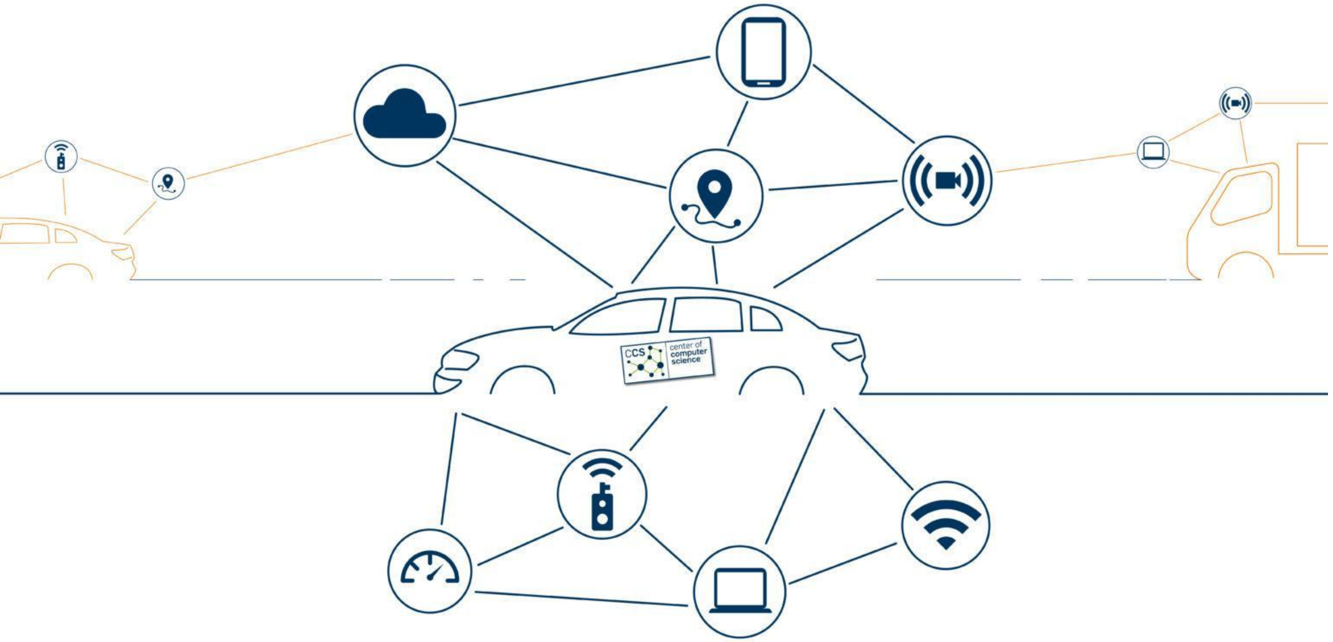


IDE Plugin Development

Winter term 2022/23

Kevin Hermann, Johan Martinson, Thorsten Berger

Prof. Dr. Thorsten Berger
Chair of Software Engineering
Faculty of Computer Science

RUHR UNIVERSITÄT BOCHUM RUB

Agenda

1. Introduction IDE Plugins
2. Organization
3. IDE Plugin Structure
4. Your Task

Introduction IDE Plugins

IDE Plugins

Plugins assist developers in accomplishing a given task

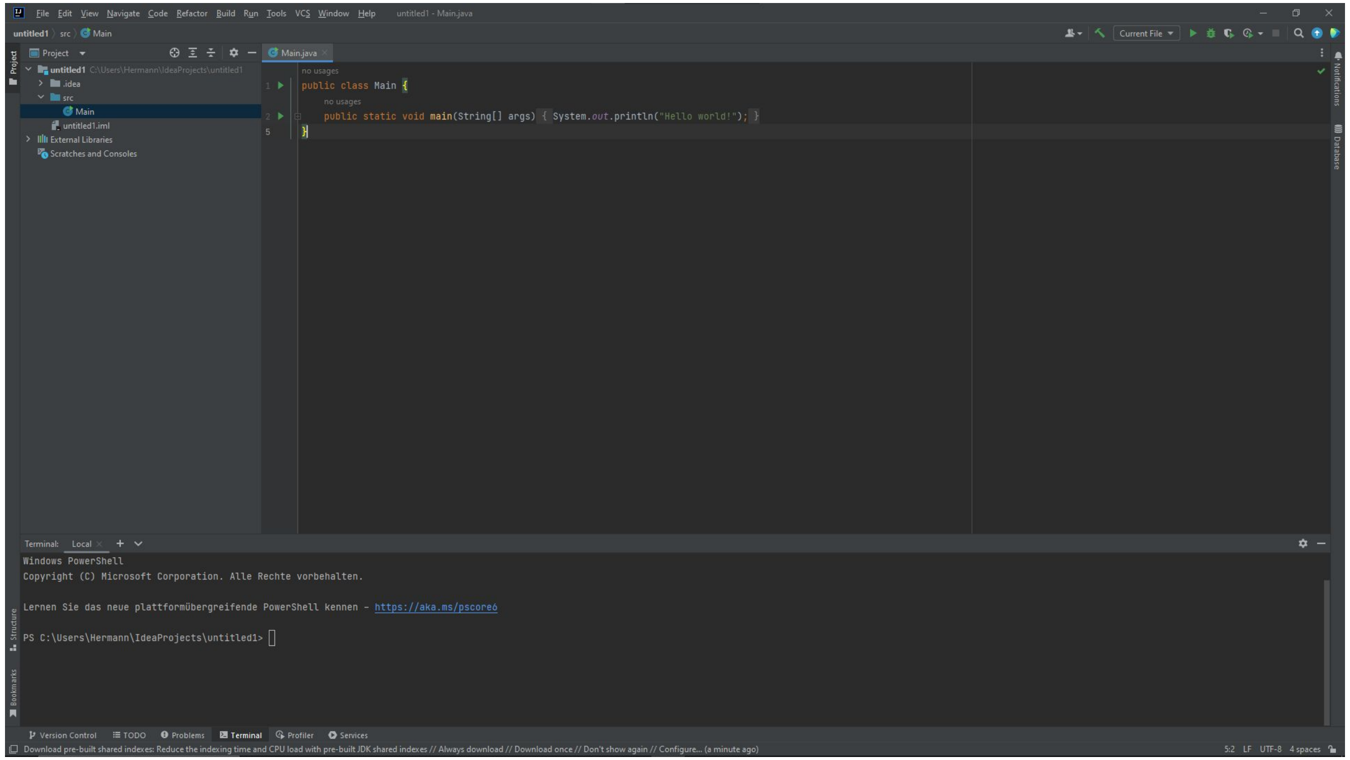
Common types of plugins:

- Custom language support
- Framework integration
- Tool integration
- User interface add-ons
- Themes

IntelliJ IDEA

One of the most popular IDEs used by Java developers

- Syntax highlighting
- Auto-complete/Code suggestion
- Debugging tools
- Unit-testing tools
- Refactoring
- …



```
no usages
public class Main {
    no usages
    public static void main(String[] args) { System.out.println("Hello world!"); }
}
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

Lernen Sie das neue plattformübergreifende PowerShell kennen – https://aka.ms/pscore6

PS C:\Users\Hermann\IdeaProjects\untitled1>
```

Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built JDK shared indexes // Always download // Download once // Don't show again // Configure... (a minute ago)

Agenda

1. ~~Introduction IDE Plugins~~
2. Organization
3. IDE Plugin Structure
4. Your Task

Organization

Organization

In this course you:

- learn how IDE Plugins are built
- comprehend and improve another IDE Plugin
- deliver and present your results
- get prepared to build complex plugins in a potential follow-up thesis

To this end, you will need:

- a partner (teams are already assigned)
- a working IntelliJ IDEA Ultimate installation

Learning

We cover you with the basics:

- Components of Plugins
- Some important classes
- Examples to work with

As we can‘t cover everything IntelliJ has to offer, you might have to do some research on your own

- Some sources are in the Moodle course (If you find good material, feel free to share!)
- Ask questions if you run into problems

Meetings

13.03. 9:30 – 11:30 Introduction MC1.31

14.03. 9:00 – 11:00 HAnS plugin introduction MC1.31

14.03. 16:00 – 18:00 HAnS consultation MC1.31

15.03. 16:00 – 18:00 HAnS consultation and Task distribution MC1.30

17.03. 16:00 – 17:00 Checkpoint MC1.54

Second Week:

20.03. 15:00 – 15:45 Checkpoint (zoom)

21.03. 15:00 – 16:00 Checkpoint (zoom)

23.03. 10:00 – 11:00

28.03. 15:30 – 17:30 final presentations

Agenda

1. ~~Introduction IDE Plugins~~
2. ~~Organization~~
3. IDE Plugin Structure
4. Your Task

IDE Plugin Structure

Plugin Structure

A plugin consists of:

- A Plugin Configuration File
- A Gradle Build File
- Actions
- Extensions
- Services
- Listeners

Configuration File

Contains all the information about our plugin:

