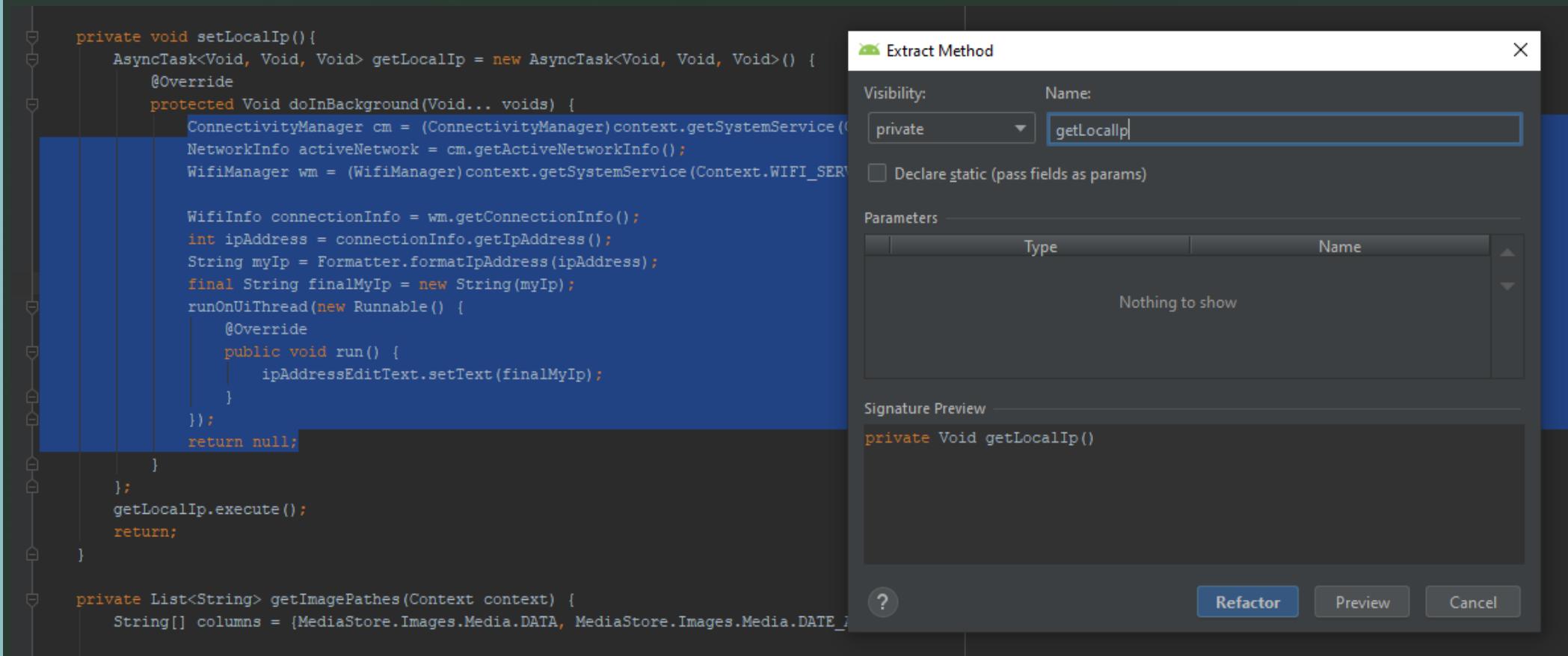
IntelliJ IDEA

Одним из удобных инструментов рефакторинга, встроенным в IntelliJ IDEA является выделение метода. Для того, чтобы вынести участок кода в отдельный метод, нужно его выделить, щелкнуть правой кнопкой мыши на выделение, выбрать Refactor -> Extract -> Method... Либо нажать сочетание клавиш Ctrl+Alt+M.



IntelliJ IDEA

В появившемся окне необходимо выбрать модификатор доступа к новому методу, а так же ввести его название, после чего необходимо нажать на клавишу Refactor и IntelliJ IDEA сама выделит новый метод.



IntelliJ IDEA

Результат представлен на скриншоте.

```
private void setLocalIp(){
    AsyncTask<Void, Void, Void> getLocalIp = new AsyncTask<Void, Void, Void>() {
        @Override
        protected Void doInBackground(Void... voids) {
            return getLocalIp();
        }
    };
    getLocalIp.execute();
    return;
}

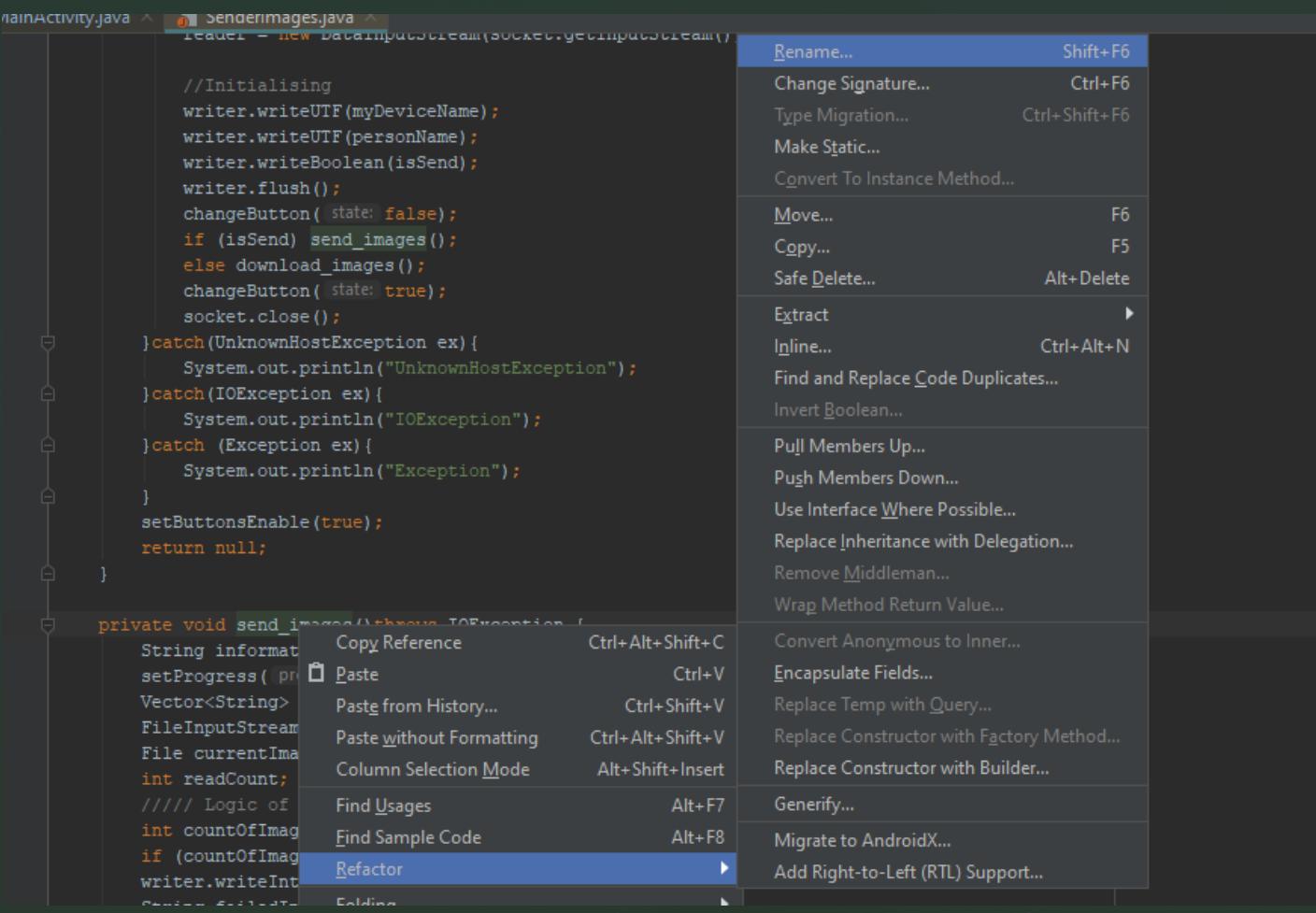
private Void getLocalIp() {
    ConnectivityManager cm = (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo activeNetwork = cm.getActiveNetworkInfo();
    WifiManager wm = (WifiManager) context.getSystemService(Context.WIFI_SERVICE);

    WifiInfo connectionInfo = wm.getConnectionInfo();
    int ipAddress = connectionInfo.getIpAddress();
    String myIp = Formatter.formatIpAddress(ipAddress);
    final String finalMyIp = new String(myIp);
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            ipAddressEditText.setText(finalMyIp);
        }
    });
    return null;
}
```

