JET BRAINS

Multi-steps Refactorings in IntelliJ IDEA

Anna Kozlova
anna.kozlova@jetbrains.com

Refactoring - is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.

Martin Fowler https://refactoring.com/catalog/

- Refactor This... ⌥⇧⌘T
- Rename... ⇧F6
- Rename File...
- Change Signature... ⌘F6
- Edit Property Value... ⌥F6
- Type Migration... ⇧⌘F6
- Make Static...
- Convert To Instance Method...

- Move Members... F6
- Copy Class... F5
- Safe Delete... ⌘⌦

- Extract/Introduce >
  - Variable... ⌥⌘V
  - Constant... ⌥⌘C
  - Field... ⌥⌘F
  - Parameter... ⌥⌘P

  - Functional Parameter... ⌥⇧⌘P
  - Functional Variable...
  - Parameter Object...

  - Method... ⌥⌘M
  - Replace Method With Method Object...

  - Delegate...
  - Extract Class...
  - Interface...
  - Introduce Type...
  - Superclass...
  - Subquery as CTE
  - RSpec 'let'
- Inline Method... ⌥⌘N
- Find and Replace Code Duplicates...
- Invert Boolean...

- Pull Members Up...
- Push Members Down...
- Push ITDs In...
- Use Interface Where Possible...
- Replace Inheritance with Delegation...
- Remove Middleman...
- Wrap Method Return Value...

- Encapsulate Fields...
- Replace Temp with Query...
- Replace Constructor with Factory Method...
- Replace Constructor with Builder...

- Generify...
- Migrate...

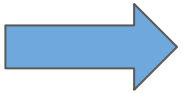
- Lombok >
- Delombok >
- Internationalize...

- Part I. Existing Refactorings
  - Replace temp with query
  - Replace constructor with factory method
  - Remove middle man
- Part II. New combinations
  - Extract method combinations
  - Merge 2 interfaces

Part I. Existing Refactorings

Replace Temp with Query

```
int avg = (a + b) / 2;

doSmth(avg);
```



```
doSmth(avg(a, b));

int avg(int a, int b) {
    return (a + b) / 2;
}
```

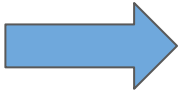
```
    int avg = (a + b) / 2;
    doSmt
}

 @Contract(
private s

}
```

Refactor This

1	Rename...	⇧F6
2	Copy Class...	F5
3	Safe Delete...	⌘⌫
	Extract/Introduce	
4	Introduce Constant...	⌥⌘C
5	Introduce Field...	⌥⌘F
6	Introduce Parameter...	⌥⌘P
7	Introduce Functional Parameter...	⌥⇧⌘P
8	Type Parameter...	
9	Inline Variable	⌥⌘N
0	Replace Temp with Query...	

Replace Temp with Query. Side Effects

```
int r = sendResult();

doSmth(r);

doSmth(r);
```

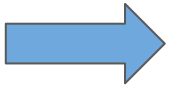


```
doSmth(sendResult());
doSmth(sendResult());
```

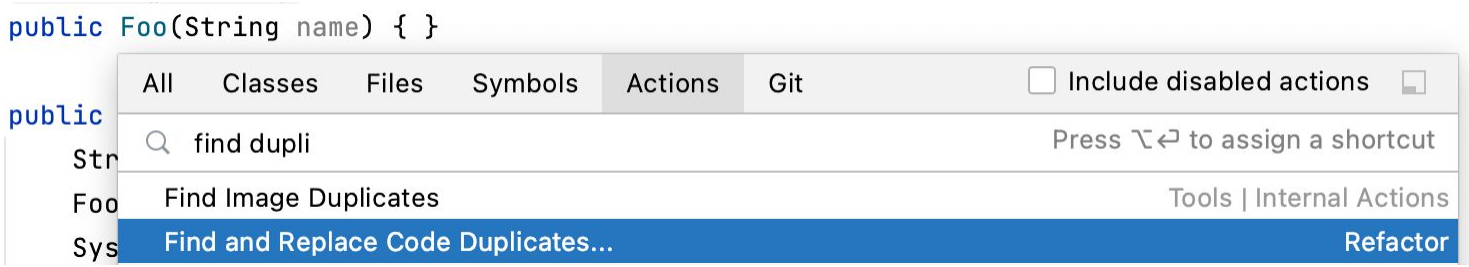
?

Replace Constructor with Factory Method

```
Foo(String name) { }

Foo foo = new Foo("name");
```



```
Foo(String name) { }

Foo foo = createFoo("name");
```



Remove middle man

```
class Person {

   private Department department;

   public Manager getManager() {

       return department.getManager();

   }

}

Manager manager = person.getManager();
```

```
 Department department;
Manager getMan
urn department



ss Department
 Manager manag

ct(pure = true)
Manager getMan
urn manager;
```

Refactor This

1 Rename... ⇧F6

2 Move Members... F6

3 Copy Class... F5

4 Safe Delete... ⌘⌦

Extract/Introduce

5 Extract Delegate...

6 Extract Interface...

7 Extract Superclass...

8 Inline Field... ⌥⌘N

9 Remove Middleman...

0 Encapsulate Fields...

Encapsulate Fields - removemiddleman.Demo.Person

Fields to Encapsulate

	Field	Getter	Setter
☑	f 🔒 department:Departm...	getDepartment	setDepartment

Encapsulate

- ☑ Get access
- ☑ Set access
- ☑ Use accessors even when field is accessible

Encapsulated Fields' Visibility

- ◉ Private
- ○ Package local
- ○ Protected
- ○ As is

Accessors' Visibility

- ◉ Public
- ○ Protected
- ○ Package local
- ○ Private

? Cancel Preview Refactor

Remove middle man. Encapsulate field

```
class Person {
   private Department department;
   public Department getDepartment() {return department;}
   public Manager getManager() {
       return getDepartment().getManager();
   }
}
Manager manager = person.getManager();
```

Remove middle man. Inline

```
class Person {

   private Department department;

   public Department getDepartment() {return department;}

}

Manager manager = person.getDepartment().getManager();
```

Extract delegate

```
class Person {
   int officeNumber; int officeAreaCode;
   public Person(int officeNumber, int officeAreaCode) {
       this.officeNumber = officeNumber;
       this.officeAreaCode = officeAreaCode;
   }
}
```

Part II. New Combinations

Replace multiple expressions

```
public T getFirstElement(List<T> elements) {
    if (elements.size() == 0) return null;
    return elements.get(0);
}
public T removeFirstElement(List<T> elements) {
    if (elements.size() == 0) return null;
    return elements.remove(0);
}
```

Replace multiple expressions. How?