author

name

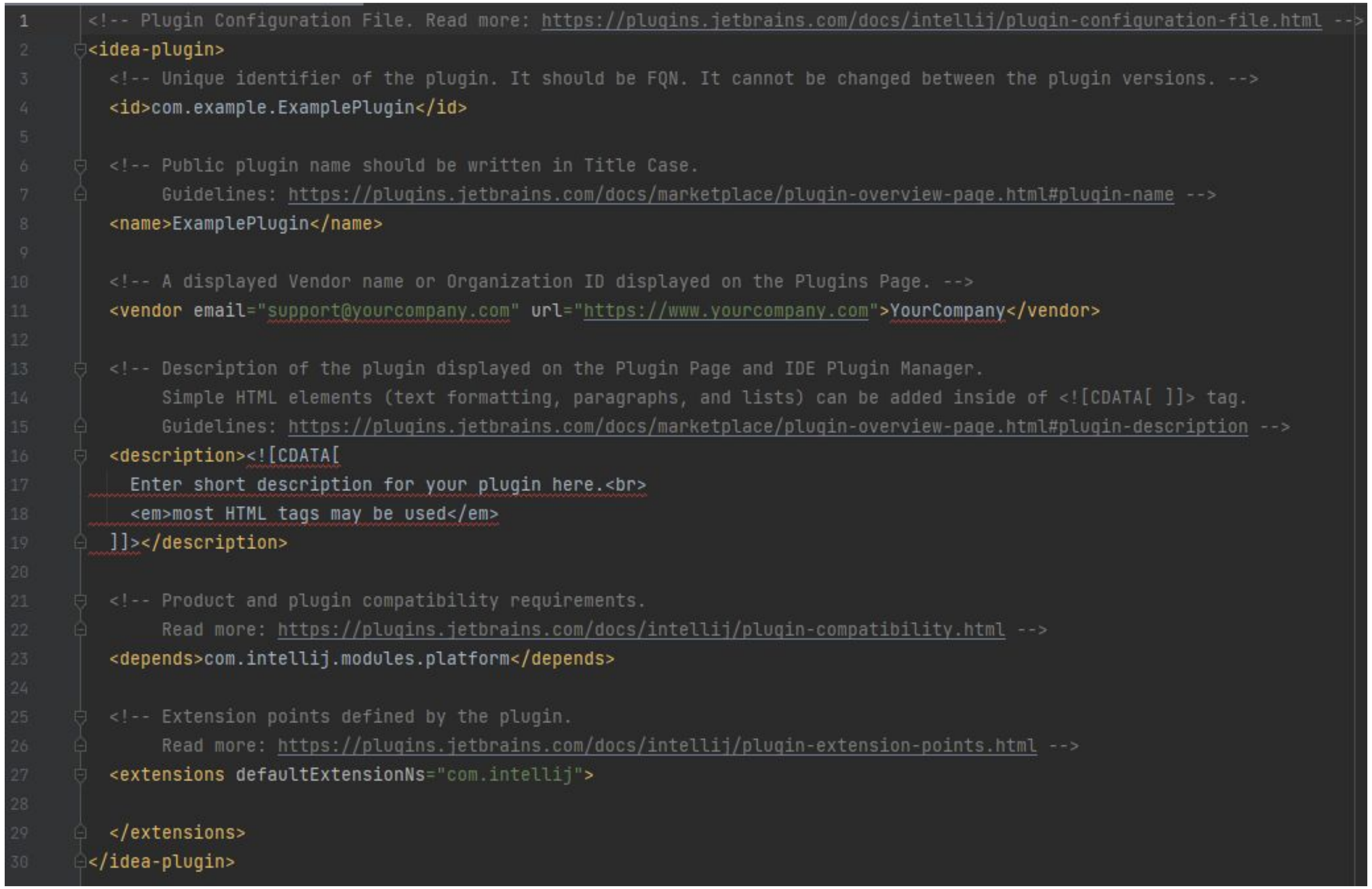
contact details

description

Only relevant if we want to publish our plugin

registered action, listener, extensions, ...

Default Configuration



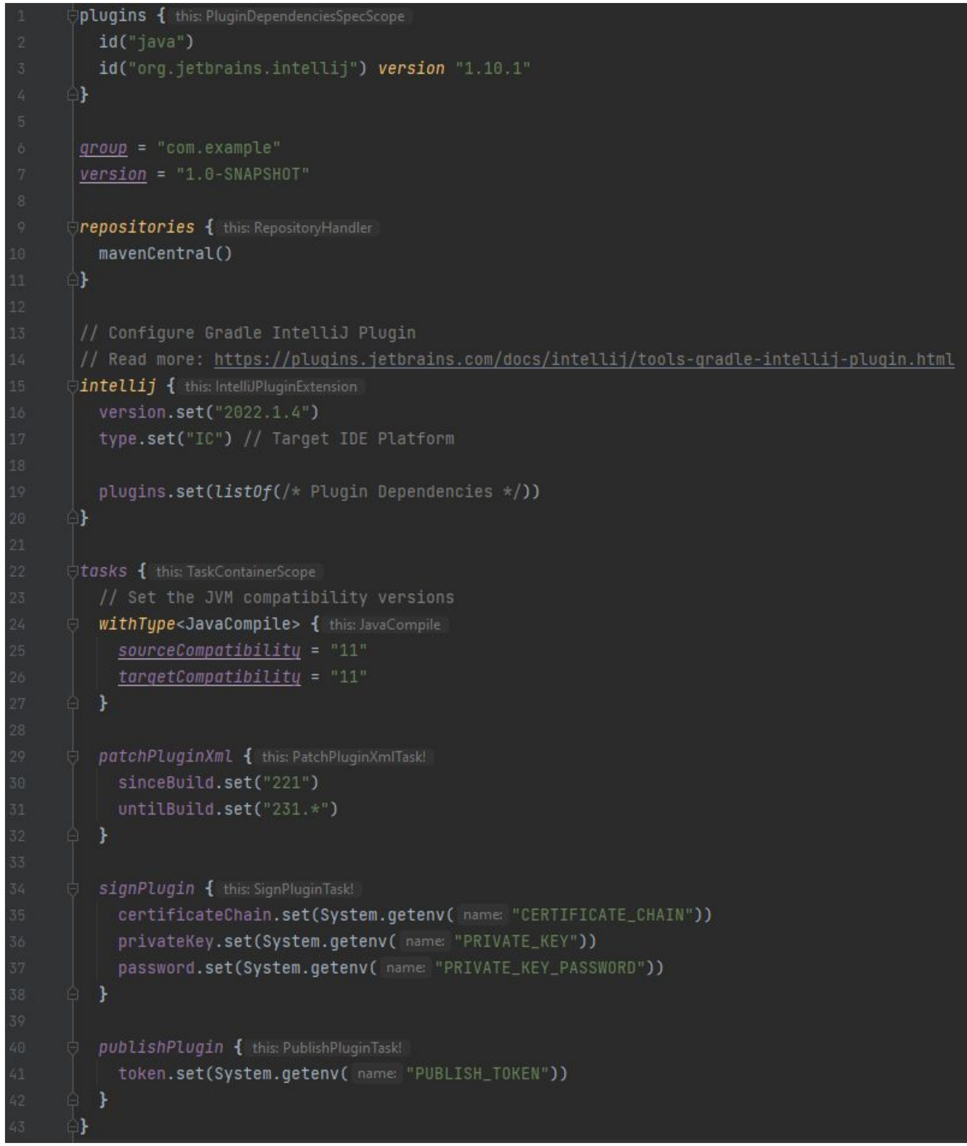
```xml
<!-- Plugin Configuration File. Read more: https://plugins.jetbrains.com/docs/intellij/plugin-configuration-file.html -->
<idea-plugin>
  <!-- Unique identifier of the plugin. It should be FQN. It cannot be changed between the plugin versions. -->
  <id>com.example.ExamplePlugin</id>

  <!-- Public plugin name should be written in Title Case.
       Guidelines: https://plugins.jetbrains.com/docs/marketplace/plugin-overview-page.html#plugin-name -->
  <name>ExamplePlugin</name>

  <!-- A displayed Vendor name or Organization ID displayed on the Plugins Page. -->
  <vendor email="support@yourcompany.com" url="https://www.yourcompany.com">YourCompany</vendor>

  <!-- Description of the plugin displayed on the Plugin Page and IDE Plugin Manager.
       Simple HTML elements (text formatting, paragraphs, and lists) can be added inside of <![CDATA[ ]]> tag.
       Guidelines: https://plugins.jetbrains.com/docs/marketplace/plugin-overview-page.html#plugin-description -->
  <description><![CDATA[
    Enter short description for your plugin here.<br>
    <em>most HTML tags may be used</em>
  ]]></description>

  <!-- Product and plugin compatibility requirements.
       Read more: https://plugins.jetbrains.com/docs/intellij/plugin-compatibility.html -->
  <depends>com.intellij.modules.platform</depends>

  <!-- Extension points defined by the plugin.
       Read more: https://plugins.jetbrains.com/docs/intellij/plugin-extension-points.html -->
  <extensions defaultExtensionNs="com.intellij">

  </extensions>
</idea-plugin>
```

Gradle Build File

Plugins are build with Gradle by default

Build file contains information relevant for building the plugin:

- Java Version
- IntelliJ Version
- Dependencies
- Tasks
- ...



```kotlin
plugins {
    id("java")
    id("org.jetbrains.intellij") version "1.10.1"
}

group = "com.example"
version = "1.0-SNAPSHOT"

repositories {
    mavenCentral()
}

// Configure Gradle IntelliJ Plugin
// Read more: https://plugins.jetbrains.com/docs/intellij/tools-gradle-intellij-plugin.html
intellij {
    version.set("2022.1.4")
    type.set("IC") // Target IDE Platform

    plugins.set(listOf(/* Plugin Dependencies */))
}

tasks {
    // Set the JVM compatibility versions
    withType<JavaCompile> {
        sourceCompatibility = "11"
        targetCompatibility = "11"
    }

    patchPluginXml {
        sinceBuild.set("221")
        untilBuild.set("231.*")
    }

    signPlugin {
        certificateChain.set(System.getenv("CERTIFICATE_CHAIN"))
        privateKey.set(System.getenv("PRIVATE_KEY"))
        password.set(System.getenv("PRIVATE_KEY_PASSWORD"))
    }

    publishPlugin {
        token.set(System.getenv("PUBLISH_TOKEN"))
    }
}
```

Actions

Most common way to invoke functionalities of a plugin

Invoked through:

