МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

**Лабораторная работа № 2 по дисциплине “Инструментальные средства разработки ПО”**

**“Создание плагинов для IDE”**

Выполнила студентка группы №М32111:

Елагин Дмитрий Андреевич

Проверил:

Приискалов Роман Андреевич

Санкт-Петербург

2021

Разработка плагина для IDE

**Задачи:**

1. Изучить архитектуру IDE
2. Наметить детальный план реализации
3. Разработка самого плагина

Плагин разрабатывался под IntelliJ IDEA

**Теоретические сведения**

Плагин – это единственный поддерживаемый путь расширения функциональности IDEA. Плагин использует предоставляемый программный интерфейс среды или других плагинов для реализации собственной функциональных возможностей. Обратим внимание на структуру и жизненный цикл плагина.

**Компоненты плагинов**

Компоненты – это фундаментальный концепт интеграции плагинов.   
Существуют три вида компонентов:

1. уровня приложения;
2. уровня проекта;
3. уровня модуля.

Компоненты уровня приложения создаются и инициализируются во время старта IntelliJ IDEA. Они могут быть получены от экземпляра класса Application с помощью метода getComponent(Class).

Компоненты уровня проекта создаются для каждого экземпляра класса Project (они могут быть созданы даже для неоткрытого проекта). Их можно получить от экземпляра Project вызовом метода getComponent(Class).

Компоненты уровня модуля создаются для каждого модуля в каждом проекте, загруженном в IDEA. Они могут быть получены аналогичным методом от экземпляра Module.

**Жизненный цикл компонентов**

**Компоненты загружаются в следующем порядке:**

•создание – выполнение конструктора;  
•инициализация – вызов метода initComponent (если реализован интерфейс ApplicationComponent);   
•конфигурация – вызов readExternal (если реализован JDOMExternalizable) или loadState (если реализован PersistentStateComponent и компонент в состоянии не по-умолчанию);   
•для компонентов уровня модуля – вызов moduleAdded (если реализован ModuleComponent); •для компонентов уровня проекта – projectOpened (если реализован интерфейс ProjectComponent).

**Компоненты выгружаются в следующем порядке:**•сохранение конфигурации – вызов writeExternal (если реализован интерфейс JDOMExternalizable) или getState (если реализован PersistentStateComponent);   
•освобождение ресурсов – вызов метода disposeComponent. В конструкторе компонента запрещено использовать метод getComponent() для получения каких-либо зависимостей. Если данному компоненту нужно получить другие компоненты при инициализации, они должны быть переданы в параметрах конструктора, либо перенести инициализацию в метод initComponent().

**Действия (Actions)**

Также Intellij IDEA предоставляет концепт действий (actions). Действие – это класс, наследуемый от AnAction, чей метод actionPerformed() вызывается, когда выбран элемент меню или кнопка тулбара. Действия объединяются в группы, которые также могут содержать вложенные группы. Группы действий могут быть отображены как меню или тулбары. Подгруппы отображаются как подменю.

**Сервисы**

Сервис – это компонент, загружаемый по требованию, когда плагин вызывает метод getService() класса ServiceManager. Intellij IDEA гарантирует, что будет создан только один экземпляр сервиса, независимо от того сколько раз был вызван метод.

Сервисы должны иметь интерфейс, определенный в plugin.xml. Класс с реализацией будет использован при создании сервиса.

**Plugin.xml**

Вся функциональность, предоставляемая плагином, а также описание и данные об авторе перечисляются в файле plugin.xml. Рассмотрим структуру файла, в котором присутствуют практически все значимые элементы.

Изображение выглядит как текст

Автоматически созданное описание

**Регистрация действий**

**Изображение выглядит как текст

Автоматически созданное описание**

**Создание интерфейса для действий**

Если плагину необходимо отобразить тулбар или всплывающее меню со специфическим интерфейсом это может быть выполнено с помощью классов ActionPopupMenu и ActionToolbar. Их объекты могут быть созданы вызовами метода ActionManager.createActionPopupMenu() и ActionManager.createActionToolbar().

Если действие содержится в специфическом компоненте (например, панели), обычно требуется вызвать ActionToolbar.setTargetComponent() и передать ему экземпляр компонента как параметр. Это гарантирует, что состояние кнопок тулбара зависит только от состояния связанного компонента, а не от фокуса на фрейме IDE.

