Теоретический материал для индивидуального задания №8

JavaBeans

Программный интерфейс JavaBeans предоставляет среду для разработки многократно используемых, встраиваемых, модульных программных компонентов. Спецификация JavaBeans дает следующее определение компонента (bean): «многократно используемый программный компонент, которым можно манипулировать в визуальных средах разработки». Это достаточно свободное определение; компоненты могут принимать самые разные формы. На самом простом уровне компонентами JavaBeans являются все отдельные графические компоненты, тогда как на гораздо более высоком уровне в качестве компонента может также функционировать встраиваемое графическое приложение. Однако большинство компонентов находятся где-то между этими двумя крайностями.

Одной из целей модели JavaBeans является обеспечение взаимодействия с аналогичными компонентными средами. Так, например, обычная Windows программа может с помощью соответствующего моста или компонента-обертки пользоваться компонентом Java так, как будто бы он является компонентом COM или ActiveX. Однако детали этого взаимодействия выходят за рамки этой темы.

Основы компонентов

Любой объект, удовлетворяющий определенным базовым правилам и соглашениям об именах, может считаться компонентом JavaBeans. Не существует никакого класса Bean, который все компоненты должны были бы иметь в качестве базового. Многие компоненты являются AWT-компонентами, но также возможно, и часто полезно, создавать «невидимые» компоненты, не имеющие видимого представления. (Однако то, что компонент не имеет видимого представления в конечном приложении, не означает, что им нельзя визуально манипулировать контейнерными средствами.)

Любой компонент экспортирует свойства, события и методы. Свойство (property) - это часть внутреннего состояния компонента, которую можно программно устанавливать и опрашивать, обычно через стандартную пару методов доступа get и set. Компонент может генерировать события (event) таким же образом, как AWT-компонент, например Button, генерирует события ActionEvent. Интерфейс JavaBeans использует ту же модель событий (фактически он определяет модель событий), которая используется в графических пользовательских интерфейсах Swing и AWT. Компонент определяет событие путем предоставления методов для добавления и удаления объектов-слушателей событий из списка слушателей, заинтересованных в данном событии. И, наконец, методы (methods), экспортируемые компонентом, являются простыми открытыми (public) методами, определенными в компоненте, за исключением методов, используемых для установки (set) и чтения (get) свойств и регистрации и удаления слушателей событий.

Кроме обычных типов свойств интерфейс JavaBeans обеспечивает поддержку индексных (indexed), связанных (bound) и ограниченных (constrained) свойств. **Индексное свойство** - это любое свойство, имеющее значение типа «массив» и для которого компонент предоставляет методы для извлечения и установки отдельных компонентов этого массива, а также методы для считывания и записи массива целиком. **Связанное свойство** - это свойство, которое рассылает уведомление при изменении

своего значения, тогда как **ограниченное свойство** - это то, которое рассылает уведомление при изменении своего значения и позволяет слушателям запретить это изменение.

Из-за того, что Java разрешает загружать классы динамически, контейнерные программы могут загружать неизвестные им классы. Контейнерные средства определяют поддерживаемые компонентом свойства, события и методы при помощи механизма интроспекции (introspection). Интроспекция основана на механизме отражения (reflection) java.lang.reflect, предназначенном для получения информации о членах класса. Кроме этого, компонент может предоставлять вспомогательный класс BeanInfo, поставляющий дополнительную информацию о компоненте. Класс BeanInfo предоставляет эту дополнительную информацию в виде нескольких объектов FeatureDescriptor, каждый из которых описывает какое-либо отдельное свойство компонента. У FeatureDescriptor есть несколько подклассов: BeanDescriptor, PropertyDescriptor, IndexedPropertyDescriptor, EventSetDescriptor, MethodDescriptor и ParameterDescriptor.

Основной задачей контейнерного приложения является предоставление пользователю возможности настройки компонента путем установки значений свойств. Контейнеры определяют редакторы свойств для широко используемых типов свойств, таких как числа, строки, шрифты и цвета. Однако, если у компонента есть свойство более сложного типа, для него может потребоваться создание класса ProprtyEditor, с помощью которого контейнер может предоставить пользователю возможность задать значение этого свойства.