- Menu or toolbar item
- Keyboard shortcut
- `Help | Find Action...` lookup

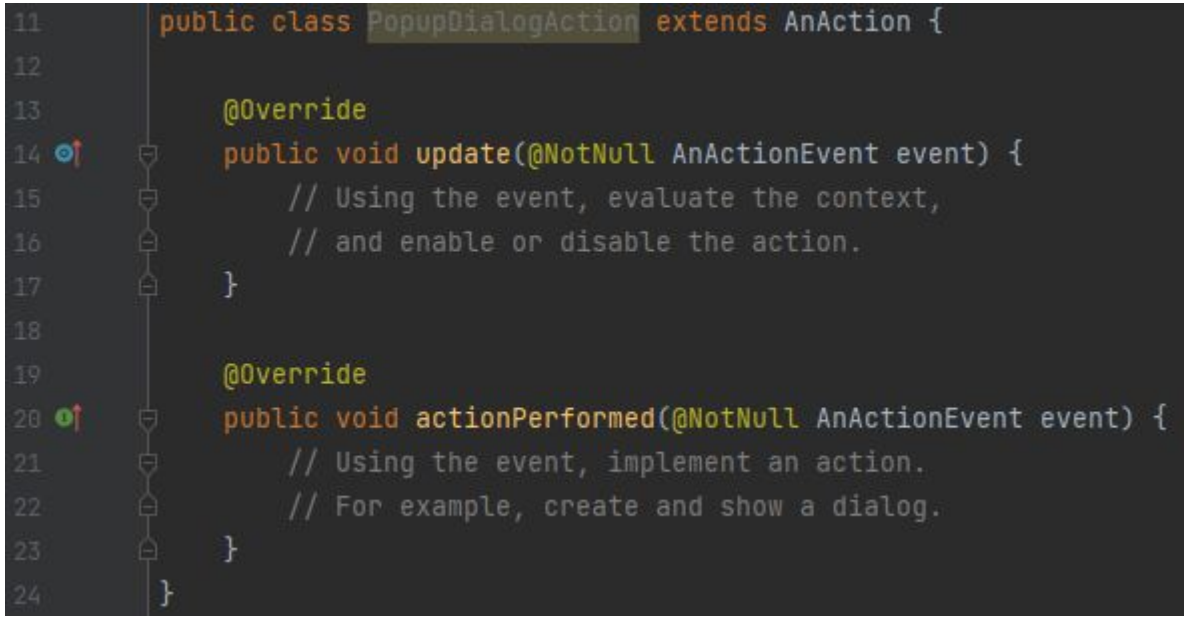
Organized into groups

- Groups of Actions can form a toolbar or a menu

Creating Actions

Create a new java class and extend `AnAction`

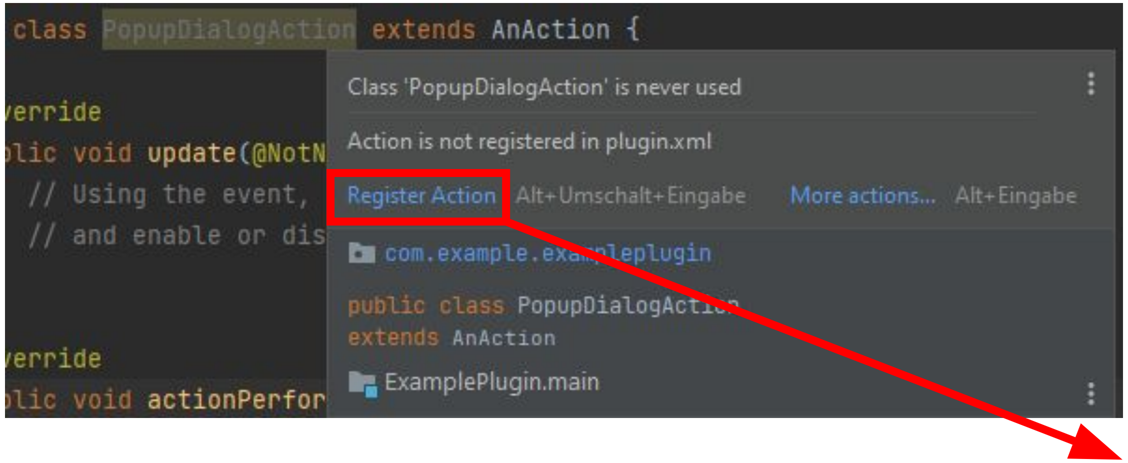
- Override `update(AnActionEvent event)` (Enable or disable the action)
- Override `actionPerformed(AnActionEvent event)` (Implement the action)



```java
public class PopupDialogAction extends AnAction {

    @Override
    public void update(@NotNull AnActionEvent event) {
        // Using the event, evaluate the context,
        // and enable or disable the action.
    }

    @Override
    public void actionPerformed(@NotNull AnActionEvent event) {
        // Using the event, implement an action.
        // For example, create and show a dialog.
    }
}
```

Registering Actions

Actions must be registered in the configuration to be able to use them



```
class PopupDialogAction extends AnAction {

verride
lic void update(@NotN
  // Using the event,
  // and enable or dis

verride
lic void actionPerfor
```

Class 'PopupDialogAction' is never used

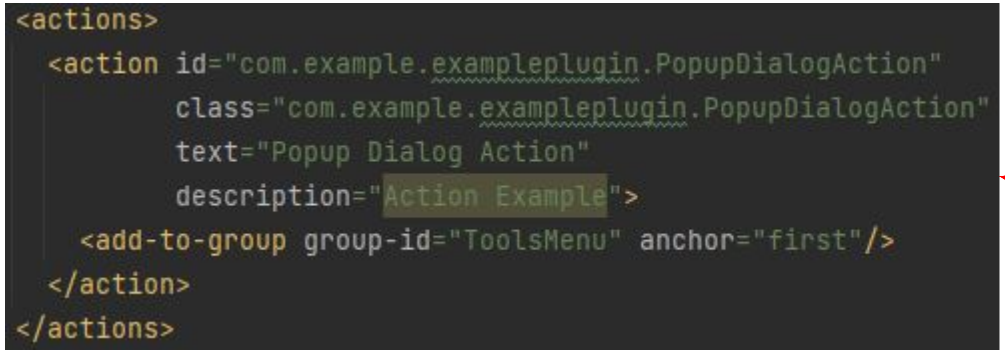
Action is not registered in plugin.xml

Register Action Alt+Umschalt+Eingabe   More actions... Alt+Eingabe

com.example.exampleplugin

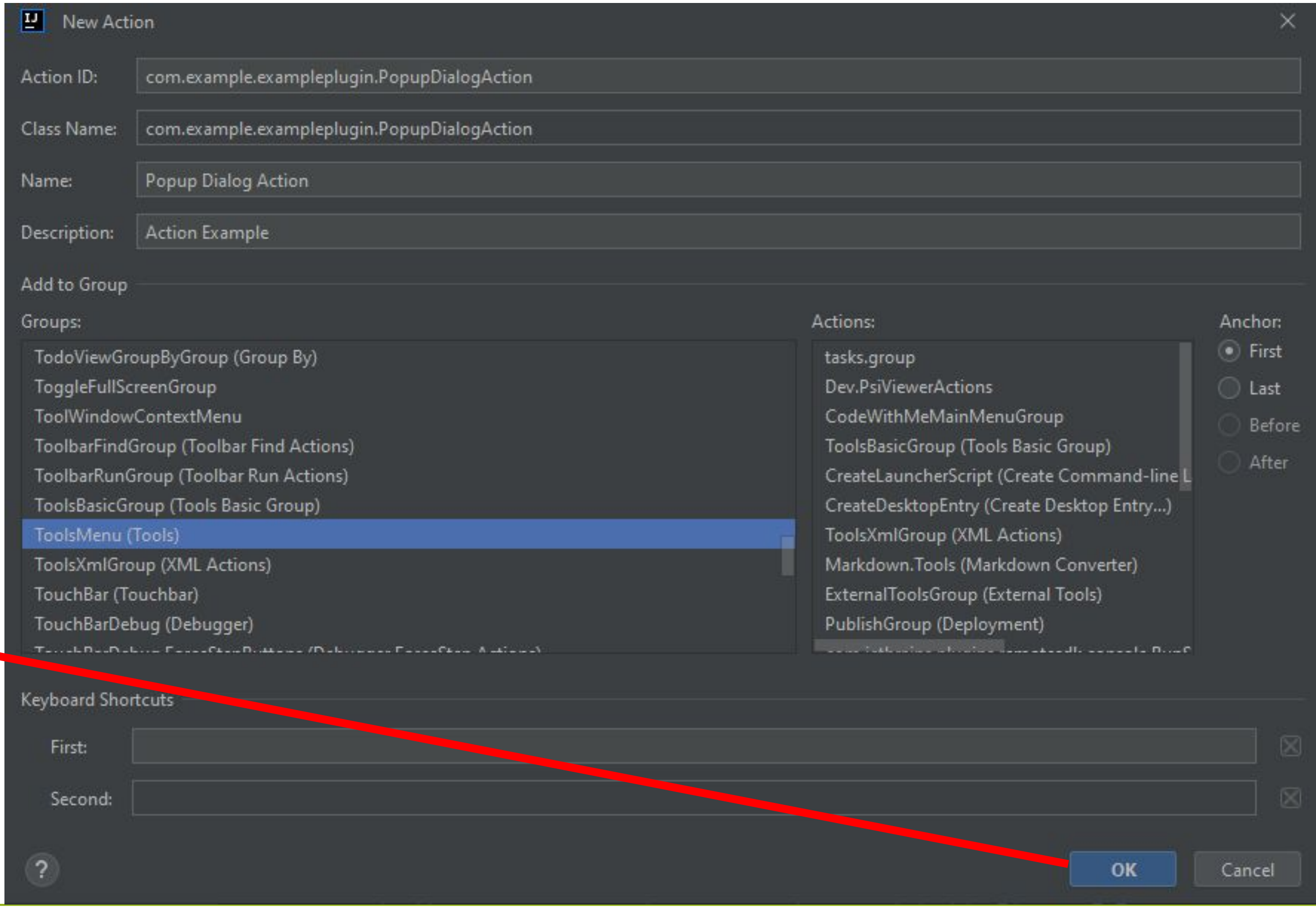
```
public class PopupDialogAction
extends AnAction
```

ExamplePlugin.main



```
<actions>
  <action id="com.example.exampleplugin.PopupDialogAction"
          class="com.example.exampleplugin.PopupDialogAction"
          text="Popup Dialog Action"
          description="Action Example">
    <add-to-group group-id="ToolsMenu" anchor="first"/>
  </action>
</actions>
```

Plugin Configuration file



New Action

Action ID: com.example.exampleplugin.PopupDialogAction

Class Name: com.example.exampleplugin.PopupDialogAction

Name: Popup Dialog Action

Description: Action Example

Add to Group

Groups:
- TodoViewGroupByGroup (Group By)
- ToggleFullScreenGroup
- ToolWindowContextMenu
- ToolbarFindGroup (Toolbar Find Actions)
- ToolbarRunGroup (Toolbar Run Actions)
- ToolsBasicGroup (Tools Basic Group)
- ToolsMenu (Tools)
- ToolsXmlGroup (XML Actions)
- TouchBar (Touchbar)
- TouchBarDebug (Debugger)

Actions:
- tasks.group
- Dev.PsiViewerActions
- CodeWithMeMainMenuGroup
- ToolsBasicGroup (Tools Basic Group)
- CreateLauncherScript (Create Command-line L
- CreateDesktopEntry (Create Desktop Entry...)
- ToolsXmlGroup (XML Actions)
- Markdown.Tools (Markdown Converter)
- ExternalToolsGroup (External Tools)
- PublishGroup (Deployment)