IntelliJ IDEA

Еще одним удобным инструментом рефакторинга, встроенным в IntelliJ IDEA является переименование.

Для того, чтобы изменить название метода или переменной во всех местах, где они используются, необходимо щелкнуть правой кнопкой мыши на выделение, выбрать Refactor -> Rename... Либо нажать сочетание клавиш Shift+F6.



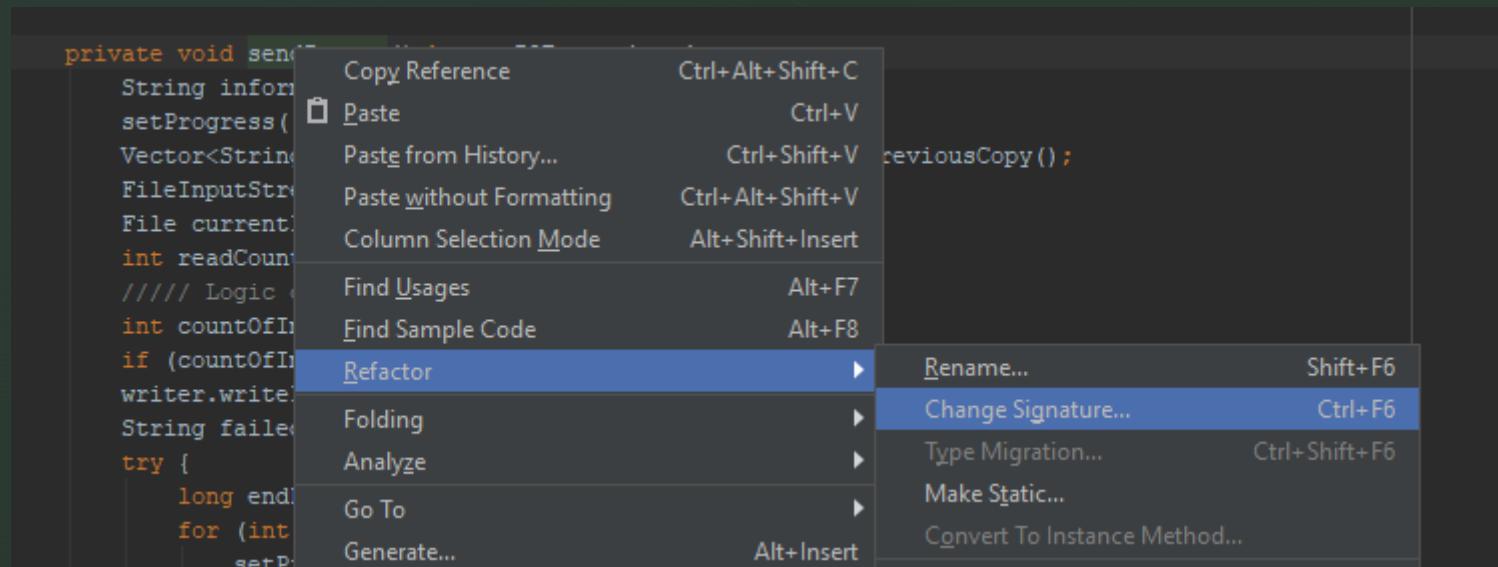
IntelliJ IDEA

После чего нужно ввести новое название, либо выбрать из предложенных вариантов. Название метода или переменной поменяется автоматически во всех местах использования.

```
private void send_images() throws IOException {
    String : sendImages
    setProg: send_images
    Vector<: Press Shift+F6 to show dialog with more options getImagesFrom
    FileInputStream fileReader;
    File file = new File("C:\\" + fileReader.getName());
```

