Управление Java classpath (Windows)

Сlasspath представляет собой связующее звено между исполняемым модулем Java и файловой системой. Он описывает, где компилятор и интерпретатор ищут пакет файлов .class для загрузки. Основная идея заключается в том, что иерархия файловой системы отражает иерархию пакета Java, а classpath указывает, какие директории в файловой системе являются корневыми для иерархии пакета Java.

К сожалению, файловые системы в высшей степени сложны, зависят от платформы, и не очень хорошо согласуются с пакетами Java. Данное утверждение в большей степени справедливо по отношению к Windows. Java спроектирован хакерами UNIX, а это означает, мягко говоря, далеко не идеальную синхронизацию с соглашениями Windows. Следовательно, classpath в течение многих лет был источником постоянного раздражения, как у новых пользователей, так и у опытных Java-программистов. Classpath - далеко не самая приятная часть платформы Java, а скорее раздражающая помеха, которая вынуждает вас задерживаться на работе в попытках устранить маленькую проблему, которая упорно не хочет решаться.

Качественная IDE, например Eclipse, может оградить Вас от некоторых трудностей управления classpath, но только частично, и только до тех пор, пока все гладко (а что-то всегда оказывается наперекосяк). Следовательно, совершенно необходимо, чтобы каждый программист на Java понимал classpath в полном объеме. Только при глубоком понимании можно надеяться на разрешение проблем, возникающих из-за classpath.

В данной статье описано все, что Вам необходимо знать о Java classpath (и сопутствующем sourcepath) в Windows. В [соответствующей статье](http://www.ibm.com/developerworks/ru/library/j-classpath-unix/) продемонстрированы аналогичные методы для UNIX и Mac OS X. Соблюдение указанного алгоритма должно не только предотвратить возникновение лишних трудностей с classpath, но и устранить большинство возникающих проблем.

Структура пакета

Освоение classpath начинается с исходного кода. Каждый класс принадлежит пакету, а этот пакет *должен* соответствовать стандартным соглашениям по именованию. Вкратце: имя пакета начинается с двухуровневого реверсированного имени домена, например com.example или edu.poly. За ним следует как минимум еще одно слово, которое описывает содержимое пакета. Например, так как у меня есть имя домена elharo.com, то если бы я писал класс Fraction, то мог бы разместить его в одном из следующих пакетов:

* com.elharo.math
* com.elharo.numbers
* com.elharo.math.algebra.fields

После реверсированного имени домена используйте только имена подпакетов, состоящие из одного слова. Не используйте аббревиатуры и следите за правописанием слов. При необходимости используйте программы проверки орфографии. Самое большое число проблем, связанных с classpath, вызваны использованием в исходном коде одного слова, а в файловой системе - аббревиатуры этого слова, или того же самого слова, но с незначительно отличающейся орфографией. Единственным разумным решением является постоянное использование правильно написанных несокращенных имен.

Полное имя пакета должно состоять из строчных букв, даже для имен собственных и аббревиатур, которые обычно пишутся заглавными буквами. На самом деле Windows не различает заглавные и строчные буквы, используемые в именах файлов, в отличие от Java и некоторых файловых систем UNIX. Зависимость от регистра - это верный путь к возникновению проблем при перемещении файлов из одной системы в другую.

Имя пакета должно состоять исключительно из символов ASCII. Хотя компилятор и принимает написание имен пакетов на иврите, кириллице, греческом и других шрифтах, но многие файловые системы этого не допускают и, как Вы вскоре увидите, эти имена пакетов будут иметь двойное назначение в качестве имен директорий. Хотя имена классов и пакетов Java и пишутся в Unicode, многие файловые системы, включая FAT, пока не работают с Unicode. Печально, но в этом списке все еще присутствует много файловых систем FAT. Простое копирование файла в систему с другой кодировкой, заданной по умолчанию, может помешать компилятору и интерпретатору при поиске правильных классов.