Anchor:
- ● First
- ○ Last
- ○ Before
- ○ After

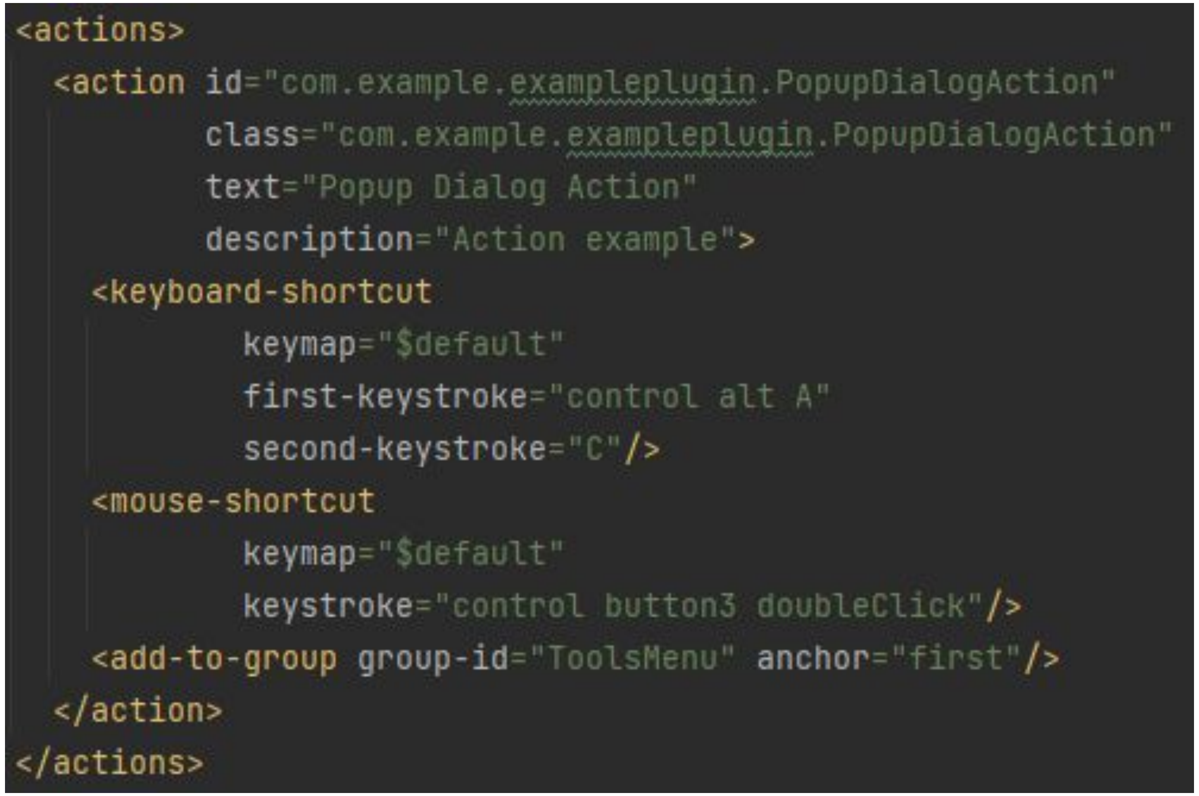
Keyboard Shortcuts

First:

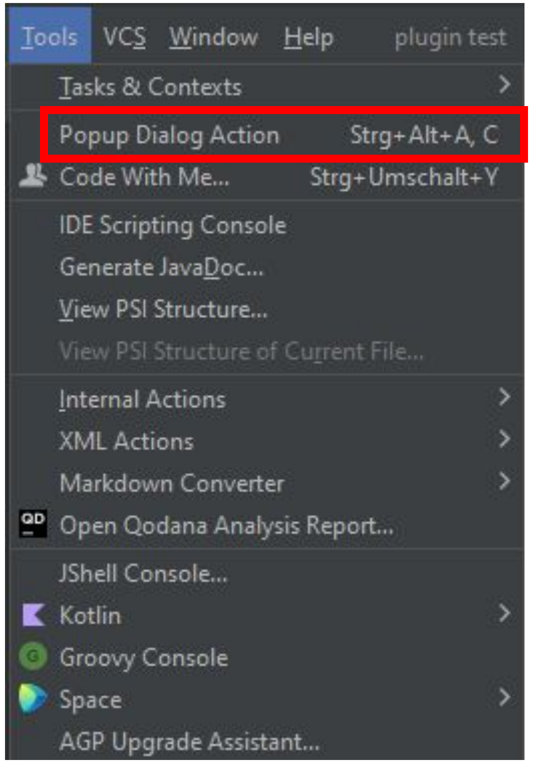
Second:

Manual Attribute Registration

We can also register and modify action manually in the `plugin.xml`

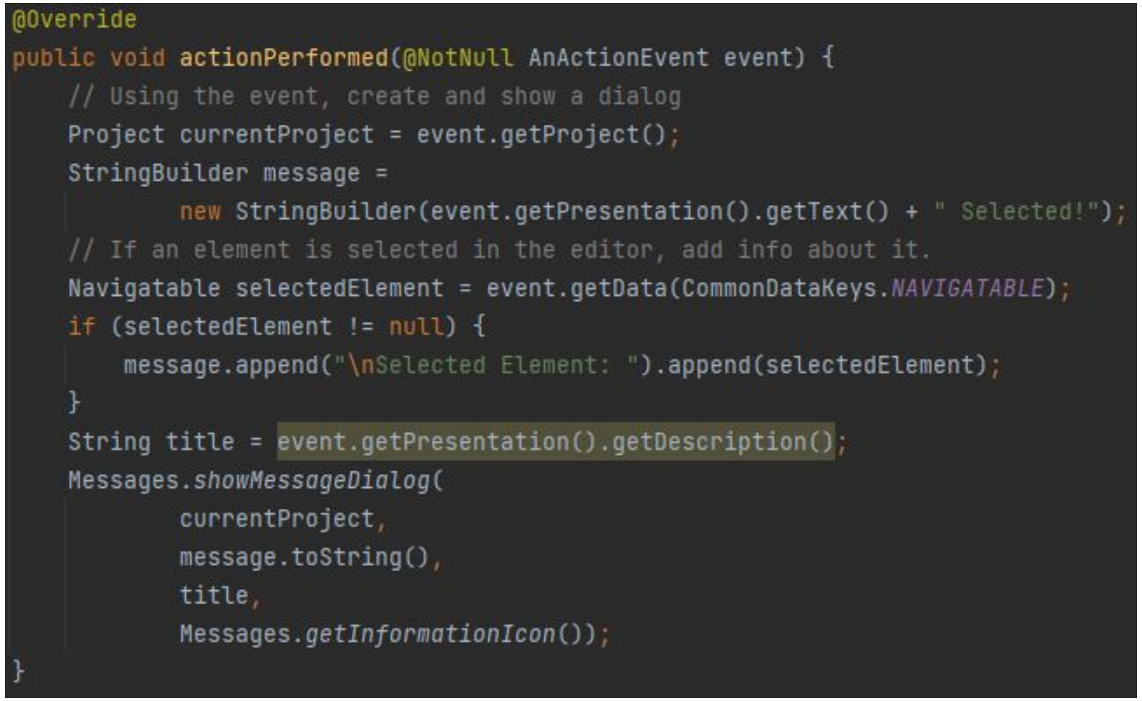


```
<actions>
  <action id="com.example.exampleplugin.PopupDialogAction"
          class="com.example.exampleplugin.PopupDialogAction"
          text="Popup Dialog Action"
          description="Action example">
    <keyboard-shortcut
            keymap="$default"
            first-keystroke="control alt A"
            second-keystroke="C"/>
    <mouse-shortcut
            keymap="$default"
            keystroke="control button3 doubleClick"/>
    <add-to-group group-id="ToolsMenu" anchor="first"/>
  </action>
</actions>
```

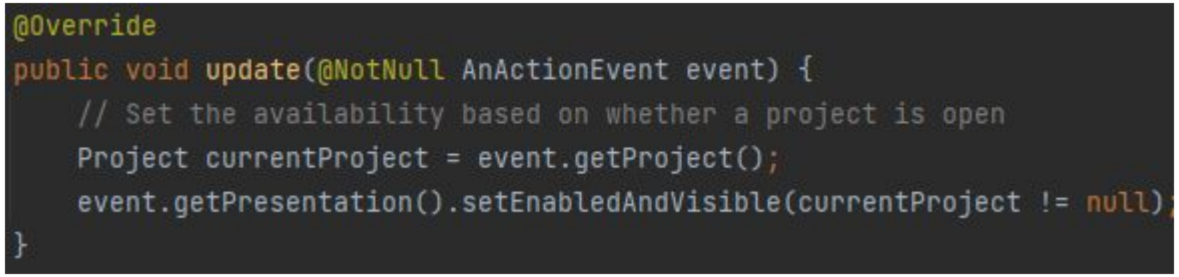


Implementing Actions - Example

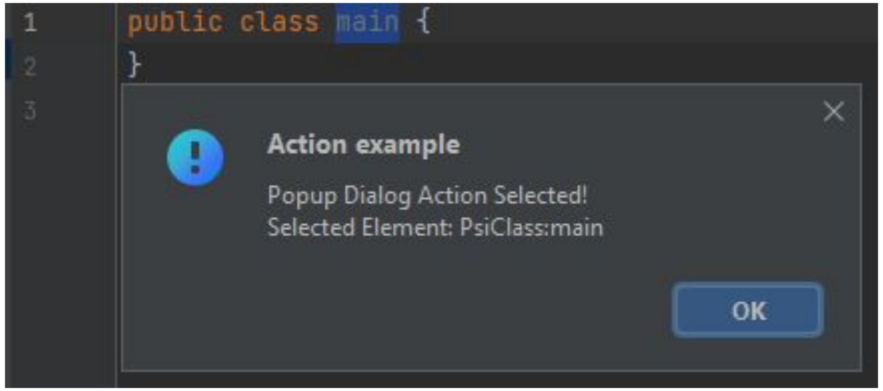
This example shows a popup dialog and shows a message containing the currently selected element



```java
@Override
public void actionPerformed(@NotNull AnActionEvent event) {
    // Using the event, create and show a dialog
    Project currentProject = event.getProject();
    StringBuilder message =
            new StringBuilder(event.getPresentation().getText() + " Selected!");
    // If an element is selected in the editor, add info about it.
    Navigatable selectedElement = event.getData(CommonDataKeys.NAVIGATABLE);
    if (selectedElement != null) {
        message.append("\nSelected Element: ").append(selectedElement);
    }
    String title = event.getPresentation().getDescription();
    Messages.showMessageDialog(
            currentProject,
            message.toString(),
            title,
            Messages.getInformationIcon());
}
```



```java
@Override
public void update(@NotNull AnActionEvent event) {
    // Set the availability based on whether a project is open
    Project currentProject = event.getProject();
    event.getPresentation().setEnabledAndVisible(currentProject != null);
}
```



```java
public class main {
}
```