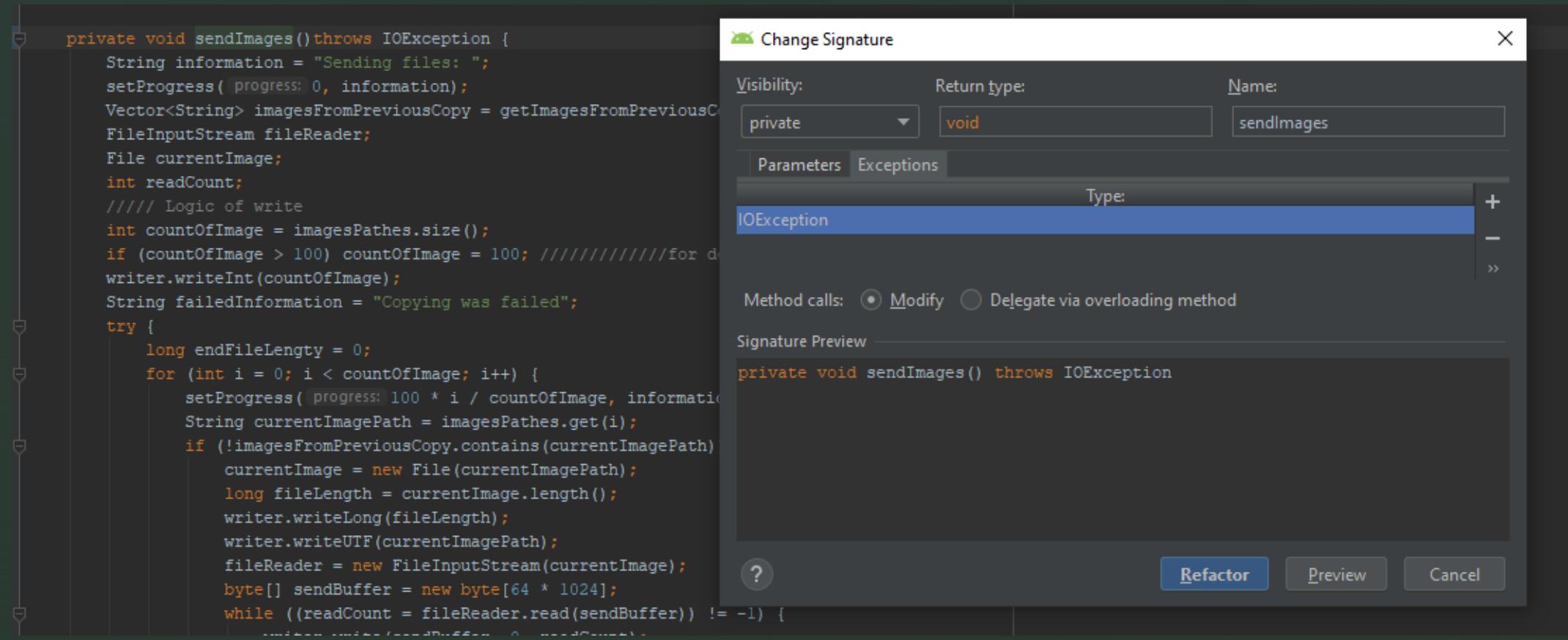
IntelliJ IDEA

Изменить не только название, но и возвращаемый тип, бросаемые исключения, параметры или тип метода можно с помощью ПКМ -> Refactor -> Change Signature... Либо с помощью сочетания клавиш Ctrl+F6



IntelliJ IDEA

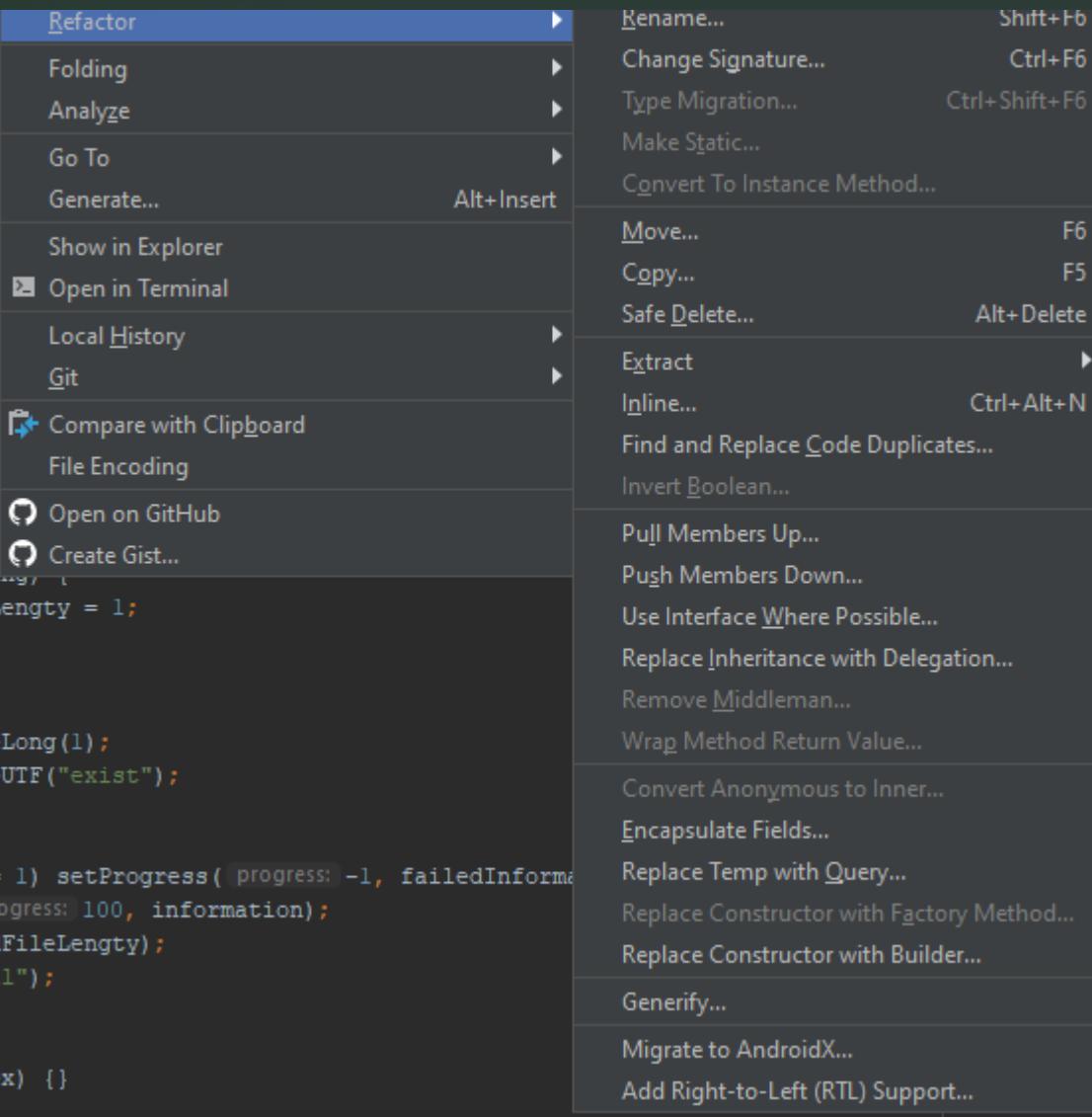
Это всё меняется в удобном диалоговом окне



IntelliJ IDEA

Так же через меню Refactor можно:

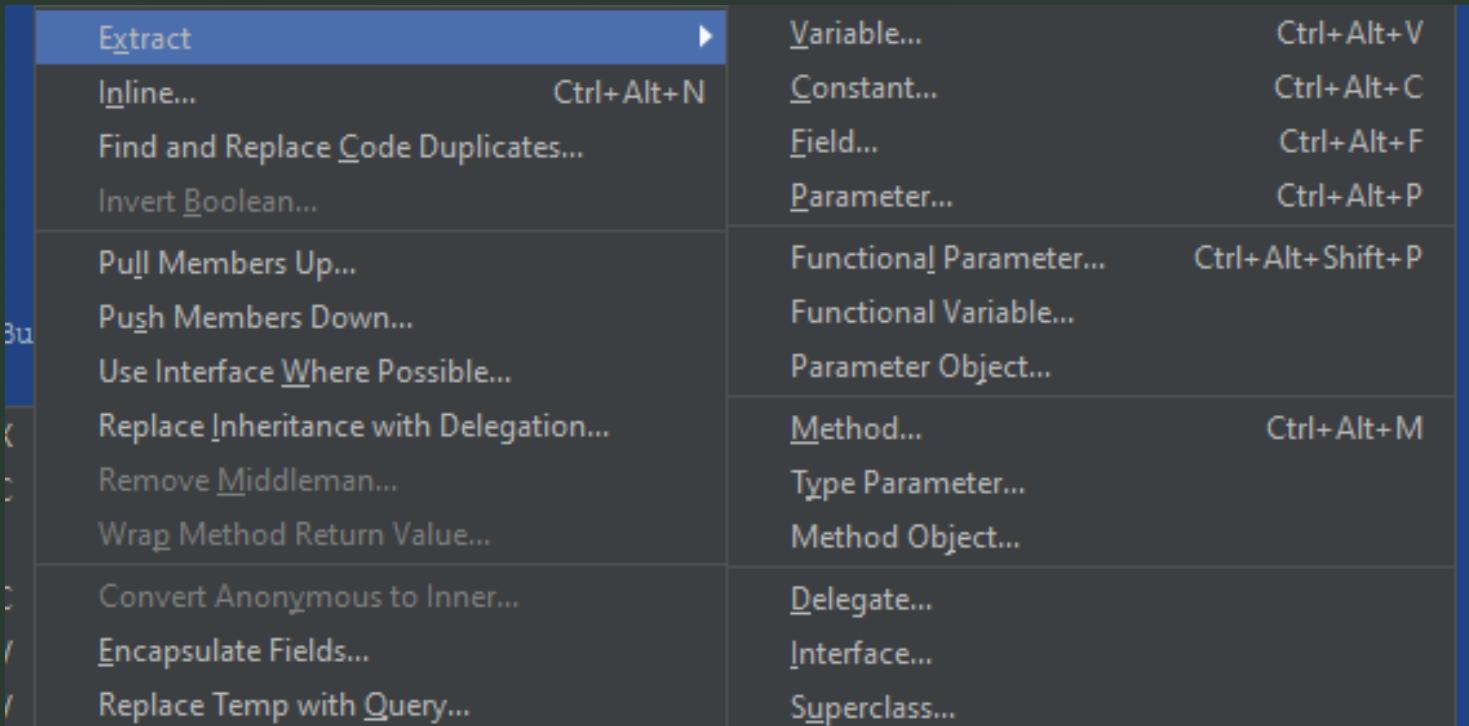
- Сделать статическими (например метод или переменную)
- Изменить тип
- Переместить
- Скопировать
- Безопасно удалить (с проверкой на использование)
- Собрать в одну строку
- Найти повторения в коде
- И многое другое



IntelliJ IDEA

Так же через меню Refactor -> Extract можно:

- Выделить значение
- Выделить константу
- Выделить поле
- Выделить параметр
- Выделить делегат
- Выделить метод
- Выделить интерфейс
- И другое

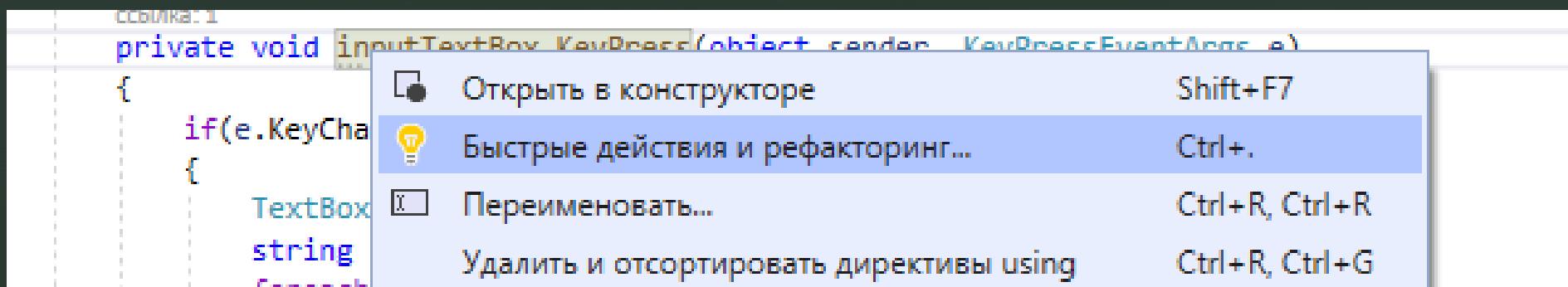


Visual Studio

Microsoft Visual Studio — линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств.

Visual Studio

Для того, чтобы провести рефакторинг в Visual Studio необходимо нажать ПКМ на код, и выбрать «Быстрые действия и рефакторинг», либо воспользоваться сочетанием клавиш Ctrl+.



Visual Studio

Здесь можно переименовать метод. При чем, если есть нарушение правил именования, среда сама подскажет где, и как их исправить.

The screenshot shows a code editor in Visual Studio with the following code:

```
79     private void inputTextBox_KeyPress(object sender, KeyPressEventArgs e)
80
81     Устранитe нарушение имени: InputTextBox_KeyPress
82     Изменить подпись...
83
84     Свернуть каждый параметр
85     Развернуть все параметры и добавить отступы для них
86
87     Настройка или подавление проблем
88             addLine();
89             clearInput();
90
91         }
92     public void addOutputSymbol(char message)
```

A context menu is open at line 80, item 81. It contains the following options:

- Устранитe нарушение имени: InputTextBox_KeyPress
- Изменить подпись...
- Свернуть каждый параметр
- Развернуть все параметры и добавить отступы для них
- Настройка или подавление проблем

To the right of the menu, a tooltip provides more details about the naming violation:

IDE1006 Нарушение правила именования: Эти слова должны начинаться с прописных символов: inputTextBox_KeyPress

The code editor highlights the method names `inputTextBox_KeyPress` and `InputTextBox_KeyPress`. A "Change Style Options" button is visible below the tooltip, and a "Предварительный просмотр изменений" (Preview changes) button is at the bottom.