1. Search and Replace
2. Multiple carets
3. Structural Search
4. Refactoring

Search and replace (Ctrl/Cmd + R)

elements.size() == 0

elements.isEmpty()

2 results

Replace | Replace All | Exclude

```
1   package extractMethod;
2   import java.util.List;
3   public class Inline<T> {
4       public T getFirstElement( @NotNull List<T> elements) {
5           if (elements.size() == 0) return null;
6           return elements.get(0);
7       }
8       public T removeFirstElement( @NotNull List<T> elements) {
9           if (elements.size() == 0) return null;
10          return elements.remove( index: 0);
```

Multiple carets (Ctrl/Cmd + F)

```
elements.size() == 0
1/2
1  package extractMethod;
2  import java.util.List;
```

Select All Occurrences ^⌘G

```
public T getFirstElement(List<T> elements) {
    if (el) return null;
    return elements.get(0);
}
public T removeFirstElement(List<T> elements) {
    if (el) return null;
    r
}
```

p elements	List<T>
m elements.isEmpty()	boolean

Structural Replace

Structural Replace

Search template:

Search in injected code Match case File type: Java

```
$elements$.size() == 0
```

No filters added for the whole template.
Add filter

Replace template:

Shorten fully qualified names Use static imports Reformat

```
$elements$.isEmpty()
```

In Project Module Directory Scope Current File ...

Replace multiple expressions, inexact match

```
public T getFirstElement(List<T> children) {
    if (children.size() == 0) return null;
    return children.get(0);
}

public T removeFirstElement(List<T> elements) {
    if (elements.size() == 0) return null;
    return elements.remove(0);
}
```

Replace multiple expressions: extract method (Ctrl+Alt+M)

```
public T getFirstElement(List<T> children) {
    if (children.size() == 0) return null;
    return children.get(0);
}

public T removeFirstElement(List<T> elements) {
    if (elements.size() == 0) return null;
    return elements.remove(0);
}
```

Replace multiple expressions: change

```
public T getFirstElement(List<T> children) {
    if (isEmpty(children)) return null;
    return children.get(0);
}
                    ...

private boolean isEmpty(List<T> elements) {
    return elements.isEmpty();      elements.size() == 0;
}
```

Replace multiple expressions: inline (Ctrl+Alt+N)

```
public T getFirstElement(List<T> children) {
    if (children.isEmpty()) return null;
    return children.get(0);
}
public T removeFirstElement(List<T> elements) {
    if (elements.isEmpty()) return null;
    return elements.remove(0);
}
```

Replace multiple expressions. Plan

Extract method: predict method signature

```
int numStrings = arrayOfStrings.length;
System.out.println("Processing " + numStrings + " strings");
StringBuilder sb = new StringBuilder();
sb.append(numStrings).append(":");
for (String string : arrayOfStrings) {
    doSmth(string, sb);
}
System.out.println("Strings:");
System.out.println(sb.toString());
```

```
StringBuilder
processStrings(String[] array,
               int length)
```

Extract method: predict method signature

```
System.out.println("Processing " + arrayOfStrings.length + " strings");
```

```
StringBuilder sb = new StringBuilder();

sb.append(arrayOfStrings.length).append(":");

for (String string : arrayOfStrings) {

    doSmth(string, sb);

}
```

```
System.out.println("Strings:");

System.out.println(sb.toString());
```

Extract method: predict method signature

```
System.out.println("Processing " + arrayOfStrings.length + " strings");
```

```
StringBuilder sb = new StringBuilder();

sb.append(arrayOfStrings.length).append(":");

for (String string : arrayOfStrings) {

    doSmth(string, sb);

}

String result = sb.toString();
```

```
System.out.println("Strings:");

System.out.println(result);
```

Extract method: predict method signature

```
private String processStrings(String[] arrayOfStrings) {
    StringBuilder sb = new StringBuilder();
    sb.append(arrayOfStrings.length).append(":");
    for (String string : arrayOfStrings) {
        doSmth(string, sb);
    }
    return sb.toString();
}
```

Extract method: new parameters

```
int avgA = (10 + 20) / 2;

int avgB = (100 + 200)/ 2;
```

Extract method: new parameters

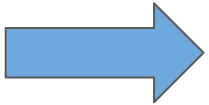
```
int avgA = avg();

int avgB = (100 + 200) / 2;
```

?

```
int avg() {
    return (10 + 20) / 2;
}
```

Extract method: new parameters

```
int a = 10;

int b = 20;

int avgA = (a + b) / 2;

int avgB = (100 + 200) / 2;
```



```
int avg(int a, int b) {
   return (a + b) / 2;
}

int a = 10;

int b = 20;

int avgA = avg(a, b);

int avgB = avg(100, 200);
```

Extract method: new parameters

```
int a = 10;

int b = 20;

int avgA = a + b / 2;

int avgB = 100 + 200 / 2;
```

```
int avg(int a, int b) {
   return (a + b) / 2;
}

int a = 10;

int b = 20;

int avgA = avg(a, b);

int avgB = avg(100, 200);
```

Extract method: introduce parameter

```
int avgA = avg();

int avgB = (100 + 200) / 2;
```

```
int avg() {
    return (10 + 20) / 2;
}
```

Extract method: introduce parameter

```
int avgA = avg(10);

int avgB = (100 + 200)/ 2;
```

```
int avg(int a) {
    return (a + 20)/ 2;
}
```

Extract method: introduce parameter 2

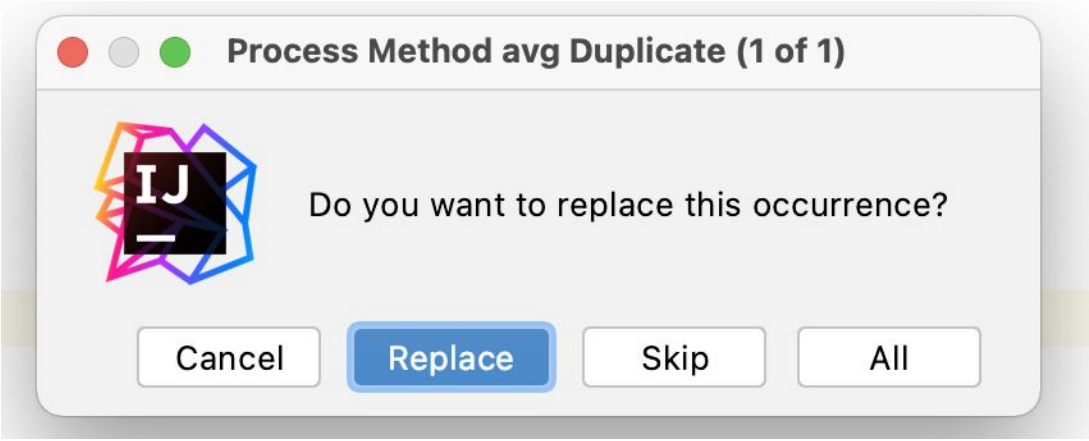
```
int avgA = avg(10, 20);

int avgB =(100 + 200)/ 2;
```

```
int avg(int a, int b) {
    return (a + b) / 2;
}
```

Extract method: replace duplicate

```
int avgA = avg(10, 20);

int avgB =(100 + 200)/ 2;
```

