Теоретический материал для индивидуального задания №10

JavaBeans

(продолжение)

Создание простого редактора свойств

Компонент также может предоставлять используемые контейнерным средством вспомогательные классы `PropertyEditor`. `PropertyEditor` является достаточно гибким интерфейсом, предоставляющим компоненту возможность сообщить контейнеру, как нужно отображать и редактировать значения свойств определенных типов.

Для свойств обычных типов, таких как строки, числа, шрифты и цвета, контейнер всегда предоставляет простые редакторы свойств. Однако если у вашего компонента есть свойство нестандартного типа, вы должны зарегистрировать для этого типа свой редактор свойств. Самый простой способ зарегистрировать редактор свойств состоит в использовании простого соглашения об именах. Если ваш тип определен в классе `X`, редактор для него должен быть определен в классе `XEditor`.

Другим возможным способом является регистрация редактора свойств путем вызова метода `PropertyEditorManager.registerEditor`(), например, из конструктора вашего класса `BeanInfo`. В случае если вы будете вызывать этот метод из самого компонента, компонент будет зависеть от класса редактора свойств, поэтому редактор будет привязан к компоненту при использовании того в приложениях, что является нежелательным. Другой способ регистрации редактора свойства состоит в использовании в классе `BeanInfo` объекта `PropertyDescriptor` с целью указания `PropertyEditor` для отдельного свойства. Класс `YesNoPanelBeanInfo`, например, делает это для свойства `messageText`.

Методы `PropertyEditor` позволяют вам определить три способа отображения значения свойства и два способа редактирования значения свойства пользователем. Значение свойства может быть показано:

В виде строки

Если вы определите метод `getAsText()`, контейнер сможет преобразовать свойство в строку и отобразить ее пользователю.

В виде перечислимого значения

Если свойство может принимать только значения из некоторого фиксированного набора, вы можете определить метод `getTags()`, что позволит контейнеру отобразить всплывающее меню с разрешенными для этого свойства значениями.

В графической форме

Если вы определите метод `paintValue()`, контейнер может попросить редактор свойств отобразить значение, используя некоторый естественный графический формат, такой как цветовой образец для цветов. Вам также будет нужно определить метод `isPaintable()` для указания того, что графический формат поддерживается.

Двумя методами редактирования являются:

Строковое редактирование

Если вы определите метод `setAsText()`, контейнер будет знать, что он может просто предоставить пользователю возможность ввести значение в текстовое поле и передать его методу `setAsText()`. Если в вашем редакторе свойств определен метод `getTags()`, в нем также должен быть определен метод `setAsText()`, чтобы контейнер смог установить значение свойства, используя отдельные значения тегов.

Пользовательское редактирование

Если в вашем редакторе свойств определен метод `getCustomEditor()`, контейнер может вызвать его для получения некоторого графического компонента, который может быть отображен в диалоговом окне и, будет служить в качестве пользовательского редактора для этого свойства. Вы также должны определить метод `supportsCustomEditor()`, чтобы указать, что пользовательское редактирование поддерживается.

Метод `setValue()` класса `PropertyEditor` вызывается для установки текущего значения свойства. Это то значение, которое методы `getAsText()` и `paintValue()` должны преобразовывать в строку или графическое представление.

Редактор свойств должен поддерживать список слушателей событий, которые интересуются изменениями значений свойства. Методы `addPropertyChangeListener()` и `removePropertyChangeListener()` являются стандартными методами регистрации и удаления слушателей событий. Когда редактор свойств изменяет значение свойства либо через `setAsText ()`, либо посредством пользовательского редактора, он должен послать событие `PropertyChangeEvent` всем зарегистрированным слушателям.

В интерфейсе `PropertyEditor` определен метод `getJavaInitializationString()` для использования контейнерньши инструментальными средствами, которые генерируют Java-код. Этот метод должен возвращать фрагмент Java-кода, который может инициализировать переменную, установив ее в текущее значение свойства.

И наконец, класс, реализующий интерфейс `PropertyEditor`, должен содержать конструктор без аргументов, чтобы контейнер мог динамически загружать его и создавать его экземпляры.