Visual Studio

Так же здесь можно свернуть и развернуть параметры

A screenshot of the Visual Studio IDE showing a context menu over some C# code. The menu has three main items: 'Изменить подпись...' (Change signature...), 'Свернуть каждый параметр' (Collapse each parameter), and 'Развернуть все параметры и добавить отступы для них' (Expand all parameters and add indentation for them). The third item is currently selected. A submenu under it contains 'Выровнять свернутые параметры' (Align collapsed parameters), 'Добавить отступы для всех параметров' (Add indentation for all parameters), and 'Создать отступы для свернутых параметров' (Create indentation for collapsed parameters). The code itself shows a foreach loop and a public method. A preview window at the bottom right shows the code after applying the selected changes.

```
80 Изменить подпись...
81 Свернуть каждый параметр
82 Разворнуть все параметры и добавить отступы для них
83
84     foreach (char a in input)
85         sendChar(a);
86         addOutputSymbol('\n');
87         addLine();
88         clearInput();
89     }
90 }
91 public void addOutputSymbol(char message)
```

...
private void InputTextBox_KeyPress(object sender, KeyPressEventArgs e)
private void InputTextBox_KeyPress(object sender,
 KeyPressEventArgs e)
{
...
Предварительный просмотр изменений

A screenshot of the Visual Studio IDE showing a context menu over some C# code. The menu has three main items: 'Изменить подпись...' (Change signature...), 'Свернуть каждый параметр' (Collapse each parameter), and 'Развернуть все параметры и добавить отступы для них' (Expand all parameters and add indentation for them). The third item is currently selected. A submenu under it contains 'Выровнять свернутые параметры' (Align collapsed parameters), 'Добавить отступы для всех параметров' (Add indentation for all parameters), and 'Создать отступы для свернутых параметров' (Create indentation for collapsed parameters). The code shows a foreach loop and a public method. A preview window at the bottom right shows the code after applying the selected changes.

```
79 private void InputTextBox_KeyPress(object sender, KeyPressEventArgs e)
80 Изменить подпись...
81 Свернуть каждый параметр
82 Разворнуть все параметры и добавить отступы для них
83
84     foreach (char a in input)
85         sendChar(a);
86         addOutputSymbol('\n');
87         addLine();
88         clearInput();
89     }
90 }
```

...
private void InputTextBox_KeyPress(object sender, KeyPressEventArgs e)
private void InputTextBox_KeyPress(
 object sender, KeyPressEventArgs e)
{
...
Предварительный просмотр изменений

Visual Studio

Если был выделен фрагмент кода, то Visual Studio предложит выделить его в отдельный метод, при нажатии ПКМ -> «Быстрые действия и рефакторинг» (Ctrl+.)

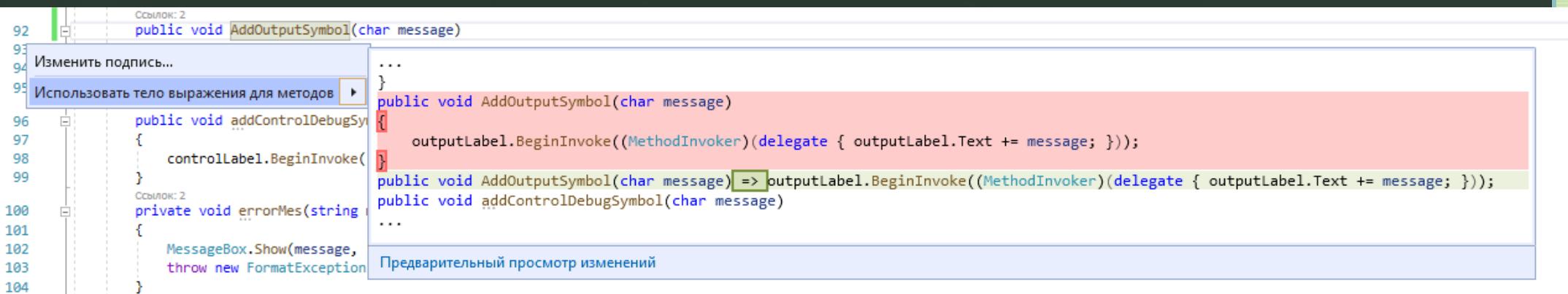
The screenshot shows a code editor in Visual Studio with the following code:

```
82     {
83         TextBox inputTextBox = (TextBox)sender;
84         string input = inputTextBox.Text;
85         foreach (char a in input)
86             sendChar(a);
87             addOutputSymbol('\n');
88             addLine();
89             clearInput();
90
91         public void NewMethod(object sender)
92         {
93             output();
94         }
95         public void cont()
96         {
97             continue();
98         }
99         private void Message()
100         {
101             MessageBox.Show("Error");
102             throw new Exception("Error");
103         }
104         private void clearInput()
105         {
106             ...
107         }
108
109         public void addOutputSymbol(char message)
110         ...
111 }
```

A context menu is open at line 91, with the option "Извлечение метода" (Extract Method) highlighted. A tooltip "Ссылок: 2" is visible near the cursor. The bottom status bar shows "Предварительный просмотр изменений" (Preview changes).

Visual Studio

Если метод состоит из одной строки, то Visual Studio предложит использовать тело выражения для методов, при нажатии ПКМ -> «Быстрые действия и рефакторинг» (Ctrl+.)