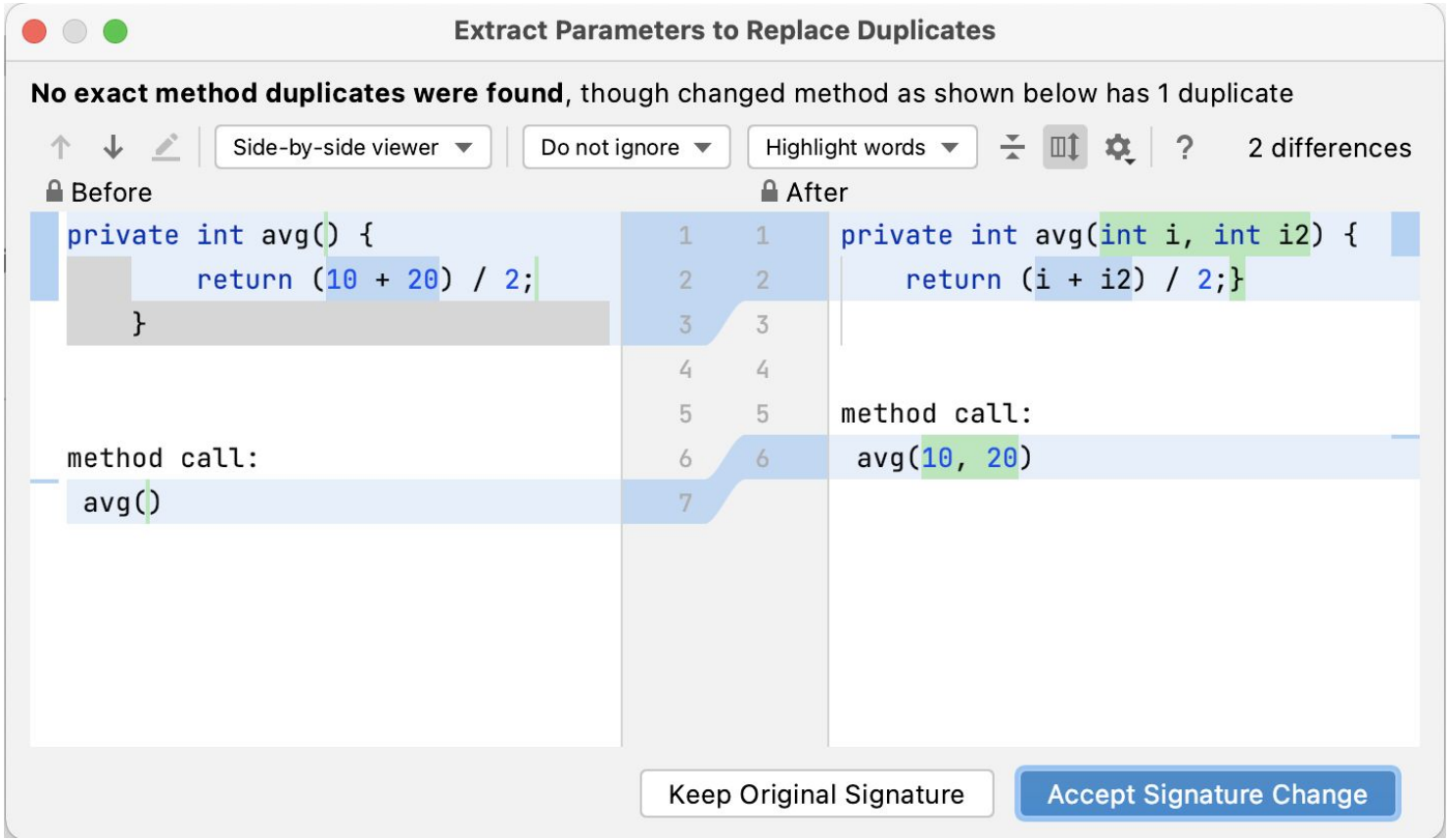
```
int avg(int a, int b) {
    return (a + b) / 2;
}
```



Extract method: replace duplicates

```
int avgA = avg(10, 20);

int avgB = avg(100, 200);
```

```
int avg(int a, int b) {
    return (a + b) / 2;
}
```


Extract Parameters to Replace Duplicates
No exact method duplicates were found, though changed method as shown below has 1 duplicate
Side-by-side viewer
Do not ignore
Highlight words
2 differences
Before
After
private int avg() {
return (10 + 20) / 2;
}
method call:
avg()
private int avg(int i, int i2) {
return (i + i2) / 2;}
method call:
avg(10, 20)
Keep Original Signature
Accept Signature Change

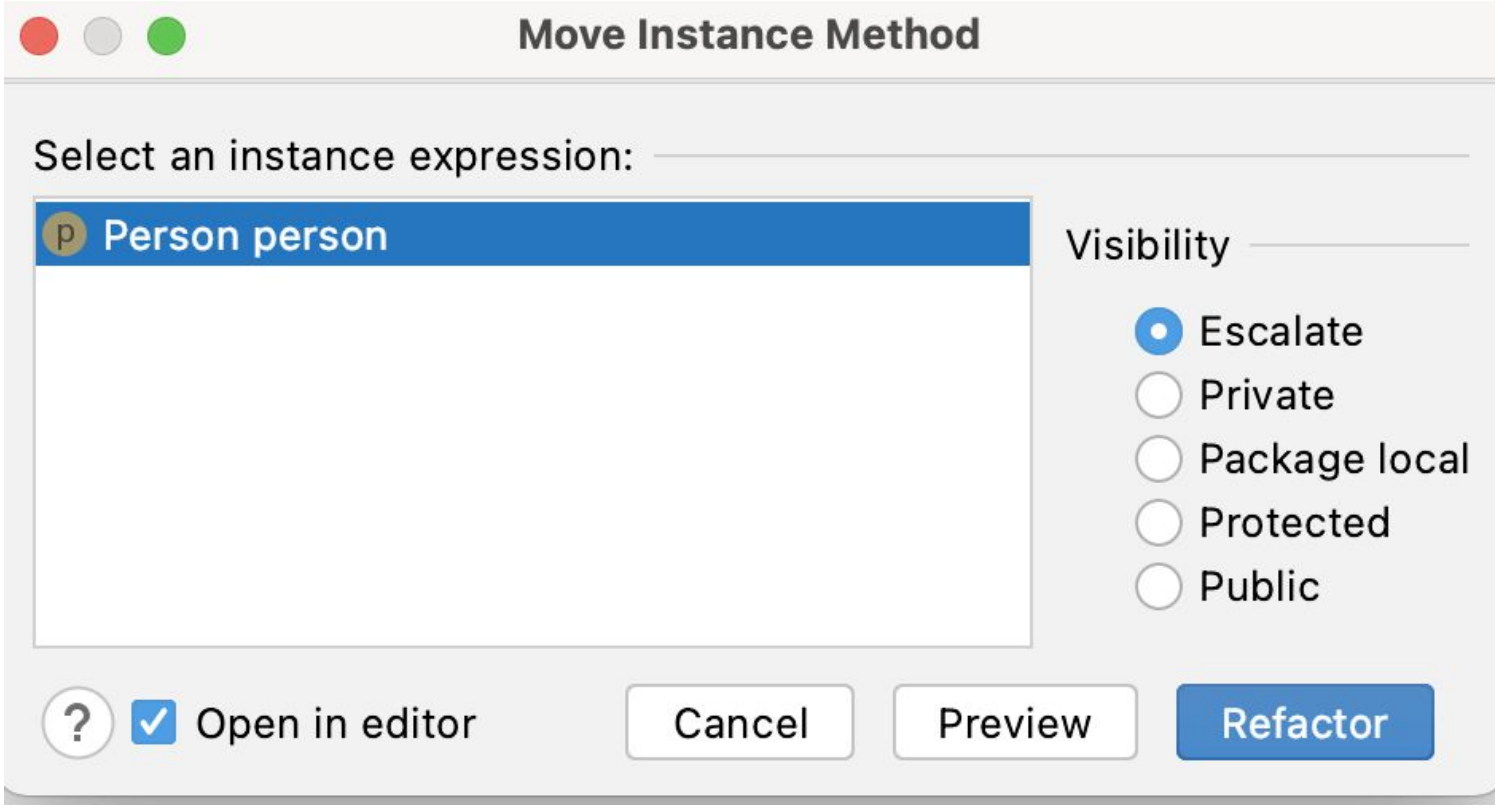
Extract method: change target class

```
record Person(String firstName, String lastName) { }


System.out.println(person.firstName() + " " + person.lastName());
```