Большинство редакторов свойств могут быть гораздо проще, чем предлагается в нашем подробном описании. В большинстве случаев вы можете создавать классы, производные от `PropertyEditorSupport`, вместо непосредственной реализации интерфейса `PropertyEditor`. Этот полезный класс предоставляет пустые реализации для большинства методов `PropertyEditor`. Кроме этого он реализует методы для добавления и удаления слушателей событий.

Свойству, которое имеет перечислимые значения, необходим простой редактор свойств. Свойство `alignment` компонента `YesNoPanel` - достаточно типичный пример свойств этого типа. Свойство имеет только три допустимых значения, определенные в классе `Alignment`. Класс `AlignmentEditor`, показанный в примере 1, является редактором свойств, который сообщает контейнеру, как необходимо отображать и

редактировать значение этого свойства. Поскольку AlignmentEditor следует соглашению JavaBeans об именах, контейнер автоматически использует его для всех свойств с типом Alignment.

Создание сложного редактора свойств

В примере, у компонента YesNoPanel есть еще одно свойство, которому требуется редактор свойств. Свойство messageText может содержать многострочное сообщение, которое будет отображаться на панели. Этому свойству необходим редактор свойств из-за того, что контейнерная программа не отличает однострочные строковые типы от многострочных. Объекты TextField, которые она использует для ввода текста, не позволяют пользователю вводить несколько строк текста. По этой причине вы должны определить класс YesNoPanelMessageEditor и зарегистрировать его с помощью PropertyDescriptor для свойства message.

YesNoPanelMessageEditor представляет собой более сложный редактор, поддерживающий создание пользовательского компонента-редактора и графической формы представления значения. Обратите внимание, что в этом случае реализуется непосредственно PropertyEditor. Это означает, что должны обрабатываться регистрация и рассылка уведомлений объектам PropertyChangeListener. Метод getCustomEditor() возвращает компонент-редактор для многострочного текста. Метод paintValue() отображает значение свойства messageText. Это многострочное значение обычно не помещается в небольшом прямоугольном участке экрана, отведенном для этого свойства, поэтому paintValue() выводит инструкции для вызова пользовательского редактора, позволяющего видеть и редактировать значение свойства.

Создание настройщика компонентов

Компонент может пожелать предоставить пользователю контейнерной программы способ настройки его свойств помимо установки их по одному во время выполнения. Компонент может сделать это, создав для себя класс Customizer (настройщик) и зарегистрировав его при помощи объекта BeanDescriptor, возвращаемого его классом BeanInfo.

Настройщик должен быть графическим компонентом для отображения в диалоговом окне, созданном контейнером. Поэтому класс настройщика обычно является производным от класса Panel. Кроме этого настройщик должен реализовывать интерфейс Customizer. Этот интерфейс содержит методы, используемые для добавления и удаления слушателей событий, возникающих при изменении свойства, и метод setObject(), вызываемый контейнером для того, чтобы сообщить настройщику, какой объект он настраивает. Всякий раз, когда пользователь посредством настройщика делает изменения в компоненте, настройщик посылает событие PropertyChangeEvent всем заинтересованным в нем слушателям. И, как и у редактора свойств, в состав настройщика должен входить конструктор без аргументов, чтобы контейнер мог легко создать экземпляр объекта его типа.

В примере показан настройщик для компонента YesNoPanel. Этот настройщик выводит на экран панель, имеющую такое же расположение, как и YesNoPanel, но он заменяет объект TextArea на окно сообщений и три объекта TextField, соответствующие трем

кнопкам, которые отображаются в диалоговом окне. В эти текстовые области ользователь может вводить значения свойств `messageText`, `yesLabel`, `noLabel` и `cancelLabel`.

Демонстрационные примеры (в папке Примеры)

Дополнительный материал смотрите в:
- Java-УП-10-1.pdf
- главе 7 книги _Books\corejava2_2.djvu

Литература

1. Хабибуллин И. Ш. Java 7. — СПб.: БХВ-Питербург, 2012
2. Г. Шилдт. Java . Полное руководство, 8-е издание, М.: ООО “И.Д. Вильямс”,2012
3. Кей С. Хорстман. Java2 Основы. Том 1. С.-Питербург. 2007 (_Books\ corejava2_1.djvu)
4. Кей С. Хорстман. Java2 Тонкости программирования. Том 2. С.-Питербург. 2007 (_Books\ corejava2_2.djvu)
5. http://docs.oracle.com/javase/8/docs/