Action example

Popup Dialog Action Selected!
Selected Element: PsiClass:main

Class AnActionEvent

Contains information necessary to execute or update an action

Important methods:

- `getPresentation()` – returns the presentation of `AnActionEvent`
- `getProject()` – returns the current project
- `getData(DataKey<T> key)` – return the context of the action

CommonDataKeys

Class of keys to access common data and resources within the IntelliJ-Platform

Important examples:

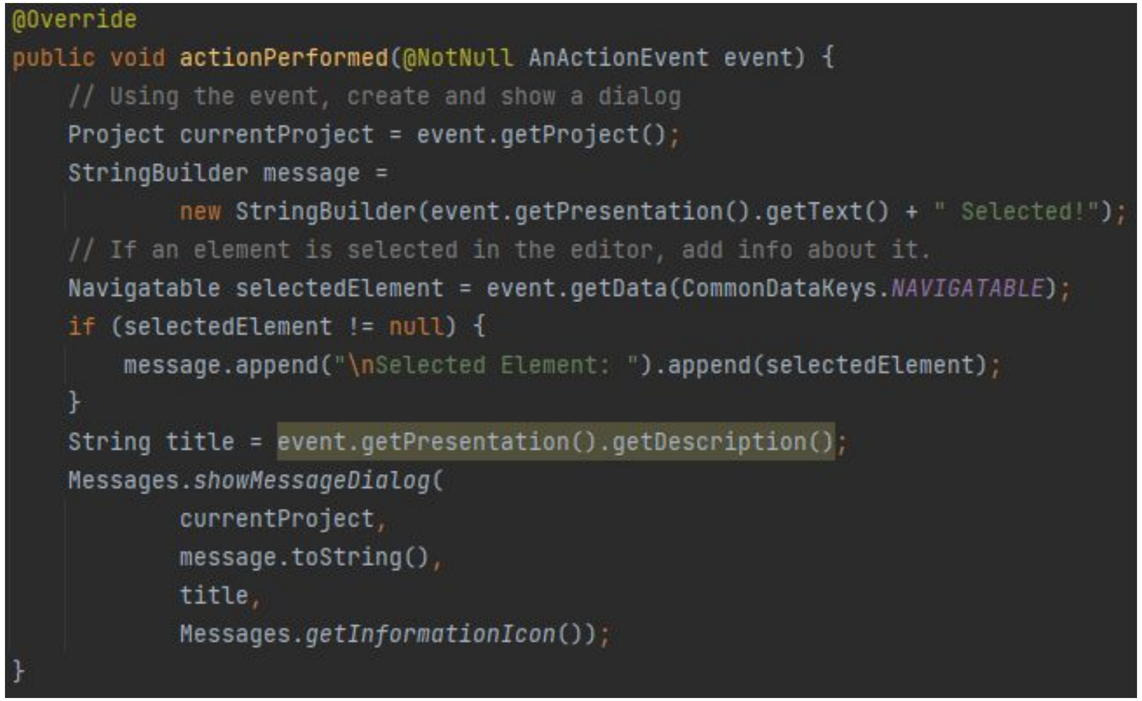
- `CARET` – Position of the Cursor
- `EDITOR` – Active editor on which the action is invoked on
- `NAVIGATABLE` – Active object which can be shown in the IDE (e.g., a file, a class, ...)
- `PROJECT` – Active project (Same as getProject())
- `PSI_ELEMENT` – Active PSI_Element
- `SELECTION` – Currently selected text in the editor

CommonDataKeys – Full list

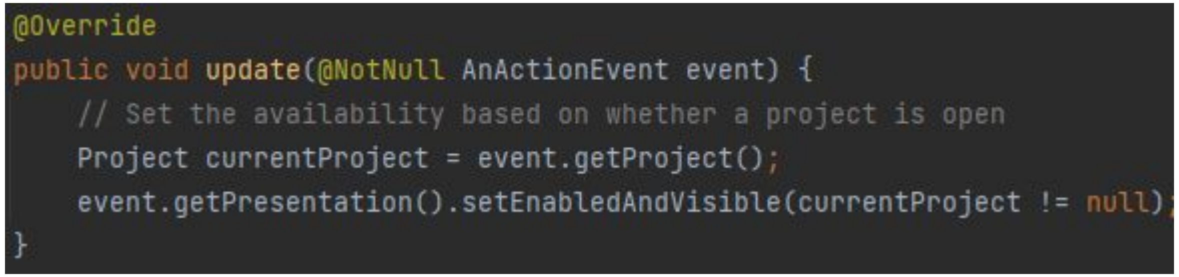
```
ACTIVE_EDITOR
ACTIVE_PROJECT
ACTIVE_VCS_DOCUMENT
CARET
EDITOR
EDITOR_CONTENTS
EDITOR_EVEN_IF_INACTIVE
FILE_EDITOR
MODULE
MODULE_DIR
NAVIGATABLE
NAVIGATABLE_DIR
PSI_ELEMENT
PSI_FILE
SELECTION
SOURCE_POSITION
VIRTUAL_FILE
VCS
VCS_FILE_STATUS
WORKSPACE
XDEBUG_SESSION
```

Implementing Actions - Example

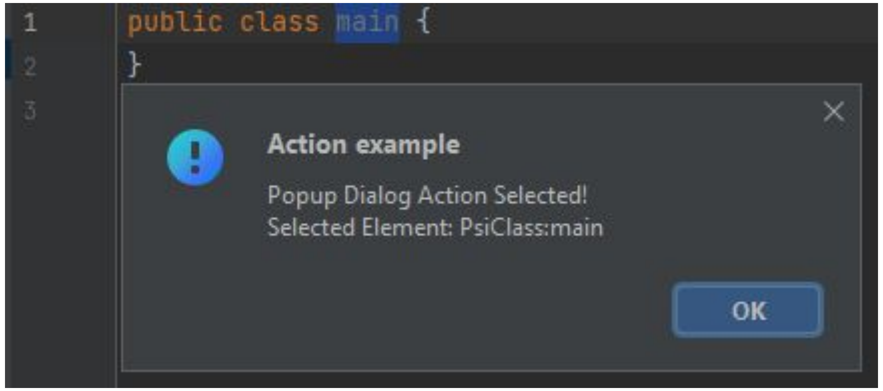
This example shows a popup dialog and shows a message containing the currently selected element



```java
@Override
public void actionPerformed(@NotNull AnActionEvent event) {
    // Using the event, create and show a dialog
    Project currentProject = event.getProject();
    StringBuilder message =
            new StringBuilder(event.getPresentation().getText() + " Selected!");
    // If an element is selected in the editor, add info about it.
    Navigatable selectedElement = event.getData(CommonDataKeys.NAVIGATABLE);
    if (selectedElement != null) {
        message.append("\nSelected Element: ").append(selectedElement);
    }
    String title = event.getPresentation().getDescription();
    Messages.showMessageDialog(
            currentProject,
            message.toString(),
            title,
            Messages.getInformationIcon());
}
```



```java
@Override
public void update(@NotNull AnActionEvent event) {
    // Set the availability based on whether a project is open
    Project currentProject = event.getProject();
    event.getPresentation().setEnabledAndVisible(currentProject != null);
}
```



```java
public class main {
}
```

Action example

Popup Dialog Action Selected!
Selected Element: PsiClass:main

Program Structure Interface (PSI) File

Internal representation of the source code in the IntelliJ-Platform

Build hierarchically (Contains Classes, Methods, Field, Parameters, ...)

Used by IntelliJ for some powerful features:

- Code analysis (Display errors or warnings in the terminal, ...)
- Refactoring (Rename variables, optimize imports, ...)
- Code generation (Generate getters, setters, constructors, ...)
- Code completion (Provide suggestions for current context)

Important PSI-classes

`PsiElement` - Most basic class of all PSI-elements

`PsiFile` - Represents a file in IntelliJ IDEA

`PsiClass` - Represents a Java class

`PsiField` - Represents a field of a Java class

`PsiMethod` - Represents a method of a Java class

`PsiParameter` - Represents a parameter of a method

`PsiAnnotation` - Represents an annotation of a Java class

`PsiStatement` - Represents all Java statements (e.g., if, while, for, ...)

Navigating the PSI

There are plenty of useful methods for selecting and navigating PSI components

Examples:

`PsiClass.getMethods()` - Returns an array of all methods of a class

`PsiMethod.getNameIdentifier()` - Returns the identifier of a method

`PsiField.getType()` - Returns the type of a field

`PsiElement.getParent()` - Returns the parent of any element

`PsiElement.getChildren()` - Returns an array of all children of any element

For language-independent navigation, consider using `PsiTreeUtil`

More useful classes and methods

IntelliJ prohibits code generation and deletion without ensuring the changes are undoable