Для сложных компонентов механизм настройки при помощи свойств, предоставляемый контейнером, может оказаться недостаточным. Такие компоненты могут определить класс Customizer, который создает графический интерфейс, позволяющий пользователю конфигурировать компонент некоторым удобным для него образом. Особенно сложные компоненты могут даже определять настройщики, которые выступают в качестве мастеров («wizard»), которые проводят пользователя через процесс пошаговой настройки.

Простой компонент

Все Swing- и AWT-компоненты могут функционировать как компоненты JavaBeans. Когда вы пишете пользовательский графический компонент, несложно сделать так, чтобы он мог действовать также в качестве JavaBeans-компонента. В *примере 1* показано определение пользовательского JavaBeans-компонента MultiLineLabel, отображающего одну или несколько строк статического текста. Это то, чего не может делать AWT-компонент Label.

Этот компонент является JavaBeans-компонентом из-за того, что у всех его свойств есть методы доступа get и set. Так как MultiLineLabel никак не реагирует на действия пользователя, он не определяет никаких событий, поэтому ему не нужны никакие методы для регистрации слушателей событий. Кроме этого, MultiLineLabel определяет конструктор без аргументов, поэтому его экземпляры могут быть легко созданы контейнером компонентов.

В MultiLineLabel для определения трех констант выравнивания используется вспомогательный класс с именем Alignment. В классе определены три константы,

содержащие экземпляры самого класса, и объявлен закрытый конструктор, поэтому никакие другие его экземпляры не могут быть созданы. Таким образом, Alignment эффективно создает перечислимый (enumerated) тип. (Эта полезная технология вовсе не является чем-то особенным для JavaBeans)

Упаковка компонента

Для того чтобы подготовить компонент к использованию в контейнере, вы должны упаковать его вместе с необходимыми ему файлами и ресурсами в файл JAR. Поскольку один компонент может иметь много вспомогательных файлов и поскольку файл JAR может содержать несколько компонентов, в манифесте JAR-файла должно быть описано, какие элементы файла являются компонентами. Файлы JAR создаются при помощи команды jar с ключом c. Если вместе с ключом c используется ключ m, jar считывает отдельный указанный вами файл манифеста, jar использует информацию из указанного вами отдельного файла манифеста при создании полного манифеста для файла JAR. Для указания того, что файл класса является компонентом, вам просто нужно в элемент манифеста файла добавить следующую строчку:

```
Java-Bean: true
```

Для упаковки класса MultiLineLabel в файл JAR, прежде всего, создайте «файл-заглушку» манифеста. Создайте файл, скажем, с именем manifest.stub и со следующим содержимым:

```
Name: MultiLineLabel.class
Java-Bean: true
```

Обратите внимание, что в системе Windows прямые слэши в файле манифеста не должны меняться на обратные. Формат файла манифеста JAR для разделения каталогов требует использования прямых слэшей независимо от платформы. Создав этот отдельный файл манифеста, вы можете теперь создать файл JAR:

```
% jar cfm MultiLineLabel.jar manifest.stub MultiLineLabel.class Alignment.class
```

Если данный компонент требует вспомогательных файлов, вы можете указать их вместе с файлами классов компонента в конце командной строки jar.

Установка компонента

В NetBeans вызываем команду меню Сервис\Палитра\Компоненты Swing/Awt. В появившемся диалоговом окне «Диспетчер Палитры» - добавляем компонент в категорию «Компоненты» или предварительно создаём новую категорию «Мои компоненты» куда будем помещать все свои компоненты.

Для добавления упакованного компонента нажимаем кнопку «Добавить из архива jar…» и далее следуем указаниям мастера установки.

Демонстрационные примеры (в папке Примеры)

Дополнительный материал в 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»,
3. 2012
4. Кей С. Хорстман. Java2 Основы. Том 1. С.-Питербург. 2007 (_Books\ corejava2_1.djvu)
5. Кей С. Хорстман. Java2 Тонкости программирования. Том 2. С.-Питербург. 2007 (_Books\ corejava2_1.djvu)
6. http://docs.oracle.com/javase/7/docs/