Extract method: change target class

```
record Person(String firstName, String lastName) { }

System.out.println(getFullName(person));


private String getFullName(Person person) {

   return person.firstName() + " " + person.lastName();

}
```

Extract method: move instance method



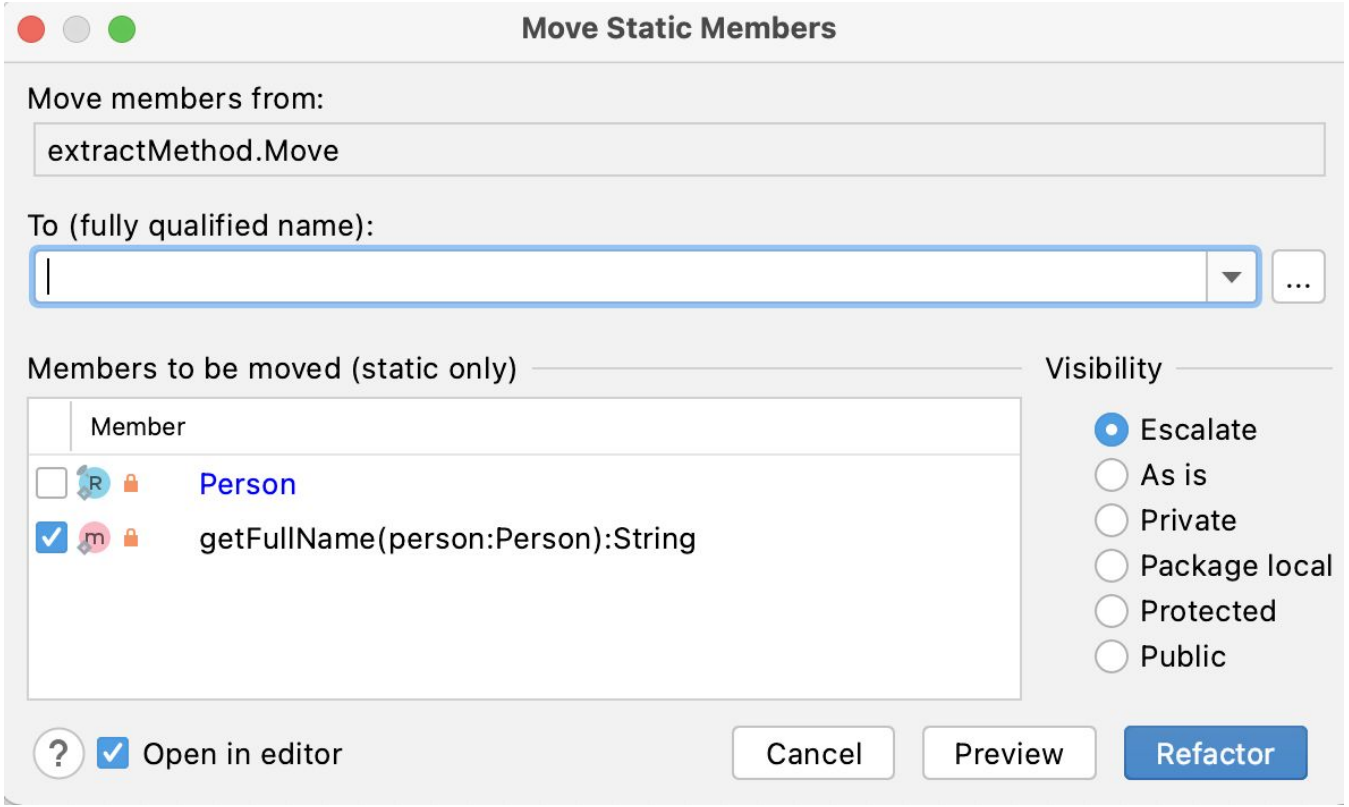
Extract method: change target class

```
record Person(String firstName, String lastName) {
   public String getFullName() {
      return firstName() + " " + lastName();
   }
}
System.out.println(person.getFullName());
```

Extract method: change target class

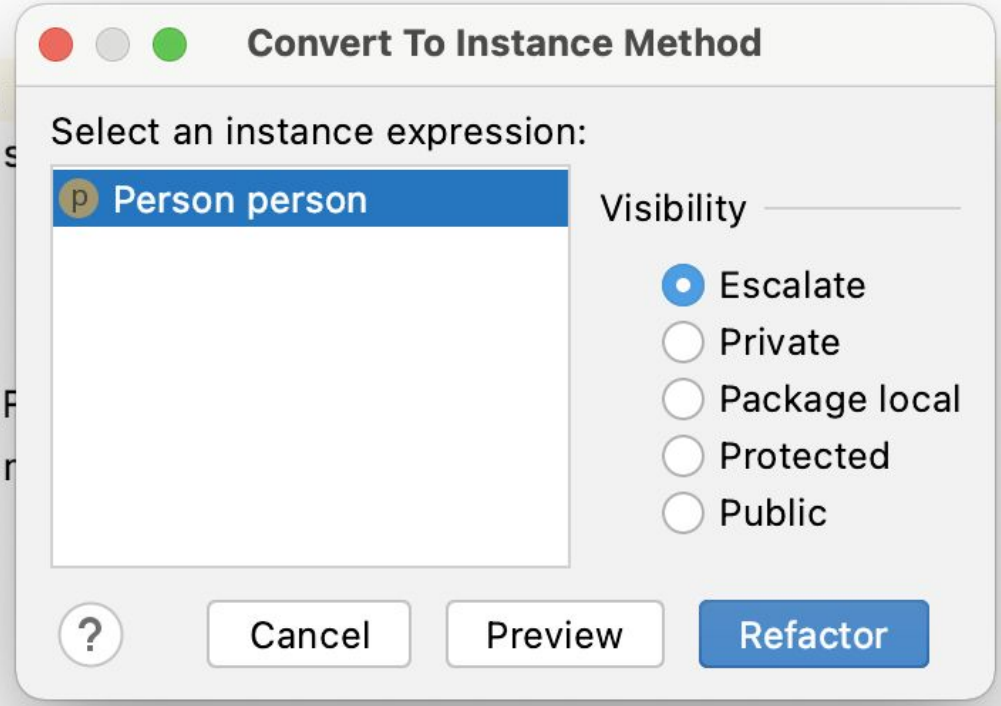
```
record Person(String firstName, String lastName) { }

System.out.println(getFullName(person));


private static String getFullName(Person person) {

    return person.firstName() + " " + person.lastName();

}
```