Use the class `WriteCommandAction` to make it undoable

The class `PropertyUtil` has some useful methods to write to PSI-files

`getName(PsiNamedElement element)` – Get name of a PSI-element

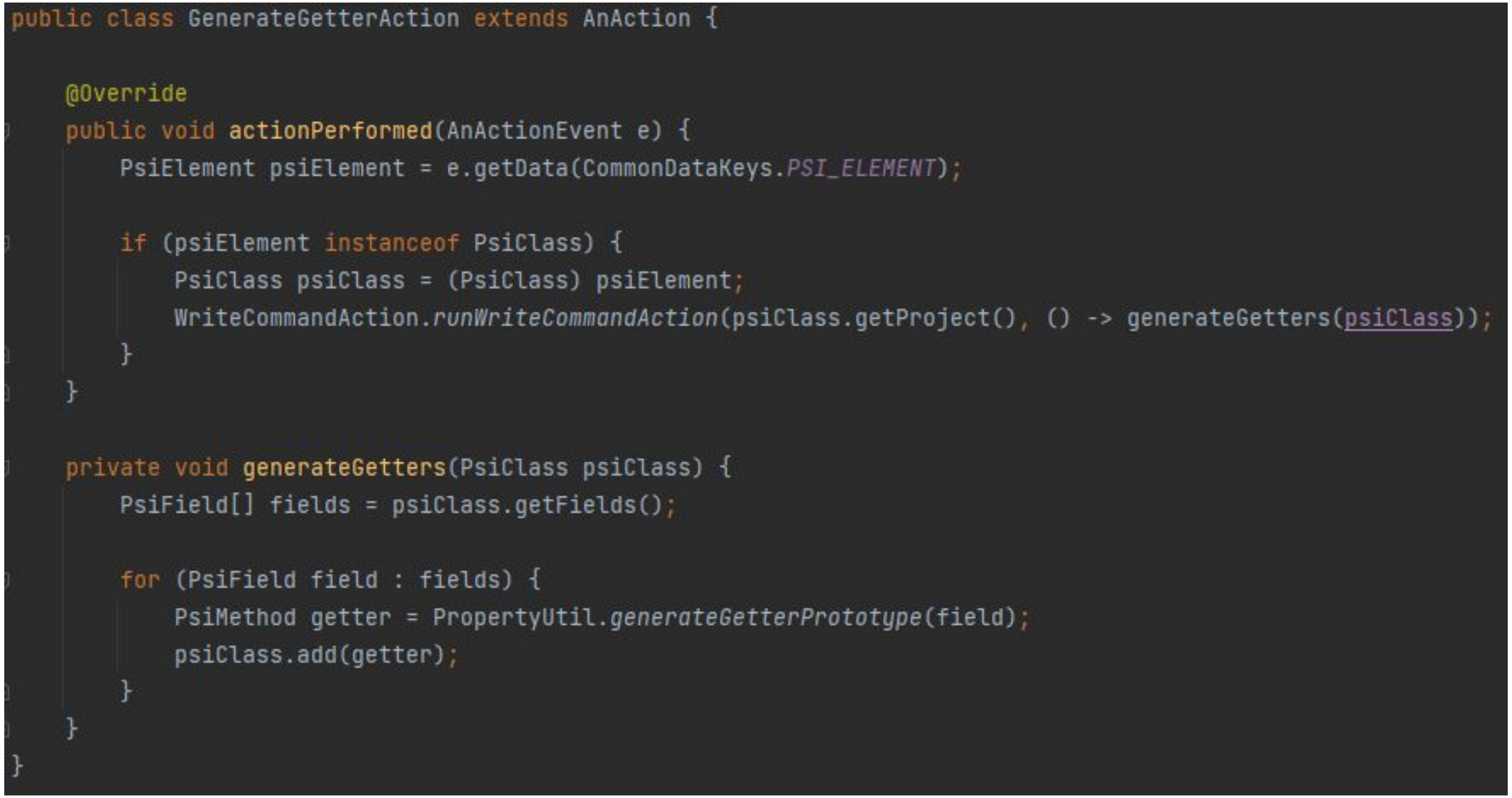
`setName(PsiNamedElement element, String name)` – Set name of a PSI-element

`insertAfter(PsiElement anchor, PsiElement[] elements)` – Insert PSI-elements after anchor

`getContainingFile(PsiElement element)` – Returns file, which contains a PSI-element

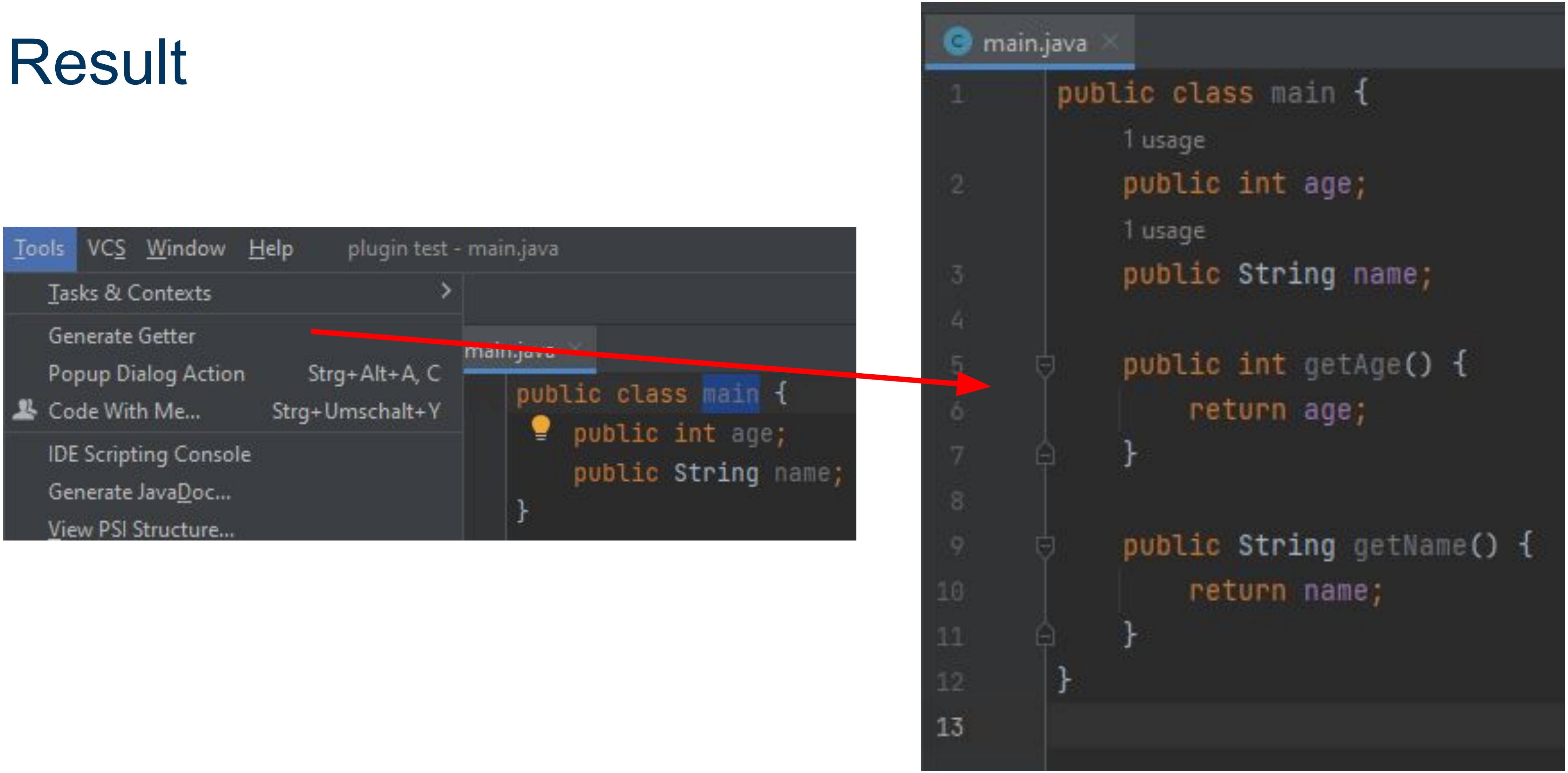
`generateGetterPrototype(PsiField field)` – Generates a getter for a field

Generate Getters using PSI



```
public class GenerateGetterAction extends AnAction {

    @Override
    public void actionPerformed(AnActionEvent e) {
        PsiElement psiElement = e.getData(CommonDataKeys.PSI_ELEMENT);

        if (psiElement instanceof PsiClass) {
            PsiClass psiClass = (PsiClass) psiElement;
            WriteCommandAction.runWriteCommandAction(psiClass.getProject(), () -> generateGetters(psiClass));
        }
    }

    private void generateGetters(PsiClass psiClass) {
        PsiField[] fields = psiClass.getFields();

        for (PsiField field : fields) {
            PsiMethod getter = PropertyUtil.generateGetterPrototype(field);
            psiClass.add(getter);
        }
    }
}
```

Result



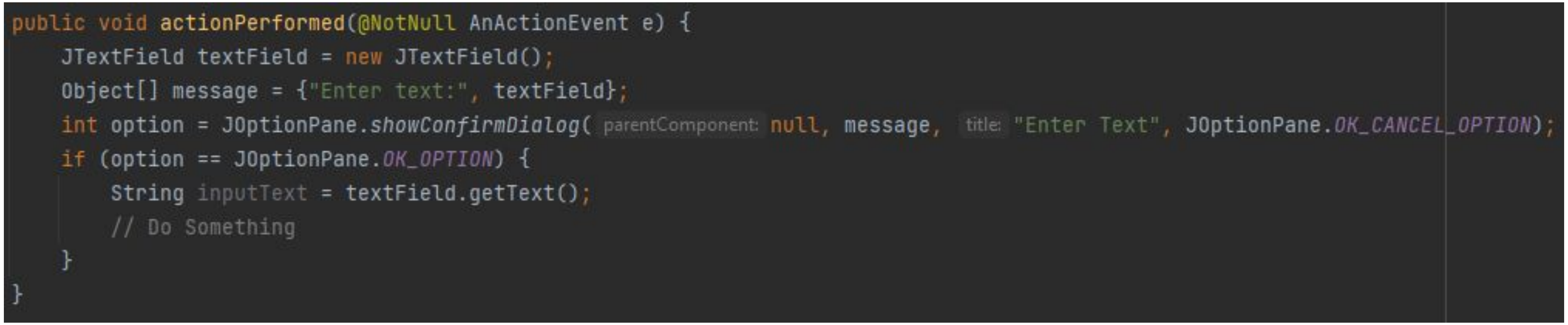
```java
public class main {
    public int age;
    public String name;
}
```

```java
public class main {
    public int age;
    public String name;

    public int getAge() {
        return age;
    }

    public String getName() {
        return name;
    }
}
```

Displaying Textboxes

Using GUI toolkits enables taking user input

Example using Swing Framework:



```
public void actionPerformed(@NotNull AnActionEvent e) {
    JTextField textField = new JTextField();
    Object[] message = {"Enter text:", textField};
    int option = JOptionPane.showConfirmDialog( parentComponent: null, message,  title: "Enter Text", JOptionPane.OK_CANCEL_OPTION);
    if (option == JOptionPane.OK_OPTION) {
        String inputText = textField.getText();
        // Do Something
    }
}
```

Extensions

Another common way to provides functionalities

Used if task cannot be accomplished by an action:

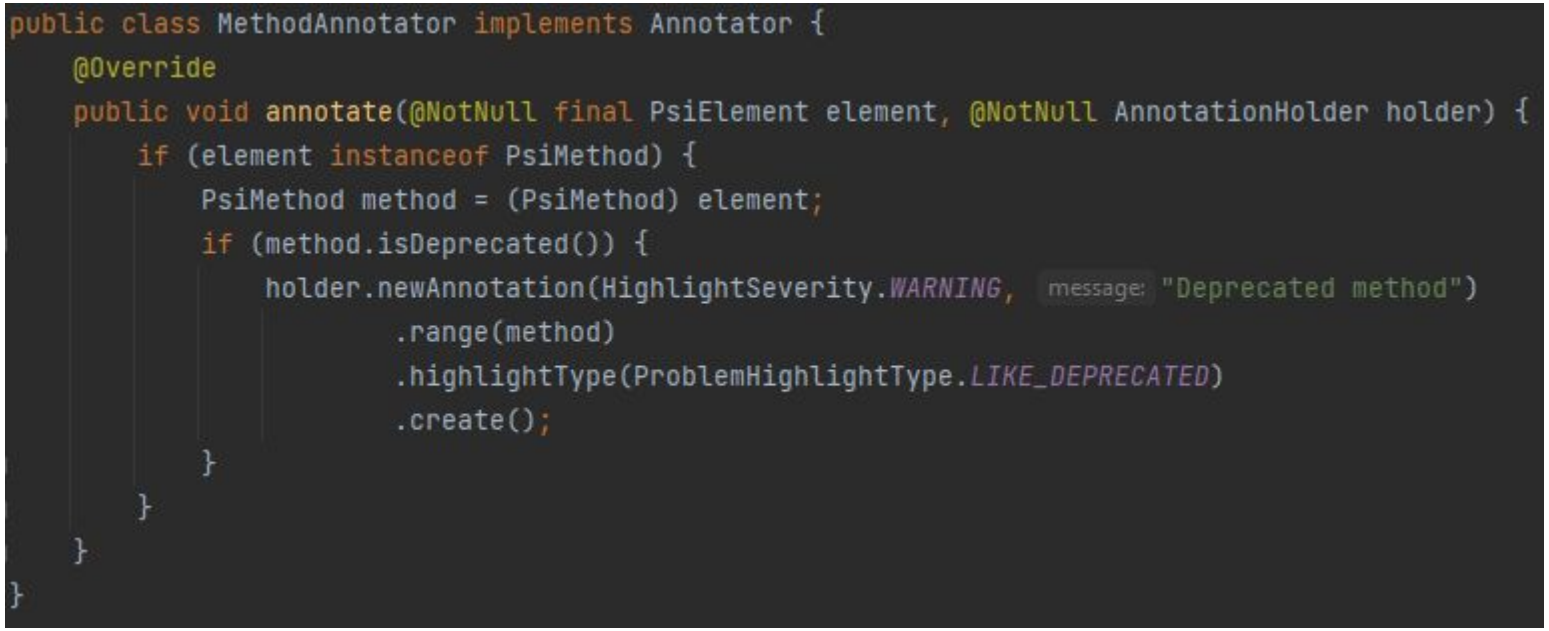
- Display a tool window (panels on the user interface)
- Add pages to the settings dialog
- Custom language support features (such as syntax highlighting)

More than 1000 extension points in the IntelliJ platform

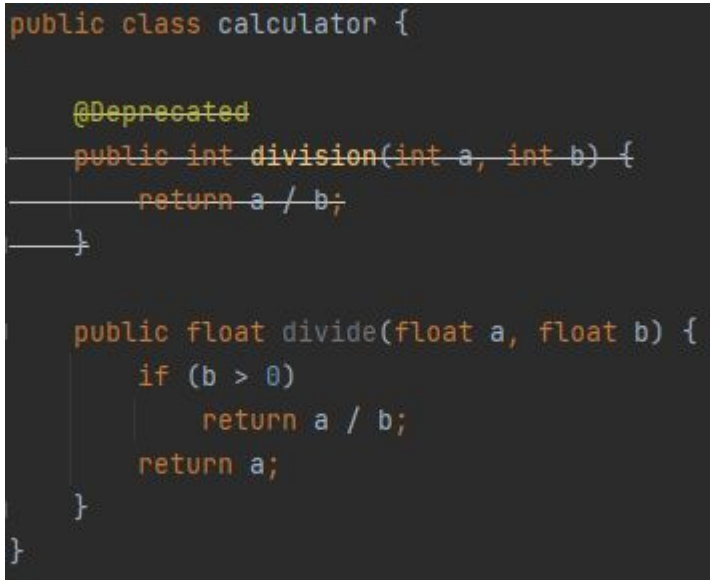
Extension Example: Highlighting Code

Implementing the Interface Annotator enables us to highlight code based on self-defined criteria

Annotations are applied over the whole project



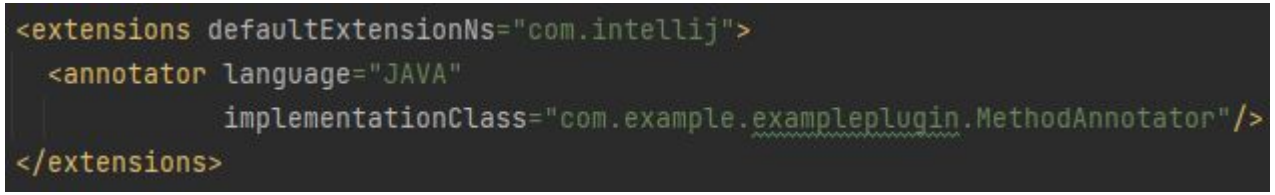
```java
public class MethodAnnotator implements Annotator {
    @Override
    public void annotate(@NotNull final PsiElement element, @NotNull AnnotationHolder holder) {
        if (element instanceof PsiMethod) {
            PsiMethod method = (PsiMethod) element;
            if (method.isDeprecated()) {
                holder.newAnnotation(HighlightSeverity.WARNING,  message: "Deprecated method")
                        .range(method)
                        .highlightType(ProblemHighlightType.LIKE_DEPRECATED)
                        .create();
            }
        }
    }
}
```



```java
public class calculator {

    ~~@Deprecated~~
    ~~public int division(int a, int b) {~~
        ~~return a / b;~~
    ~~}~~

    public float divide(float a, float b) {
        if (b > 0)
            return a / b;
        return a;
    }
}
```

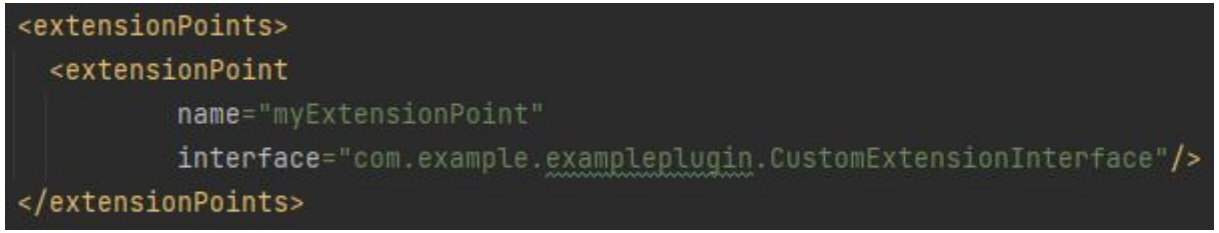
Registering Extensions

In the `plugin.xml` we have to declare the class implementing an extension



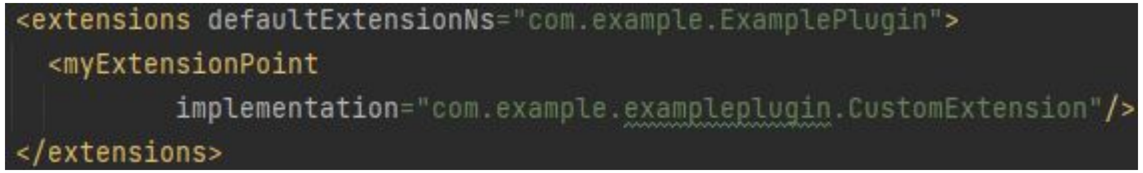
```
<extensions defaultExtensionNs="com.intellij">
  <annotator language="JAVA"
             implementationClass="com.example.exampleplugin.MethodAnnotator"/>
</extensions>
```

Custom extension points allow other plugins to extend our plugin‘s functionality



```
<extensionPoints>
  <extensionPoint
          name="myExtensionPoint"
          interface="com.example.exampleplugin.CustomExtensionInterface"/>
</extensionPoints>
```

Declaring extension points



```
<extensions defaultExtensionNs="com.example.ExamplePlugin">
  <myExtensionPoint
          implementation="com.example.exampleplugin.CustomExtension"/>
</extensions>
```

Using extension points in another plugin

Services

Services are central points to pull data or execute reusable methods from within the IDE

