Structural Search and Replace Examples

This section provides a collection of *structural search and replace* examples:

- Find one statement
- Search in comments and strings
- Add try/catch/finally code
- Finding all descendants of a class, or all classes implementing a certain interface
- Using @Modifier for finding package local and instance methods
- Searching for XML and HTML tags, attributes and their values

One statement

```
$Statement$;
```

Increasing the number of occurrences count to a certain number, you can find sequences of statements that contain up to the specified number of elements.

Method call

```
$Instance$.$MethodCall$($Arguments$)
```

This template matches method call expressions. If the number of occurrences is zero, it means that a method call can be omitted.

If statement

```
      if ($Expr$) {
    $ThenStatements$;
}
else {
    $ElseStatements$;
}
```

Search in comments and/or string literals

Consider one wants to find comments or literal containing 'foo'. Search template would be like `$SomethingWeWantToFind$` or `"$SomethingWeWantToFind$"`. In case one wants to find comments/string containing some particular words (say, foo as a word), this should be specified as a text constraint.

Add try/catch/finally code

If one wants to replace a statement with a `try/catch/finally` construct, the following pair of search and replace templates can be suggested. The search template is:

```
$Statements$;
```

with a certain maximum number of occurrences specified as a constraint.

The replacement template is:

```
      try {
    $Statements$;
}
catch(Exception ex) {
}
```

Finding all descendants of a class or all classes that implement a certain interface

Consider the following search templates:

```
class $Clazz$ extends $AnotherClass$ {}
```

or

```
class $Clazz$ implements $SomeInterface$ {}
```

As the text constraint for the variables $AnotherClass$ or $SomeInterface$, specify the name of the base class or implemented interface.

Finding all such methods

To look for the different implementations of the same interface method, use the following search template:

```
      class $a$ {
    public void $show$();
}
```

Specify text constraint for the `$show$` variable, and enable the option **This variable is the target of the search**.

Using @Modifier for finding package local and instance methods

IntelliJ IDEA suggests pre-defined templates for the *package local* and *instance* fields of a class. These templates make use of the *@Modifier* annotation, which helps describe search target, when there is no way to express it using the natural language means.

However, if you need to search for package local or instance methods, you will have to create the corresponding search templates yourself, applying the *@Modifier* annotation.

To specify criteria for finding all methods with the visibility modifiers *package local* and *instance*, use the following search template:

```
      class
      $Class$ {
    @Modifier("packageLocal") @Modifier("Instance" ) $ReturnType$ $MethodName$($ParameterType$ $P
    }
    }
```

Searching for XML and HTML tags, attributes and their values

The simplest template that is used to search for a tag is:

```
<$a$/>
```

By placing constraints on the variable `$a$`, you can specify which tags you want to find. For example, if you specify the text/regexp constraint `app.+`, you'll find the tags whose names start with `app`.

A more versatile template for searching in XML and HTML is:

```
<$tag$
      $attribute$="
      $value$
      "/>
```

By using this template with properly specified search settings and constraints, you can find practically anything that may occur in XML or HTML. For example, if you specify the text/regexp constraint `width` for the variable `$attribute$`, you'll find all the tags that have the `width` attribute.

See Also

Reference:

- Structural Search and Replace Dialogs

Web Resources:

- Developer Community