Extract method: move static method



Convert static method to instance

```
private record Person(String firstName, String lastName) {
    @NotNull
    private static String
        return person.firs
    }
}

void registerNewEmployee(P
    System.out.println(Per
}
```



Extract method: change target class

```
record Person(String firstName, String lastName) {
   public String getFullName() {
      return firstName() + " " + lastName();
   }
}
System.out.println(person.getFullName());
```

Push statements inside method

```
void execute(List<String> commands) { /*do execute*/ }

void start(List<String> commands) {

   checkValid(commands);

   execute(commands);

   //a lot of code here

}
```

Push statements inside method. Extract method

```
void execute(List<String> commands) { /*do execute*/ }

void start(List<String> commands) {
  checkValid(commands);
  execute(commands);
  //a lot of code here
}
```

Push statements inside method

```
void execute(List<String> commands) { /*do execute*/ }

void start(List<String> commands) {

    doSomething(commands);

    //a lot of code here

}
```

```
void doSomething(List<String> commands) {

    checkValid(commands);

    execute(commands);

}
```

Push statements inside method. Inline

```
void execute(List<String> commands) { /*do execute*/ }

void start(List<String> commands) {

    doSomething(commands);

    //a lot of code here

}

void doSomething(List<String> commands) {

    checkValid(commands);

    execute(commands);

}
```

Push statements inside method. Rename

```
void start(List<String> commands) {
    doSomething(commands);
    //a lot of code here
}
void doSomething(List<String> commands) {
    checkValid(commands);
    /*do execute*/
}
```

Push statements inside method

```
void start(List<String> commands) {
    execute(commands);
    //a lot of code here
}
void execute(List<String> commands) {
    checkValid(commands);
    /*do execute*/
}
```

Pull statements outside method

```
void start(List<String> commands) {
    execute(commands);
    //a lot of code here
}
void execute(List<String> commands) {
    checkValid(commands);
    /*do execute*/
}
```

Pull statements outside method. Extract method

```
void start(List<String> commands) {
    execute(commands);
    //a lot of code here
}
void execute(List<String> commands) {
    checkValid(commands);
    /*do execute*/
}
```

Pull statements outside method. Inline

```
void start(List<String> commands) {
    execute(commands);
    //a lot of code here
}
void execute(List<String> commands) {
    checkValid(commands);
    doSomething(commands);
}
void doSomething(List<String> commands) {/*do execute*/}
```

Pull statements outside method. Rename

```
void start(List<String> commands) {
  checkValid(commands);
  doSomething(commands);
   //a lot of code here
}
void doSomething(List<String> commands) {
   /*do execute*/
}
```

Pull statements outside method

```
void start(List<String> commands) {
  checkValid(commands);
  execute(commands);
   //a lot of code here
}
void execute(List<String> commands) {
   /*do execute*/
}
```

Extract Bean

```
String prefix = "before";

String infix  = "middle";

String suffix = "after";
```

```
String merged = prefix + infix + suffix;

System.out.println ("merged = " + merged);
```

Extract Bean. Extract Method

```
String prefix = "before", infix  = "middle", suffix = "after";
String merged = getMerged(prefix, infix, suffix);
System.out.println ("merged = " + merged);

private String getMerged(String prefix, String infix, String suffix) {
    return prefix + infix + suffix;
}
```

Create new class

Name

Package name extractBean ...

Target destination directory:

Leave in same source root ...

Create inner class

Name Bean

Use existing class

Name ...

Escalate visibility

Generate accessors

Parameters to Extract

Type	Name
String	prefix
String	infix
String	suffix

Extract Bean. Introduce Parameter Object

```
String prefix = "before", infix  = "middle", suffix = "after";
String merged = getMerged(new Bean(prefix, infix, suffix));
System.out.println ("merged = " + merged);

private String getMerged(Bean bean) {
    return bean.prefix + bean.infix + bean.suffix;
}
record Bean(String prefix, String infix, String suffix) {}
```

Extract Bean. Inline

```
Bean bean = new Bean("before", "middle", "after");

System.out.println("merged = " +

                   (bean.prefix + bean.infix + bean.suffix));


private record Bean(String prefix, String infix, String suffix) {}
```

Split parameter object

```
record HugeParameterObject(String attr1, String attr2,
                           String attr3, String attr4) { }
public void doSomething(HugeParameterObject payload){
   System.out.println("I only use " + payload.attr1 +
                      " and " + payload.attr2);
}
HugeParameterObject payload =
       new HugeParameterObject("hello", "world", "used?", "used?");
doSomething(payload);
```

Split parameter object. Introduce parameter

```
record HugeParameterObject(String attr1, String attr2,
                           String attr3, String attr4) { }
public void doSomething(HugeParameterObject payload, String attr1){
   System.out.println("I only use " + attr1 + " and " +payload.attr2);
}
HugeParameterObject payload =
    new HugeParameterObject("hello", "world", "used?", "used?");
doSomething(payload, payload.attr1);
```

Split parameter object

```
doSomething (HugeParameterObject payload, String attr1, String attr2)
Delegate via overloading method

record HugeParameter
public void doSomething( @NotNull HugeParameterObject pay ad, String attr1, String attr2){
    System.out.println("I only use " + attr1 + " and " + attr2);
}

{
```

```
attr2
s
Press ⌥⌘P to show dialog with more options
```

```
public void doSomething(String attr1, String attr2)
```

Introduce parameter object with existing bean

```
record Foo(int i, int j) { }

static void doSomething(int i, int j) {

   System.out.println(i + j);

}

final Foo input = new Foo(0, 1);

doSomething(input.i(), input.j());
```

Introduce parameter object. Change signature from call site

```
record Foo(int i, int j) { }

static void doSomething(int i, int j) {

    System.out.println(i + j);

}

final Foo input = new Foo(0, 1);

doSomething(input, input.i(), input.j());
```

Introduce parameter object. Change signature from call site

```
record Foo(int i, int j) { }

static void doSomething(Foo input, int i, int j) {

    System.out.println(i + j);

}

final Foo input = new Foo(0, 1);

doSomething(input, input.i(), input.j());
```

Introduce parameter object. Inline parameters

```
record Foo(int i, int j) { }

static void doSomething(Foo input, int i, int j) {

    System.out.println(i + j);

}

final Foo input = new Foo(0, 1);

doSomething(input, input.i(), input.j());
```

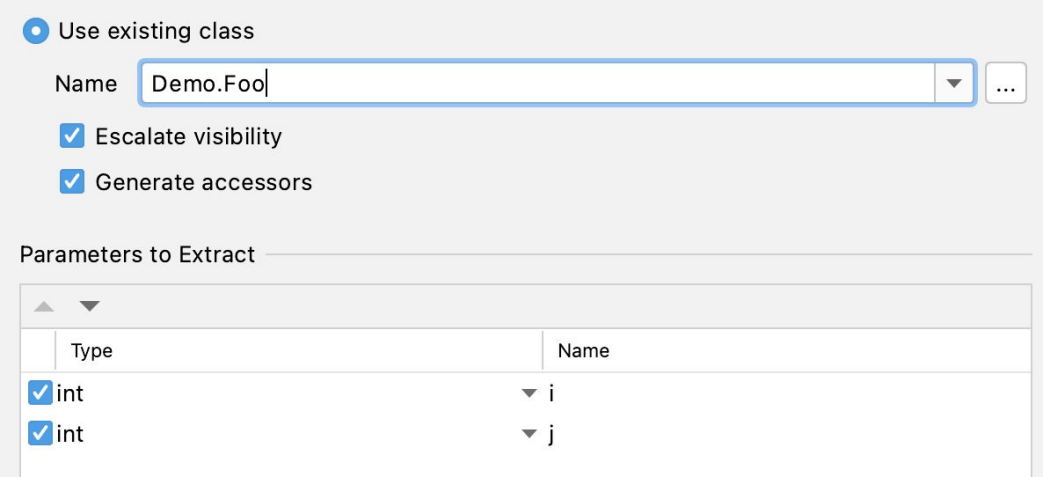
Introduce parameter object with existing bean

```
record Foo(int i, int j) { }

static void doSomething(Foo input) {

    System.out.println(input.i() + input.j());

}

final Foo input = new Foo(0, 1);

doSomething(input);
```