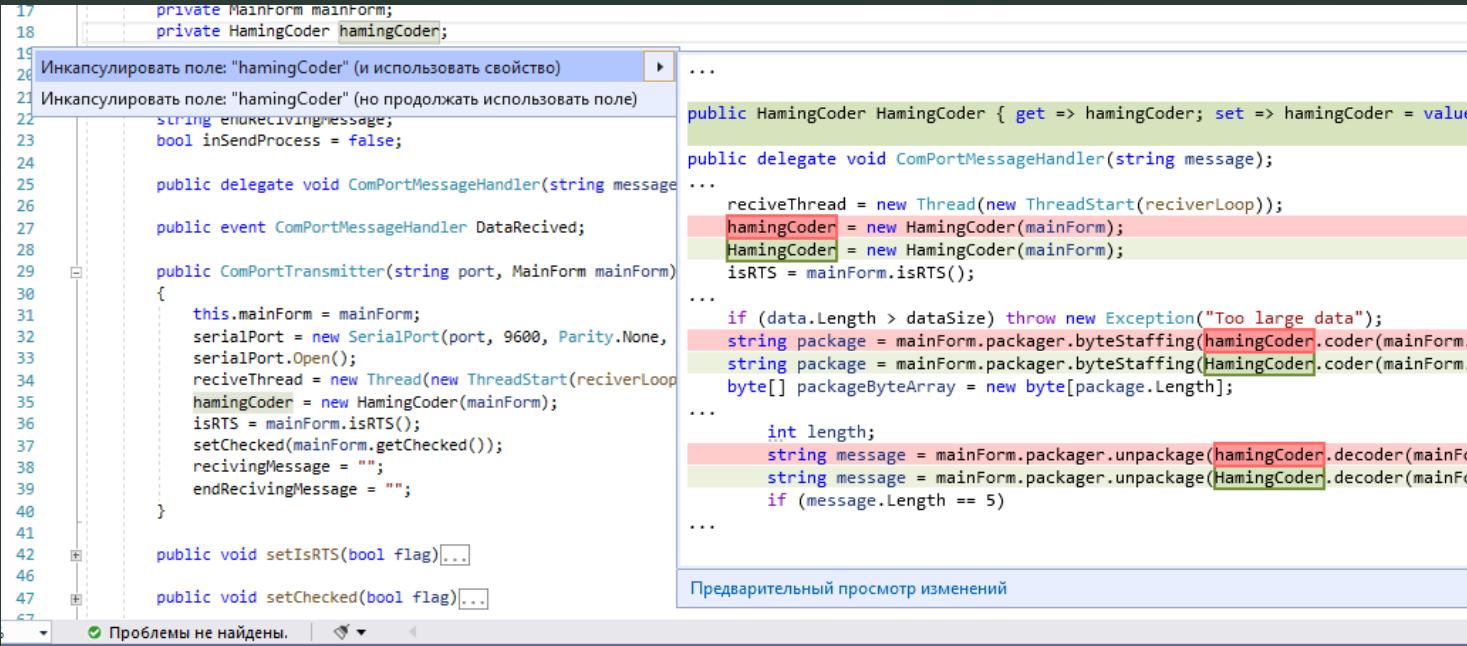


The screenshot shows a code editor in Visual Studio with the following code:

```
Ссылок: 2
92 public void AddOutputSymbol(char message)
93 {
94     // Использовать тело выражения для методов
95     public void AddOutputSymbol(char message)
96     {
97         ...
98         controlLabel.BeginInvoke(
99             () => outputLabel.Text += message);
100    }
101    private void errorMes(string message)
102    {
103        MessageBox.Show(message,
104        throw new FormatException();
105    }
}
```

A context menu is open at line 95, with the option "Использовать тело выражения для методов" (Use expression body for methods) highlighted. A tooltip "Предварительный просмотр изменений" (Preview changes) is visible at the bottom of the menu.

Visual Studio



The screenshot shows a Visual Studio code editor with a context menu open over a variable declaration. The variable is highlighted in red, indicating it is selected for refactoring. The context menu has two items: 'Инкапсулировать поле: "hamingCoder" (и использовать свойство)' (Encapsulate field: "hamingCoder" (use property)) and 'Инкапсулировать поле: "hamingCoder" (но продолжать использовать поле)' (Encapsulate field: "hamingCoder" (keep using field)). The main code area shows a class definition for a ComPortTransmitter. The variable 'hamingCoder' is used multiple times throughout the code, with some instances highlighted in green and others in red, corresponding to the refactoring options.

```
17     private MainForm mainForm;
18     private HamingCoder hamingCoder;
19
19 Инкапсулировать поле: "hamingCoder" (и использовать свойство)
20 Инкапсулировать поле: "hamingCoder" (но продолжать использовать поле)
22         string endReceivingMessage;
23         bool inSendProcess = false;
24
25         public delegate void ComPortMessageHandler(string message);
26
27         public event ComPortMessageHandler DataReceived;
28
29         public ComPortTransmitter(string port, MainForm mainForm)
30     {
31             this.mainForm = mainForm;
32             serialPort = new SerialPort(port, 9600, Parity.None,
33             serialPort.Open();
34             receiveThread = new Thread(new ThreadStart(reciverLoop));
35             hamingCoder = new HamingCoder(mainForm);
36             isRTS = mainForm.isRTS();
37             setChecked(mainForm.getChecked());
38             receivingMessage = "";
39             endReceivingMessage = "";
40         }
41
42         public void setIsRTS(bool flag){...}
43
44         public void setChecked(bool flag){...}
45
46
47
...
public HamingCoder HamingCoder { get => hamingCoder; set => hamingCoder = value; }

public delegate void ComPortMessageHandler(string message);

...
receiveThread = new Thread(new ThreadStart(reciverLoop));
hamingCoder = new HamingCoder(mainForm);
HamingCoder = new HamingCoder(mainForm);
isRTS = mainForm.isRTS();

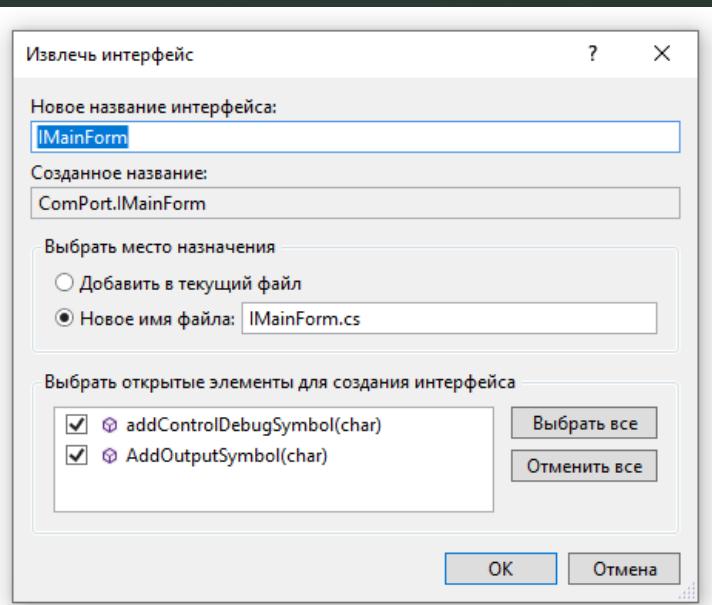
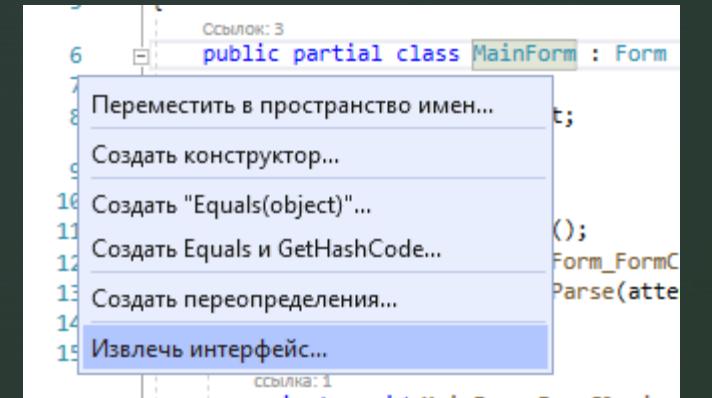
...
if (data.Length > dataSize) throw new Exception("Too large data");
string package = mainForm.packager.byteStaffing(hamingCoder.coder(mainForm.p
string package = mainForm.packager.byteStaffing(HamingCoder.coder(mainForm.p
byte[] packageByteArray = new byte[package.Length];

...
int length;
string message = mainForm.packager.unpackage(hamingCoder.decoder(mainFor
string message = mainForm.packager.unpackage(HamingCoder.decoder(mainFor
if (message.Length == 5)

...
Предварительный просмотр изменений
```

