# Change Method Signature in Java

In Java, you can use the Change Method Signature refactoring to:

- Change the method name, return type and visibility scope.
- Add new parameters and remove the existing ones. Note that you can also add a parameter using a dedicated Extract Parameter refactoring.
- Reorder parameters.
- Change parameter names and types.
- Add and remove exceptions.
- Propagate new parameters and exceptions through the method call hierarchy.

On this page:

- Examples
- Changing a method signature

## Examples

The following table shows 4 different ways of performing the same Change Method Signature refactoring.

In all the cases, a new parameter `price` of the type `double` is added to the method `myMethod()`.

The examples show how the method call, the calling method (`myMethodCall()`) and other code fragments may be affected depending on the refactoring settings.

| Before | |
|---|---|
| `public class MyClass {`<br><br>`    // This is the method whose signature will be changed:`<br><br>`    public void myMethod(int value) {`<br>`        // some code here`<br>`    }`<br>`}`<br><br>`public class MyOtherClass {`<br>`    public void myMethodCall(MyClass myClass) {`<br>`        double d=0.5;`<br><br>`        // Here is the method call:`<br><br>`        myClass.myMethod(1);`<br>`    }`<br>`}`<br><br>`// We'll ask IntelliJ IDEA to update all the method calls.`<br>`// We'll also specify a default value to be passed to the method.` | `public class My`<br><br>`    // The new parameter pric`<br><br>`    public void myMethod(int`<br>`        // some code here`<br>`    }`<br>`}`<br><br>`public class MyOtherClass {`<br>`    public void myMethodCall(`<br>`        double d=0.5;`<br><br>`        // The method call ha`<br><br>`        myClass.myMethod(1, 0`<br>`    }`<br>`}`<br><br>`// When performing the refact`<br>`// the default parameter valu` |

**Before**

```
                public class MyClass {

    // This is the method whose signature will be changed:

    public void myMethod(int value) {
        // some code here
    }
}

public class MyOtherClass {
    public void myMethodCall(MyClass myClass) {
        double d=0.5;

        // Here is the method call:

        myClass.myMethod(1);
    }
}

// We'll ask IntelliJ IDEA to update all the method calls.
// We'll also ask IntelliJ IDEA to look for a variable
// of the appropriate type near the method call and pass this
// variable to the method.
// In IntelliJ IDEA, this option is called Use Any Var.
```

```
                public class My

    // The new parameter pric

    public void myMethod(int
        // some code here
    }
}

public class MyOtherClass {
    public void myMethodCall(
        double d=0.5;

        // The method call ha

        myClass.myMethod(1, d
    }
}

// Near the method call, Inte
// which has the same type as
// this variable was used in
```

```
                public class MyClass {




    // This is the method whose signature will be changed:

    public void myMethod(int value) {
        // some code here
    }
}

public class MyOtherClass {
    public void myMethodCall(MyClass myClass) {
        double d=0.5;

        // Here is the method call:

        myClass.myMethod(1);
    }
}

// We'll ask IntelliJ IDEA to keep the method calls unchanged but
// create a new overloading method which will call the method
// with the new signature.
// In IntelliJ IDEA, this way of handling the method calls is
// referred to as Delegate via overloading method.
```

```
                public class My

    // A new overloading metho

    public void myMethod(int
                myMethod(i, 0.
                }

    // The new parameter pric

    public void myMethod(int
        // some code here
    }
}

public class MyOtherClass {
    public void myMethodCall(
        double d=0.5;

        // The method call ha

        myClass.myMethod(1);
    }
}

// Note that the new overload
// However, it calls the metho
// 0.0 was specified as the d
// when performing the refact
```

| Before | |
|---|---|
| <pre>               public class MyClass {<br><br>    // This is the method whose signature will be changed:<br><br>    public void myMethod(int value) {<br>        // some code here<br>    }<br>}<br><br>public class MyOtherClass {<br><br>    // This method will also change its signature:<br><br><br>    public void myMethodCall(MyClass myClass) {<br>        double d=0.5;<br><br>        // Here is the method call:<br><br>        myClass.myMethod(1);<br>    }<br>}<br><br>// This time we'll ask IntelliJ IDEA to propagate the new<br>// parameter to the method call through the calling method<br>// myMethodCall().</pre> | <pre>               public class My<br><br>    // The new parameter price<br><br>    public void myMethod(int :<br>        // some code here<br>    }<br>}<br><br>public class MyOtherClass {<br><br>    // The new parameter price<br>    // to the method call thro<br><br>    public void myMethodCall(M<br>        double d=0.5;<br><br>        // The method call has<br><br>        myClass.myMethod(1, pr<br>    }<br>}</pre> |

### Changing a method signature

1. In the editor, place the cursor within the name of the method whose signature you want to change.
2. Do one of the following:
   - Press `Ctrl+F6`.
   - Choose **Refactor | Change Signature** in the main menu.
   - Select **Refactor | Change Signature** from the context menu.

   > If you refactor a method that overrides another method, IntelliJ IDEA suggests either to modify the method from the base class, or to modify only the selected method.

3. In the Change Signature dialog, make the necessary changes to the method signature and specify which other, related changes are required.

   You can:

   - Change the method visibility scope (access level modifier) by selecting the necessary option under **Visibility**.
   - Change the method return type by editing the contents of the **Return type** field.

     > Code completion is available in this field, and also in other fields used for specifying the types.

   - Change the method name. To do that, edit the text in the **Name** field.

- Manage the method parameters using the controls on the **Parameters** tab:
  - To add a new parameter, click + (`Alt+Insert`) and specify the properties of the new parameter in the corresponding fields. If necessary, select the Use Any Var option.

    When adding parameters, you may want to propagate these parameters to the methods that call the current method.
  - To remove a parameter, select this parameter in the table and click — (`Alt+Delete`).
  - To reorder the parameters, use ↑ (`Alt+Up`) and ↓ (`Alt+Down`).
  - To change the name, type, or the default value for a parameter, click this parameter in the table and make the necessary edits in the corresponding fields.
- Propagate new method parameters (if any) along the hierarchy of the methods that call the current method.

  (There may be the methods that call the method whose signature you are changing. These methods, in their turn, may be called by other methods, and so on. You can propagate the changes you are making to the parameters of the current method through the hierarchy of the calling methods and also specify which calling methods should be affected and which shouldn't.)

  To propagate the new parameters:

  1. Click (`Alt+G`).
  2. In the left-hand pane of the **Select Methods to Propagate New Parameters** dialog, expand the necessary nodes and select the check boxes next to the methods you want the new parameters to be propagated to.

     To help you select the necessary methods, the code for the calling method and the method being called is shown in the right-hand part of the dialog (in the **Caller Method** and **Callee Method** panes respectively).

     As you switch between the methods in the left-hand pane, the code in the right-hand part changes accordingly.
  3. Click **OK**.
- Manage the method exceptions using the list of exception types and the buttons on the **Exceptions** tab. The procedures are similar to those used for managing the method parameters.
- Propagate new exceptions (if any) along the hierarchy of the methods that call the current method. To initiate this procedure, use (`Alt+X`). In all other respects, the procedure is similar to that used for propagating new method parameters.
- Specify how the method calls should be handled. To do that, select one of the following **Method calls** options:
  - If you want the method calls to be modified, select **Modify**.
  - If you want to leave the existing method calls unchanged, select **Delegate via overloading method**.

4. To perform the refactoring right away, click **Refactor**.

   To see the expected changes and make the necessary adjustments prior to actually performing the refactoring, click **Preview**.

### See Also

Reference:

- Change Signature Dialog for Java

Web Resources:

- Developer Community