Теоретический материал для индивидуального задания №5

Графика и пользовательский интерфейс Java

Передача данных

Пакет `java.awt.datatransfer` предоставляет возможность передачи данных между приложениями и поддерживает метод обмена данными типа «вырезание и вставка» (cut-and-paste). Пакет `java.awt.dnd` поддерживает метод передачи данных типа «перетаскивание» (drag-and-drop).

Архитектура передачи данных

Важно понять архитектуру передачи данных. Класс `java.awt.datatransfer.DataFlavor`, является центральным; он представляет тип данных, подлежащих передаче. Каждый формат данных (data flavor) содержит удобочитаемое имя, объект `Class`, указывающий тип передаваемых данных, и тип MIME, определяющий кодировку, используемую при передаче данных.

В классе `DataFlavor` предопределена пара наиболее часто используемых форматов для передачи строк и списков объектов `File`. Кроме этого, в нем определено несколько типов MIME, используемых этими форматами. Например, `DataFlavor.stringFlavor` может передавать объекты `String` в виде текста в формате Unicode. Он содержит класс представления `java.lang.String` и тип MIME:

application/x-java-serialized-object; class=java.lang.String

Интерфейс `java.awt.datatransfer.Transferable` является еще одной важной частью механизма передачи данных. Этот интерфейс задает три метода, которые должны быть реализованы каждым объектом, желающим сделать свои данные доступными для передачи:

- `getTransferDataFlavor()` – возвращает массив всех типов `DataFlavor`, которые он может использовать для передачи своих данных;
- `isDataFlavorSupported()` - проверяет, поддерживает ли объект `Transferable` данный формат
- `getTransferData()` - возвращает данные в формате, соответствующем запрошенному `DataFlavor`.

Архитектура передачи данных основывается на механизме сериализации объектов как на одном из средств передачи данных между приложениями. Это означает, что архитектура передачи данных общая и гибкая. Она была разработана для того, чтобы обеспечить возможность обмена произвольными данными между независимыми виртуальными машинами Java и даже между приложениями Java и платформа-зависимыми приложениями. К сожалению, очевидно, что архитектура передачи данных не может быть реализована полностью, поэтому передача данных между машинами JVM и между JVM и «родным» приложением будет работать только в случае, если вы используете

специальные предопределенные форматы - `DataFlavor.stringFlavor` и `DataFlavor.javaFileListFlavor`. Хотя вы и можете определять другие объекты `DataFlavor`, представляющие другие сериализуемые классы Java, эти пользовательские пакеты будут работать только внутри одной JVM.

Простое копирование и вставка

В то время как `DataFlavor` и `Transferable` предоставляют базовую инфраструктуру передачи данных, класс `java.awt.datatransfer.Clipboard` и интерфейс `java.awt.datatransfer.ClipboardOwner` поддерживают обмен данными в стиле «вырезание и вставка». Типичный сценарий вырезания и вставки работает примерно так:

- Когда пользователь дает команду скопировать или вырезать что-либо, активное приложение, прежде всего, получает системный объект `Clipboard`, используя вызов `getSystemClipboard()` объекта `java.awt.Toolkit`. Затем приложение создает объект `Transferable`, представляющий передаваемые данные. И, наконец, оно передает этот объект буферу обмена с помощью вызова метода `setContents()` этого буфера. Активное приложение, кроме этого, должно передать в `setContents()` объект, реализующий интерфейс `ClipboardOwner`. Сделав это, приложение становится владельцем буфера обмена и должно поддерживать свой объект `Transferable` до тех пор, пока не перестанет быть владельцем.

- Когда пользователь дает команду вставки, целевое приложение, прежде всего, получает системный объект `Clipboard` таким же образом, как это делало активное приложение. Затем оно вызывает метод `getContents()` системного буфера обмена для получения хранящегося там объекта `Transferable`. Теперь оно может использовать методы, определяемые интерфейсом `Transferable` для выбора объекта `DataFlavor`, используемого для обмена данными, и фактически выполнить прием данных.

- Когда пользователь копирует или вырезает другой фрагмент данных, инициируется новая передача данных, и новое активное приложение (оно может быть тем же) становится новым владельцем буфера обмена. Предыдущий владелец уведомляется о том, что он больше не является владельцем буфера обмена, когда система вызывает метод `lostOwnership()` объекта `ClipboardOwner`, указанного при начальном вызове метода `setContents()`.