**Содержимое плагинов**

Существуют три способа организации содержимого плагина. Первый – плагин содержит один jar-файл, размещенный в папке plugins. В архиве должен находиться конфигурационный файл (META-INF/plugin.xml) и классы, которые реализуют функциональность плагина. Конфигурационный файл определяет имя плагина, описание, данные о разработчике, поддерживаемая версия IDE, компоненты, действия, группы действий

**Сохранение состояния компонентов**

IntelliJ IDEA предоставляет API, позволяющий компонентам или сервисам сохранять их состояние между запусками IDE. Сохранять можно как примитивные типы, так и составные объекты благодаря использованию интерфейса PersistentStateComponent..

**Меню и панели инструментов**

Меню и панели инструментов (тулбары) строятся с использованием системы действий

**Диалоги**

Dialog wrapper— является базовым классом для всех модальных (и некоторых не модальных) диалоговых окон, использующихся в плагине IntelliJ IDEA. Он предоставляет следующие возможности:

* Расположение кнопок (платформо-специфичный порядок кнопок «ОК/Отмена», Mac OS-специфичная кнопка «Справка»);
* Контекстная помощь;
* Запоминание размера диалогового окна;
* Валидация данных (и отображение сообщения об ошибке, если данные, введенные в диалоговом окне некорректные);
* Сочетания клавиш: Esc для закрытия диалогового окна, влево/вправо для переключения между кнопками и Y/N для ответов «Да/нет» если они присутствуют в диалоговом окне;
* Необязательный флажок «Do not ask again».

При использовании класса DialogWrapper для собственного диалогового окна, необходимо выполнить следующие действия:

* Вызвать конструктор базового класса и передать проект в рамках которого будет отображаться диалоговое окно или родительский компонент для диалогового окна;
* Вызвать метод init() из конструктора класса диалогового окна;
* Вызвать метод setTitle(), чтобы задать заголовок для диалогового окна;
* Реализовать метод createCenterPanel() для возврата компонента, отвечающего за основное содержание диалогового окна;
* Необязательно: Переопределите метод getPreferredFocusedComponent(), возвращающий компонент, который окажется в фокусе, когда диалоговое окно отобразится;
* Дополнительно: Переопределите метод getDimensionServiceKey() для определения идентификатора, который будет использоваться для сохранения размера диалогового окна;
* Дополнительно: Переопределите метод getHelpId() для задания контекста справки, связанного с диалоговым окном.

Класс DialogWrapper часто используется совместно с формами UI Designer. Для того чтобы связать форму и ваш класс, расширяющий DialogWrapper, привяжите корневую панель формы к полю и возвратите его из метода createCenterPanel().  
  
Для отображения диалогового окна, вызовите метод show() и затем используйте метод getExitCode() для проверки, как было закрыто диалоговое окно.  
  
Для настройки кнопок, отображаемых в диалоговом окне (т.е. замены стандартного набора кнопок ОК/Отмена/Помощь), можно переопределить методы createActions() или createLeftActions(). Оба этих метода возвращают массив объектов Swing Action. Если кнопка, которую вы добавляете закрывает диалоговое окно, можно использовать DialogWrapperExitAction в качестве базового класса для действия.  
  
Чтобы проверить данные, введенные в диалоговом окне, можно переопределить метод doValidate(). Метод будет вызываться автоматически по таймеру. Если в настоящее время введенные данные допустимы, вам нужно возвратить null. В противном случае, возвратите объект ValidationInfo, который инкапсулирует сообщение об ошибке и, дополнительно, компонент, связанный с неправильными данными. Если указать компонент, рядом с ним будет отображаться значок ошибки и он получит фокус, когда пользователь попытается нажать кнопку ОК.

**Источники:**

* 1. <https://plugins.jetbrains.com/docs/intellij/getting-started.html?from=jetbrains.org#using-gradle>
  2. Индусы на ютубе
  3. https://habr.com/en/post/350648/

**Выполнение лабораторной работы**

Основная идея: плагин должен позволять быстро делиться кодом между разработчиками в рабочих чатах. Основная идея для расширения – дополнительное контекстное окно, которое позволяет перед отправкой задать пользователю комментарий и другую мета-информацию перед отправкой кода в беседу

**Код:**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Изображение выглядит как текст

Автоматически созданное описание**

**Вывод**

В данной лабораторной работе мы познакомились с архитектурой IDE, устройством плагинов и даже немного поучились их писать