Introduce parameter object. Alternative

```
record Foo(int i, int j) { }

static void doSomething(int i, int j) {

   System.out.println(i + j);

}

final Foo input = new Foo(0, 1);

doSomething(input.i(), input.j());
```

Introduce parameter object




```
final Foo input = new Foo(0, 1);

doSomething(new Foo(input.i(), input.j()));
```

Convert boolean param to enum

```
public void split(boolean vertical) {
    if (vertical) {
        System.out.println("vertical");
    } else {
        System.out.println("horizontal");
    }
}
split(true);
```

Convert boolean param to enum

```
public void split(boolean vertical) {
    if (vertical) { System.out.println("vertical"); }
    else { System.out.println("horizontal"); }
}
split(true);
enum Orientation {VERTICAL, HORIZONTAL}
```

Convert boolean param to enum

```
public void split(boolean vertical) {
    Orientation o = vertical ? Orientation.VERTICAL
                             : Orientation.HORIZONTAL;
    if (vertical) {
        System.out.println("vertical");
    }  else {
        System.out.println("horizontal");
    }
}
```

Convert boolean param to enum

```
public void split(boolean vertical) {
    Orientation o = vertical ? Orientation.VERTICAL
                             : Orientation.HORIZONTAL;
    if (o == Orientation.VERTICAL) {
        System.out.println("vertical");
    }  else {
        System.out.println("horizontal");
    }
}
```

Convert boolean param to enum

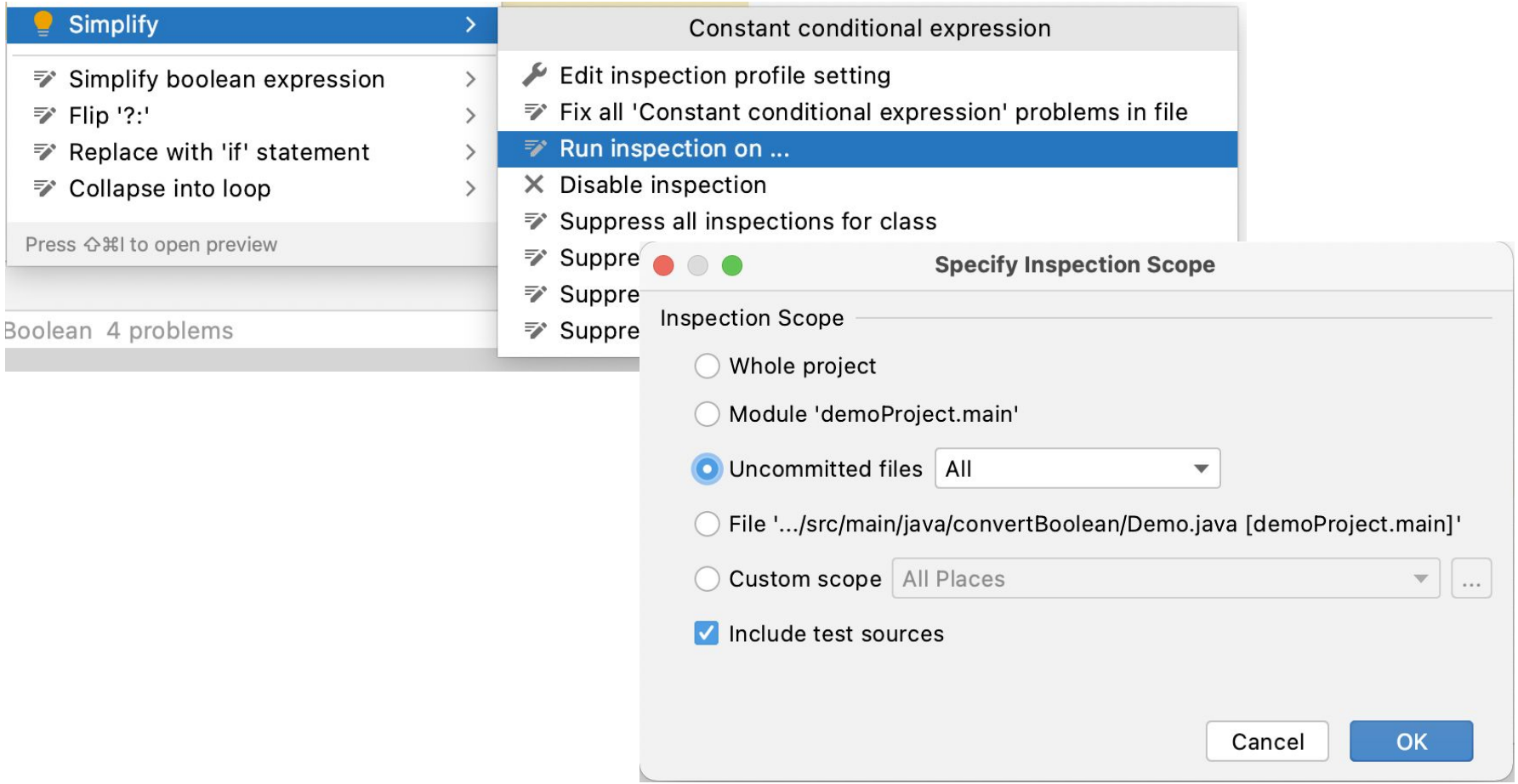
```
public void split(Orientation o) {
    if (o == Orientation.VERTICAL) {
        System.out.println("vertical");
    }  else {
        System.out.println("horizontal");
    }
}
```

```
split(true ? Orientation.VERTICAL : Orientation.HORIZONTAL);
```

Convert boolean param to enum

```
public void split(Orientation o) {
    if (o == Orientation.VERTICAL) {
        System.out.println("vertical");
    }  else {
        System.out.println("horizontal");
    }
}
split(Orientation.VERTICAL);
```