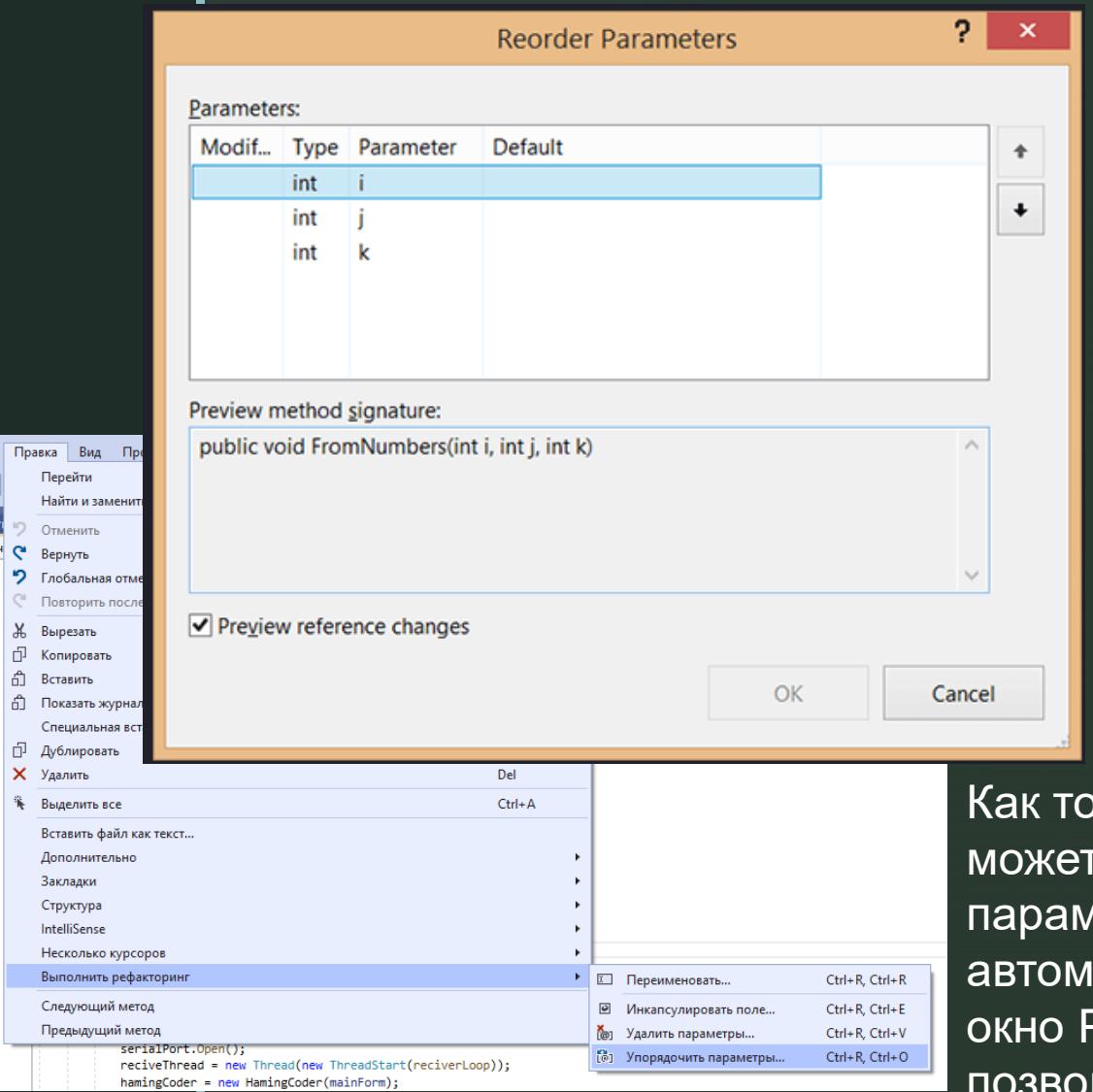
При рефакторинге часто требуется инкапсулировать в существующий класс некое свойство. Для этого предназначена *операция Инкапсулировать поле*. Чтобы выполнить это действие, следует выбрать переменную, которую требуется инкапсулировать, и выбрать соответствующую команду в контекстном меню. Это дает разработчику возможность указать свойство и выбрать место, в котором следует искать ссылку на него.

Visual Studio



По мере того как проект проходит этапы развития от прототипа на ранней стадии разработки до полной реализации или стадии роста, часто возникает необходимость выделить основные методы в виде интерфейса, доступного другим приложениям, или определить границы между непересекающимися системами. Прежде для этого приходилось копировать весь метод в новый файл и удалять его содержимое, оставляя лишь заглушку интерфейса. *Операция рефакторинга Извлечь интерфейс* позволяет извлечь интерфейс, используя любое количество методов в классе. При выполнении этой операции открывается диалоговое окно, показанное на рисунке ниже, которое позволяет пользователю выбрать метод, который он хочет включить в интерфейс. После того как разработчик сделает выбор, эти методы добавляются в новый интерфейс. Кроме того, новый интерфейс добавляется в исходный класс.

