Developing plugins with IntelliJ

@RoboNovotny

EastCode Sessions

27. sep 2017

Why JetBrains?

- IntelliJ IDEA
- Android Studio
- WebStorm
- Ruby, PHP, .NET, Go, C/C++, Python, SQL

2500 plugins

Even more?

+ language support

+ framework integration

+ UI improvements

Getting started

1. install Plugin DevKit plugin
2. create a new IntelliJ Platform Plugin project
3. ???
4. PROFIT!

Getting started

- clone IntelliJ git repo
- set up as IntelliJ project
- use “ag” heavily

Getting started

- install IntelliJ IDEA Community Edition
- use it as the testbed

Doc & Resources

- internet tutorials
- IntelliJ Platform SDK Documentation
- Open API and Plugin Development forum
- source code

Dissecting Plugins

Plugin Structure

- manifest
- actions
- extensions
- services & components

Actions

Action Time!

- declare in manifest
- set title, icon, shortcut
- override actionPerformed()
- handle update events 2x / sec

Action Placement

- place action into arbitrary action group
- main menu
- context menu
- toolbar

Data Context

- active project
- active editor

+ other info about external state

Extension Points

Extensions

- autocomplete
- syntax highlighting
- code intentions & inspections
- version control
- code formatting
- preferences & configuration

Extensions

- declared in manifest
- override corresponding base class

Editor Improvements

Document

- text loaded into memory
- lightweight wrapper over stringish data

PSI / Program Structure Interface

- language-specific AST
- in Java: class / method / field
- in XML: element / attribute

PSI File

- obtained from action data context

or

- created from language-specific source

Threading Rules

- Swing rules apply
- VFS, PSI, project model shall be read from UI thread

UI

Swing-based UI

- Swing++++
- heavy improvements everywhere
- XML based layout manager and GUI designer

Libraries

Libraries

- no sophisticated library management
- drop thy libs into the libs
- everything gets packaged

Services

Services

- declared in manifest
- loaded on demand
- singleton
- three scopes:
  - application
  - project
  - module

Services

- service state may be automatically persisted between restart
  - implements PersistentStateComponent
  - annotate with @Store

Dependency Injection

DI

- constructor-based dependency injection
- based PicoContainer
- just declare the dependencies in the constructor

Services and DI

Old-school retrieval:

ServiceManager.getService(*serviceClass*)

Dependency Injection

- works for plugin components
- does not yet work for actions

Releasing

Building

- plugin builds to ZIP / JAR

Releasing

- install locally

or

- publish to official plugin repo
- wait for the approval

Best Practices (for me)

- use the source, Luke!
- mimic existing plugins
- read forum a lot

?