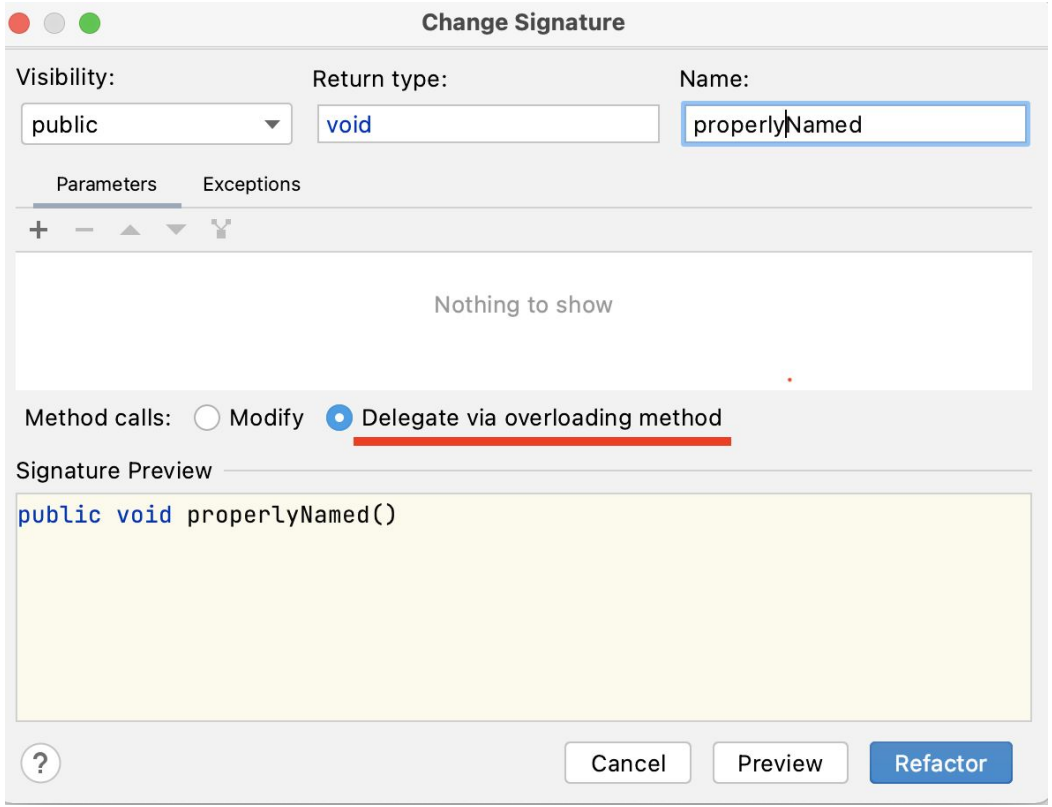
Convert boolean param to enum



Rename api method

```
public void poorlyNamed() {
    //do a lot of stuff
}
```

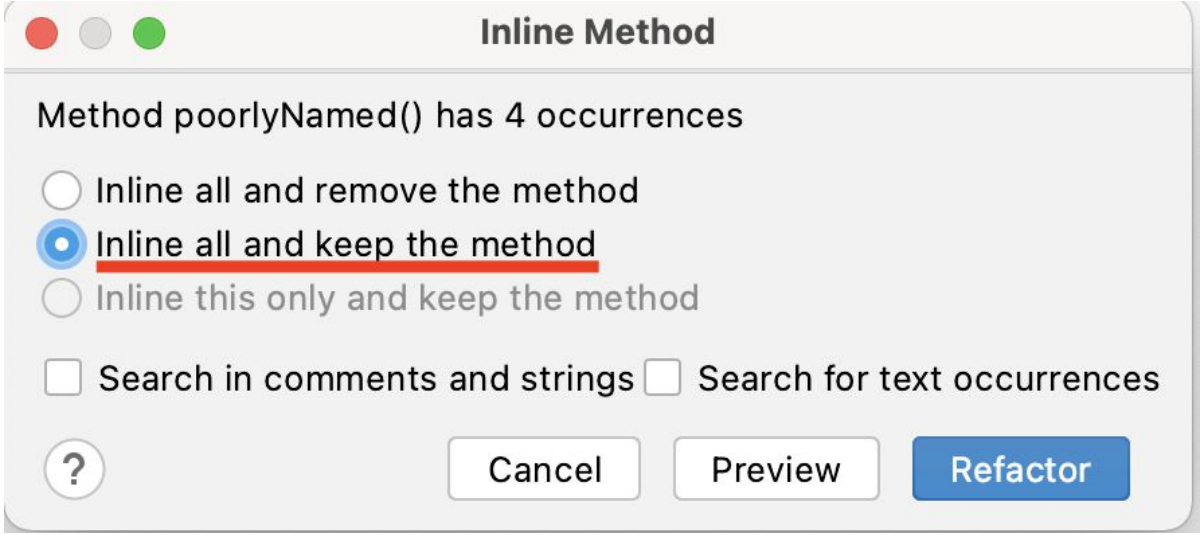
Rename api method. Change Signature



Rename api method. Delegate

```
public void poorlyNamed() {
   properlyNamed();
}

public void properlyNamed() {
   //do a lot of stuff
}
```

Rename api method. Inline



Merge 2 interfaces

```
interface ListFactory<L> { List<L> listOf(L ts); }
interface SetFactory<T> { Set<T> setOf(T ts); }

class ListFactoryImpl<L> implements ListFactory<L> {}
class SetFactoryImpl<S> implements SetFactory<S> {}

<L> List<L> listOf(ListFactory<L> factory, L l)
<S> Set<S> setOf(SetFactory<S> factory, S s)
```

Merge 2 interfaces: plan

1. Pull members up

Merge 2 interfaces: plan

1. Pull members up
2. Use interface where possible

Merge 2 interfaces: plan

1. Pull members up
2. Use interface where possible
3. Safe delete

Merge 2 interfaces: plan

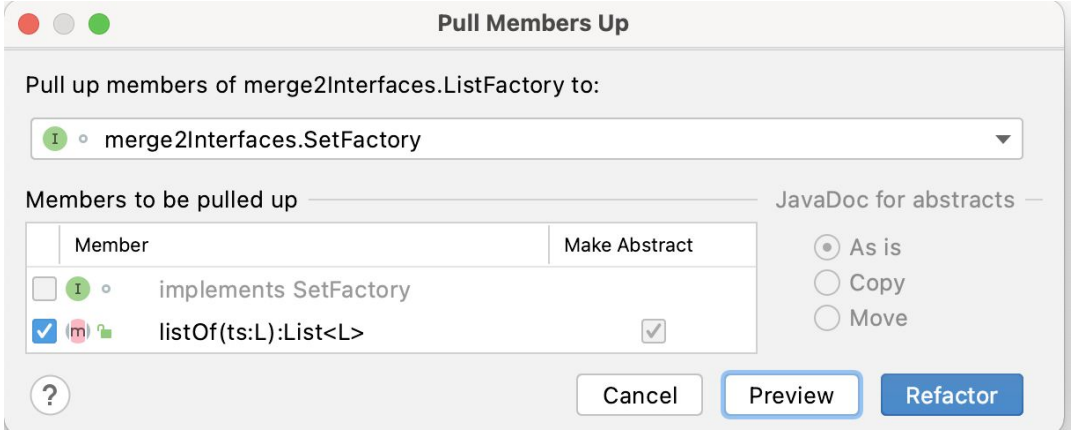
1. Pull members up
2. Use interface where possible
3. Safe delete
4. Rename

Merge 2 interfaces: extend

```
interface ListFactory<L> extends SetFactory<L> {
    List<L> listOf(L ts);
}
interface SetFactory<T> {
    Set<T> setOf(T ts);
}
```