Visual Studio

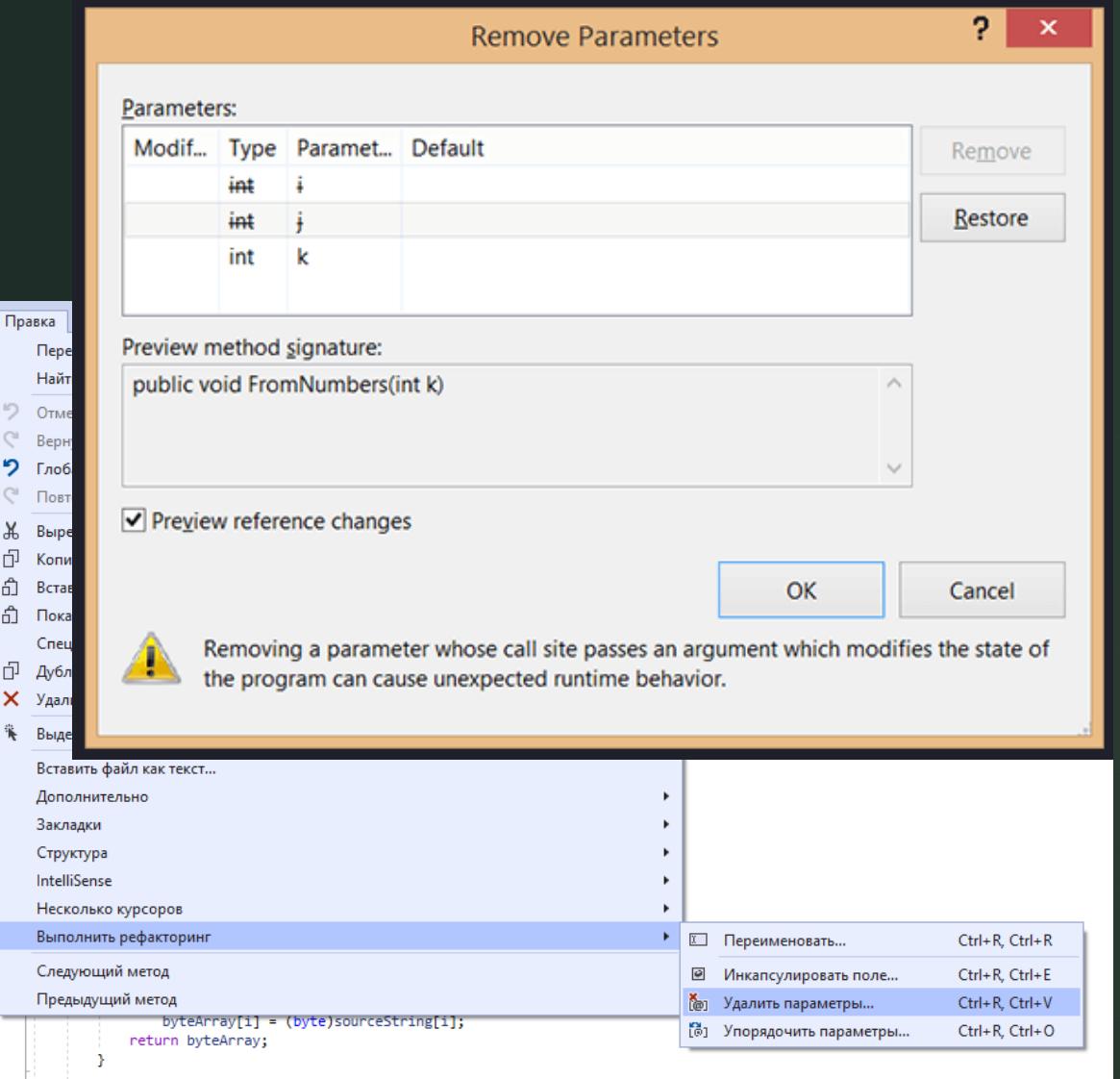


Иногда требуется переупорядочить параметры. Часто это делают по "косметическим" причинам, но эта операция может помочь повысить читабельность и иногда является необходимой при реализации интерфейсов.

Диалоговое окно Reorder Parameters, показанное на рисунке выше, вызывается с помощью «Правка -> Выполнить рефакторинг -> Упорядочить параметры» и позволяет переставлять параметры в списке в соответствии с требуемым порядком.

Как только требуемый порядок будет достигнут, пользователь может предварительно просмотреть результат. По умолчанию параметры в каждом вызове данного метода переставляются автоматически в соответствии с новым порядком. Диалоговое окно Preview, аналогичное показанному на рисунке, позволяет контролировать обновление вызовов метода.

Visual Studio



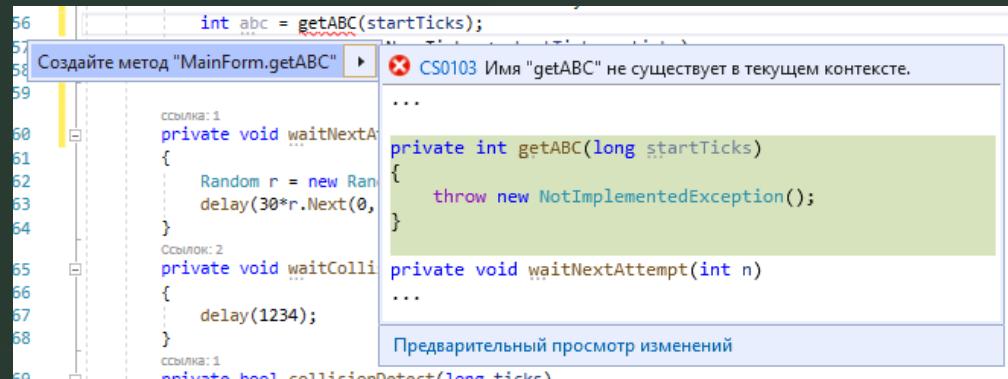
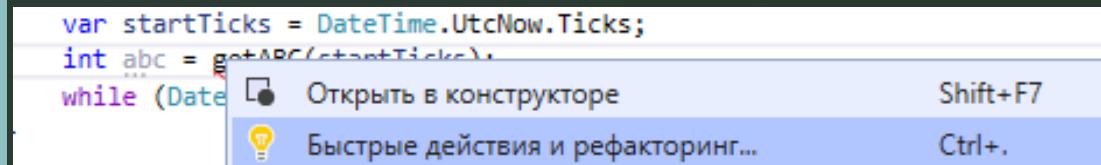
Удаление параметра из метода с помощью соответствующей операции рефакторинга значительно сокращает объем поиска ошибок компиляции, которые могут при этом возникнуть. Кроме того, эта операция очень полезна при многочисленных перегрузках метода, при которых удаление параметра не может порождать ошибки компиляции. В этом случае ошибка во время выполнения программы может возникнуть только по семантической, а не по синтаксической причине.

На рисунке выше показано диалоговое окно Remove Parameters, которое используется для удаления параметров из списка параметров. Если параметр был удален непреднамеренно, его легко восстановить. Как указывает предупреждение, размещенное в этом диалоговом окне, удаление параметров часто приводит к неожиданным функциональным ошибкам, поэтому важно контролировать внесенные изменения. Для того чтобы оценить внесенные изменения, можно снова использовать окно предварительного просмотра.

Visual Studio

Когда разработчик пишет код, он может прийти к выводу, что ему нужен вызов метода, который еще не написан. Например, следующий снippet иллюстрирует новый метод, который должен быть сгенерирован на более поздней стадии:

```
var startTicks = DateTime.UtcNow.Ticks;
int abc = getABC(startTicks);
```



Разумеется, этот код порождает ошибку при сборке, поскольку вызываемый метод еще не определен. Используя операцию рефакторинга Создать метод-заглушку (доступную с помощью интеллектуального указателя в самом коде), можно сгенерировать заглушку метода. Как видно из следующего примера, заглушка метода содержит входные параметры и тип возвращаемого значения:

```
ссылка: 1
private int getABC(long startTicks)
{
    throw new NotImplementedException();
```

Спасибо за
внимание!