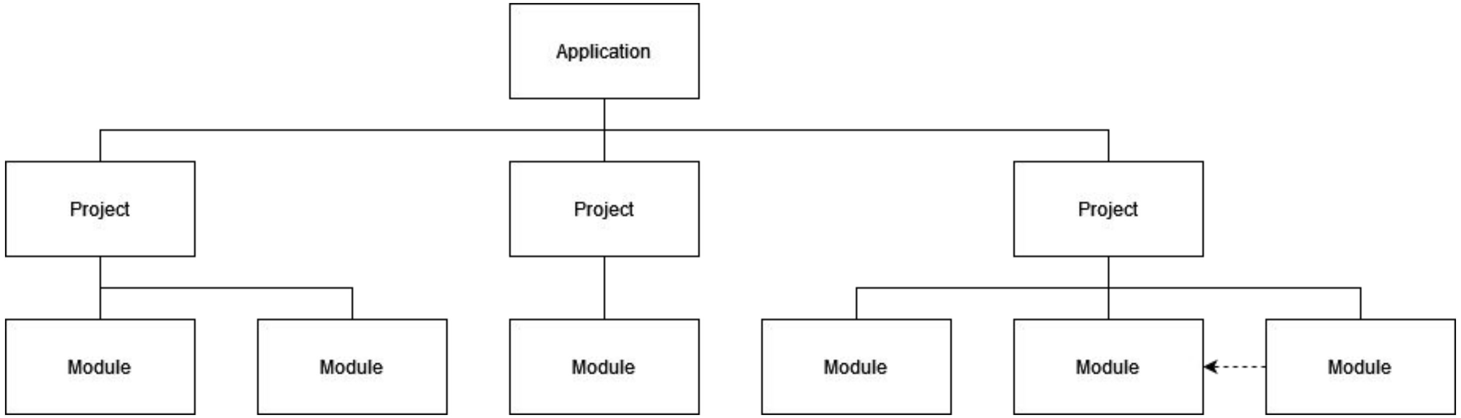
Requires a custom implementation

The ServiceManager can be used to access a service

- Always ensures that only one instance of a service is running

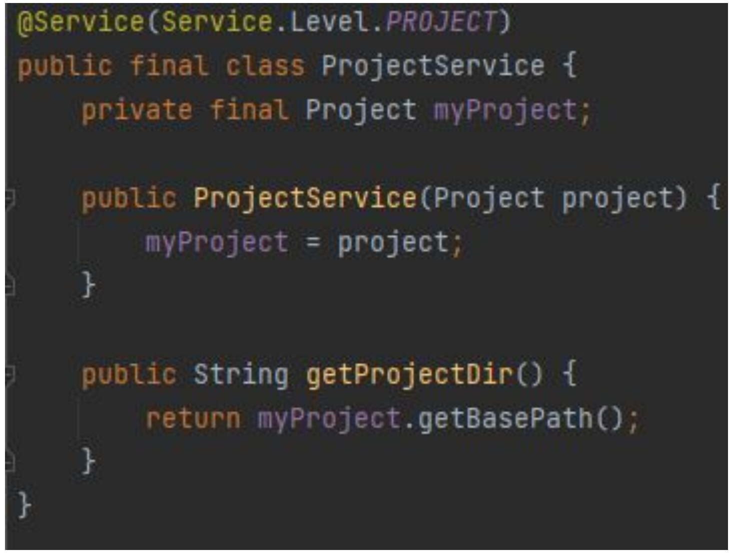
Three types of services

- Application-level
- Project-level
- Module-level



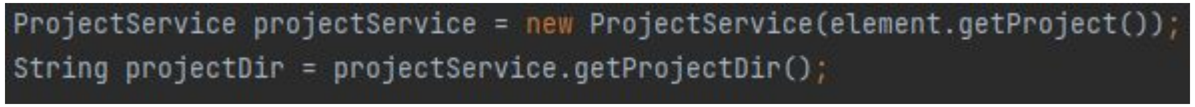
Project Service Example

Use `@Service` Annotation to declare a service



```java
@Service(Service.Level.PROJECT)
public final class ProjectService {
    private final Project myProject;

    public ProjectService(Project project) {
        myProject = project;
    }

    public String getProjectDir() {
        return myProject.getBasePath();
    }
}
```

Implementing a service



```java
ProjectService projectService = new ProjectService(element.getProject());
String projectDir = projectService.getProjectDir();
```

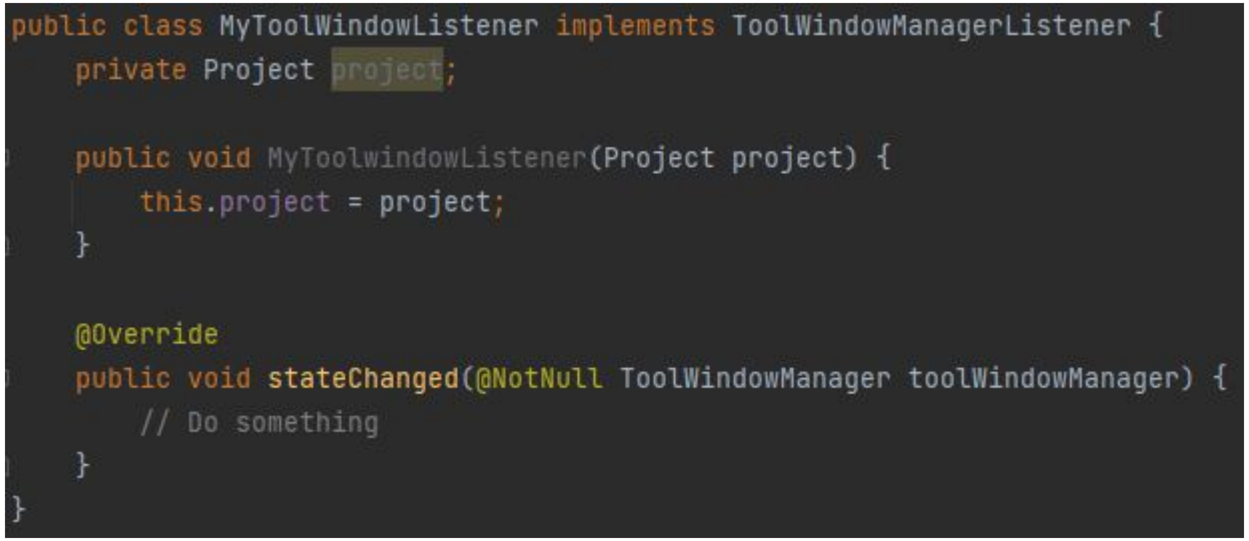
Using a service

Listeners

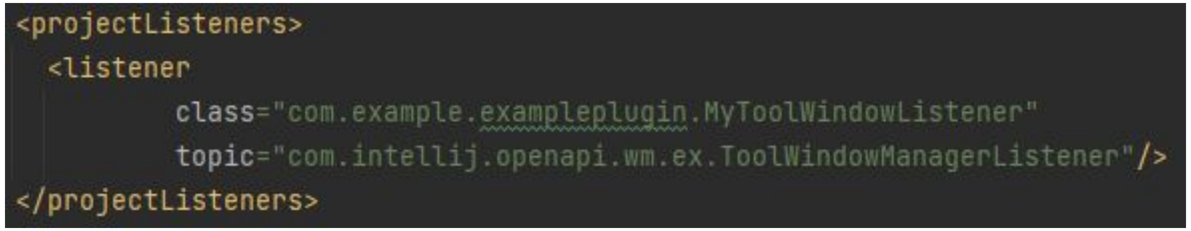
IntelliJ uses a Publisher Subscriber Pattern

- Subscribe to a topic and receive messages about events

Create a class implementing a listener for a specific event



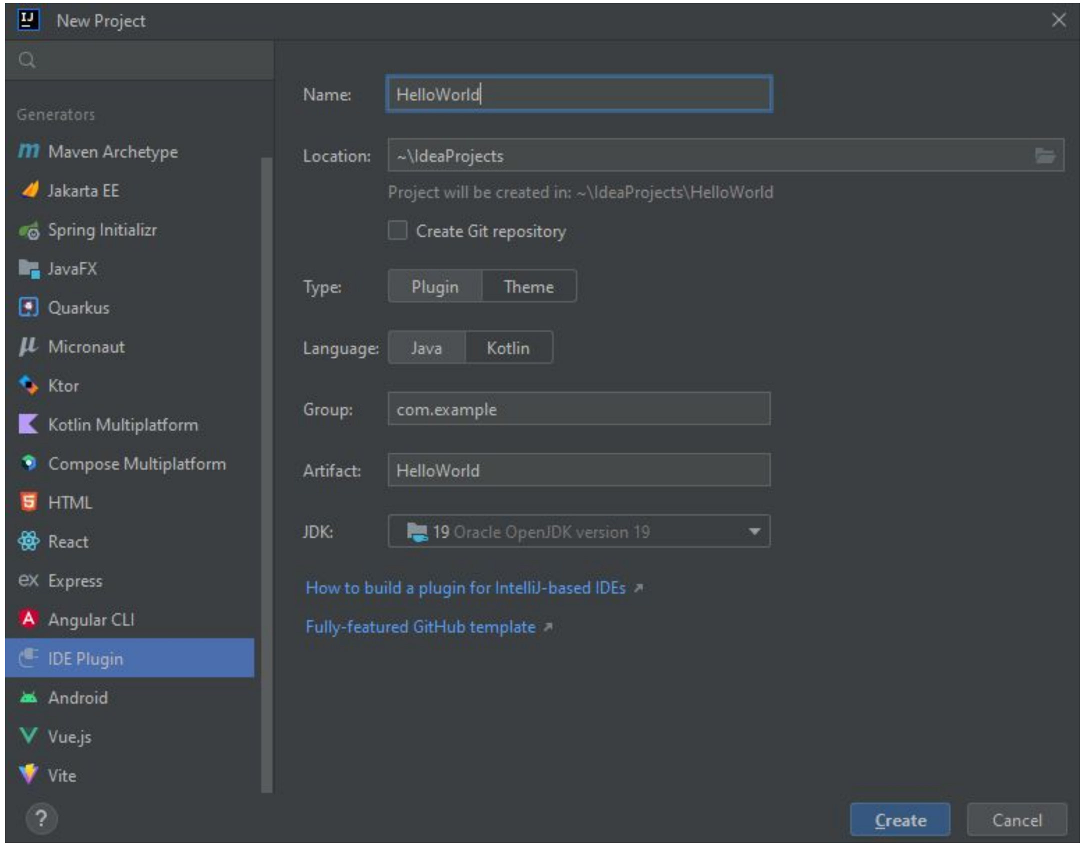
```java
public class MyToolWindowListener implements ToolWindowManagerListener {
    private Project project;

    public void MyToolwindowListener(Project project) {
        this.project = project;
    }

    @Override
    public void stateChanged(@NotNull ToolWindowManager toolWindowManager) {
        // Do something
    }
}
```



```xml
<projectListeners>
  <listener
          class="com.example.exampleplugin.MyToolWindowListener"
          topic="com.intellij.openapi.wm.ex.ToolWindowManagerListener"/>
</projectListeners>
```

Creating and testing a Plugin Project

Creating a Plugin Project is as simple as creating a new project and selecting IDE Plugin



Name: HelloWorld

Location: ~\IdeaProjects

Project will be created in: ~\IdeaProjects\HelloWorld

□ Create Git repository

Type: Plugin | Theme

Language: Java | Kotlin

Group: com.example

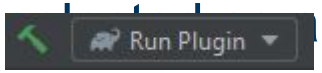
Artifact: HelloWorld

JDK: 19 Oracle OpenJDK version 19

How to build a plugin for IntelliJ-based IDEs ↗

Fully-featured GitHub template ↗

To test your plugin, make sure „Run Plugin“ is selected as a run configuration



Running the build will open a new IntelliJ instance with your plugin built in

Agenda

1. ~~Introduction IDE Plugins~~
2. ~~Organization~~
3. ~~IDE Plugin Structure~~
4. Your Task

Your Tasks

Tasks – For Today

Create a simple plugin which allows a user to create an annotation for a single method.

When a user selects a method, they should have the opportunity to press a button, which opens a textfield. The user then can provide a custom text which adds an annotation in front of the method. It does not matter how the action is performed (through a menu button, Keyboard shortcut, ...) You do not have to check if the text provided is a valid annotation.

Hint: To create an Annotation from text, you can use the call:

```
JavaPsiFacade.getElementFactory(Project).createAnnotationFromText(String, PsiElement);
```

Additionally display a warning for every method, that does not have an annotation.

Hint: `GetAnnotations()` returns a list of all annotations of a method

Questions