Обратите внимание на то, что ненадежным алплетам не разрешается работать с системным буфером обмена из-за того, что там могут находиться важные данные других приложений. Это значит, что алплеты не могут принимать участие в обмене данными типа «вырезание и вставка» между приложениями. Вместо этого алплеты должны создавать свой закрытый буфер обмена для использования обмена данными между апплетами.

Пример 1 представляет собой простую программу на базе AWT, демонстрирующую поддержку возможностей механизма вырезания и вставки в приложении. Она полагается на предопределенный объект `DataFlavor.stringFlavor` и класс `StringSelection`, реализующий интерфейс `Transferable` для строковых данных. Кроме этого, в программе использован объект `DataFlavor.javaFileListFlavor` для разрешения вставки имен файлов. Обратите внимание, что программа выполняет операции копирования, а не вырезания. Для того чтобы сделать вырезание, просто удалите текст из источника после его копирования в буфер обмена. Довольно

интересными являются методы `copy()` и `paste()`, расположенные ближе к концу примера. Заметьте также, что пример реализует интерфейс `ClipboardOwner`, поэтому он получает уведомление, когда его данные больше не находятся в буфере обмена. Лучший способ протестировать данную программу - запустить две ее независимые копии и передать данные между ними. Вы также можете попытаться передать данные между экземпляром данной программы и родным для вашей платформы приложением.

Тип данных Transferable

Класс `java.awt.datatransfer.StringSelection` упрощает обмен строковыми значениями между приложениями. Однако для передачи других типов данных вы должны создать собственную реализацию интерфейса `Transferable`. В *примере 2* показан класс `Scribble` - тип данных, представляющий набор отрезков линий. Он реализует интерфейс `Shape`, поэтому может быть отображен с помощью программного интерфейса `Java 2D`. Но наиболее важно то, что он реализует интерфейс `Transferable`, поэтому эти рисунки (отрезки линий, scribbles) могут передаваться от одного приложения другому.

Класс `Scribble` фактически поддерживает передачу данных с помощью двух форматов данных. Он определяет пользовательский формат, передающий сериализованные экземпляры `Scribble`. Однако, как было отмечено ранее, такие пользовательскиео форматы не могут применяться для передачи данных между независимыми виртуальными машинами. Следовательно, класс `Scribble` позволяет передавать себя также и в виде объекта `String` и определяет методы для преобразования рисунков (отрезков линий) в формат строки и обратно. Заметим, что в примере 2 не демонстрируется передача данных методами «вырезание и вставка» или «перетаскивание», в нем просто определяется инфраструктура передачи данных, используемая в последующих примерах, иллюстрирующих механизмы вырезания и вставки, а также перетаскивания. Он также заслуживает изучения как пользовательская реализация `Shape`.

Перетаскивание рисунков

Начиная с Java 1.2, была введена поддержка обмена данными методом «перетаскивание» (drag-and-drop). Программный интерфейс для этого механизма находится в пакете `java.awt.dnd` и основан на той же архитектуре `DataFlavor` и `Transferable`, что и механизм вырезания и вставки. В *примере 2* приведена программа, позволяющая пользователю создать изображение в режиме «рисование» и перетащить его в режиме «перетаскивание».

Программный интерфейс для механизма перетаскивания значительно сложнее интерфейса вырезания и вставки, что отразилось на длине этого примера. Ключевыми интерфейсами являются `DragGestureListener`, который начинает новое перемещение, а также `DragSourceListener` и `DropTargetListener`, которые уведомляют источник и приемник перетаскивания о некоторых важных событиях, возникающих во время перетаскивания. В примере реализованы все три интерфейса.

Во время изучения данного примера вам следует уделить особое внимание методам `dragGestureRecognized()`, указывающему место инициализации операции перетаскивания, и `drop()`, обозначающему действительное место передачи данных от

источника приемнику. В этом примере используется список `List` объектов `Scribble`, каждый из которых представляет набор соединенных друг с другом отрезков линий.

Демонстрационные примеры (в папке Примеры)

Для запуска примера используйте команду go.cmd в каталоге примера (требуется java.exe, возможно нужно скорректировать переменную PATH)

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/8/docs/