Merge 2 interfaces: pull up

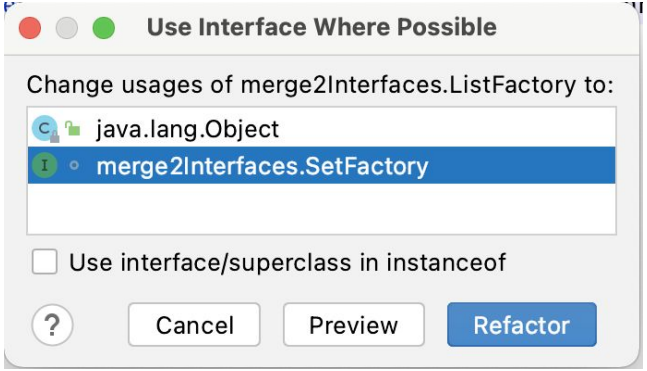
```
interface SetFactory<T> {

    Set<T> setOf(T ts);

    List<L> listOf(L ts);

}
```




```
interface ListFactory<L> extends SetFactory<L> { }
```

Merge 2 interfaces: use interface where possible

```
interface ListFactory<L> extends SetFactory<L> { }

interface SetFactory<T> {

    Set<T> setOf(T ts);

    List<L> listOf(L ts);

}

<L> List<L> listOf(ListFactory<L> factory, L l) {

    return factory.listOf(l);

}
```



Merge 2 interfaces: use interface where possible

```
interface ListFactory<L> extends SetFactory<L> { }

interface SetFactory<T> {

   Set<T> setOf(T ts);

   List<L> listOf(L ts);

}

<L> List<L> listOf(SetFactory<L> factory, L l) {

   return factory.listOf(l);

}
```

Merge 2 interfaces: safe delete

~~interface ListFactory<L> extends SetFactory<L> { }~~

```
interface SetFactory<T> {
    Set<T> setOf(T ts);
    List<L> listOf(L ts);
}
<L> List<L> listOf(SetFactory<L> factory, L l) {
    return factory.listOf(l);
}
```

Merge 2 interfaces: rename

```
interface CollectionFactory<T> {
    Set<T> setOf(T ts);
    List<L> listOf(L ts);
}
<L> List<L> listOf(CollectionFactory<L> factory, L l) {
    return factory.listOf(l);
}
```

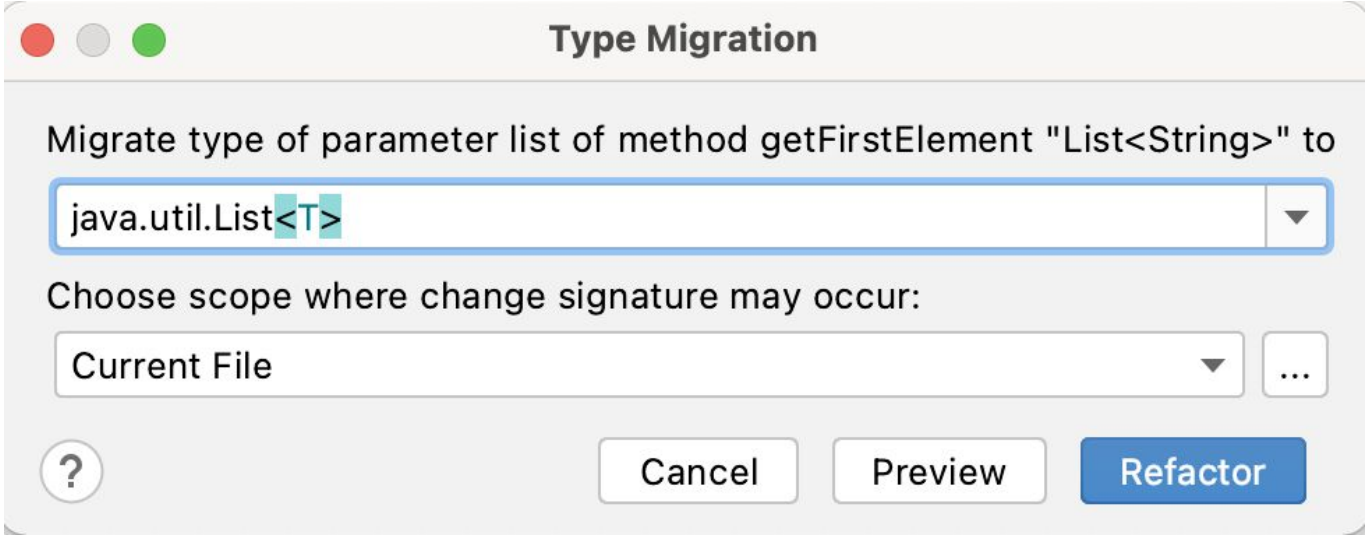
Extract method type parameter

```
public static String getFirstElement(List<String> list) {
    if (list.isEmpty()) return null;
    return list.get(0);
}
void m(List<String> strings){
    String firstStr = getFirstElement(strings);
}
```

Extract method type parameter

```
public static <T> String getFirstElement(List<String> list) {
    if (list.isEmpty()) return null;
    return list.get(0);
}
void m(List<String> strings){
    String firstStr = getFirstElement(strings);
}
```

Extract method type parameter



Extract method type parameter

Find: Migrate Type of parameter **list** from 'List' to 'List'

```
public static <T> String getFirstElement(List<String> list) {(in typeMigration.Demo.getFirstEle
    public static <T> String getFirstElement(List<String> list) {(in typeMigration.Demo)
        String firstStr = getFirstElement(strings);(in typeMigration.Demo.m)
    void m(List<String> strings){(in typeMigration.Demo.m)
```

Structure

Favorites

Migrate | Cancel | Rerun Type Migration | Help

Extract method type parameter

Find: Migrate Type of parameter **list** from 'List' to 'List'

```
public static <T> String getFirstElement(List<String> list) {(in typeMigration.Demo.getFirstEle
  public static <T> String getFirstElement(List<String> list) {(in typeMigration.Demo)
    String firstStr = getFirstElement(strings);(in typeMigration.Demo.m)
  void m(List<String> s                                        .m)
```

- Exclude
- Include ^N
- Jump to Source F4
- Local History
- Git

Migrate | Cancel | Rerun Type Migration | Help

Extract method type parameter

```
public static <T> T getFirstElement(List<T> list) {
    if (list.isEmpty()) return null;
    return list.get(0);
}
void m(List<String> strings){
    String firstStr = getFirstElement(strings);
}
```

Final Thoughts

- In most cases it’s possible to build complex refactorings from small refactoring-steps
- Extract method - Introduce { parameter / variable / parameter object } - Inline allows to achieve a lot!
- Spend a moment before trying to do something manually - check if there is already a refactoring for that

Q & A

Replace collection parameter with return value

```
List<Integer> ints = new ArrayList<>();

callee(ints);

void callee(List<Integer> ints) { }
```

Replace collection parameter with return value: inline parameter

```
List<Integer> ints = new ArrayList<>();

callee();

void callee() {

   ArrayList<Integer> result = new ArrayList<>();

}
```

Replace collection parameter with return value: change signature

```
List<Integer> ints = new ArrayList<>();

ints.addAll(callee());


List<Integer> callee() {

   List<Integer> result = new ArrayList<>();

   return result;

}
```