**Одноразовая программа**

Если Вы пишете один единственный класс, который будете использовать только один раз, и больше он Вам не понадобится, например, класс, создаваемый для проверки своих знаний API - то Вам не надо помещать его в пакет. Однако если какой-либо класс будет использоваться более одного раза, то он должен находиться в пакете.

Не экономьте на имени пакета, а то это приведет к катастрофе! Если Вам необходимо имя домена, то купите его. Если имена слишком длинные, то купите покороче. (Однажды я купил xom.nu, следовательно, префикс моего пакета состоял всего лишь из шести букв.) Не помещайте свои классы в пакете, заданном по умолчанию (в том пакете, который Вы получаете, если не включили выражение пакета в класс). Если доступ к пакету не позволяет объектам взаимодействовать, добавьте в классы больше общих методов. Каждый класс, который Вы используете более одного раза, должен находиться в пакете.

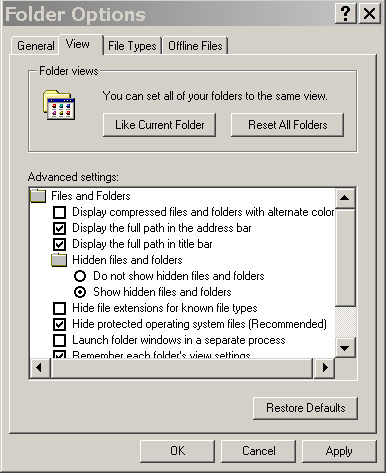
Настройка конфигурации Windows

Расширения файлов и пути доступа очень важны для Java и Windows. Однако прежде чем перейти к следующему этапу, убедитесь, что Вы можете их посмотреть. Скрытие части имени файла может быть приемлемым для конечного пользователя (хотя у меня есть сомнения на этот счет), но это совершенно не годится для разработчика. Чтобы это исправить, необходимо изменить некоторые заданные по умолчанию настройки в Windows Explorer.

Сначала откройте папку на рабочем столе Windows, не важно какую. Зайдите в меню Сервис(Tools) и выберите Свойства папки (Folder Options). В открывшемся диалоговом окне убедитесь, что отмечены три следующих опции, как показано на рисунке 1:

* "Выводить полный путь в панели адреса (Display the full path in the address bar)" должно быть отмечено.
* "Выводить полный путь в строке заголовка (Display the full path in title bar)" должно быть отмечено.
* "Скрывать расширения для зарегистрированных типов файлов(Hide file extensions for known file types)" должно быть *не отмечено*.

Рисунок 1. Свойства Windows Explorer



Возможно, появится желание отметить и пункт "Показывать скрытые файлы и папки (Show hidden files and folders)." Это не значительно повлияет на Вашу работу с Java, но лично я предпочитаю видеть, с чем я работаю. Настройка данных свойств предоставляет более подробную информацию о том, что и где Вы делаете, и помогает гораздо легче устранять любые потенциальные проблемы.

Структура директории

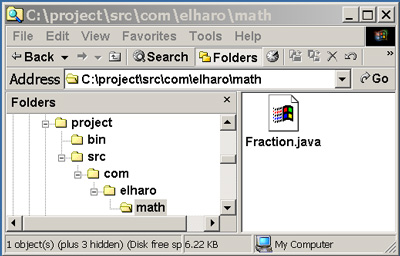
Следующим этапом является организация Ваших исходных файлов в соответствии со структурой пакета. Создайте где-нибудь пустую директорию. Для данной статьи я назову ее *project*. Обычно гораздо легче разместить ее в корневом каталоге, например C:\project. Можно разместить ее и на рабочем столе или в папке Документы и Настройки (Documents and Settings), но только в случае необходимости, поскольку это приведет к тому, что необходимые команды станут более длинными и сложными. (Возможно, у Вас не будет выбора, если Вы работаете в Windows XP или более поздними версиями и не имеете прав администратора. В однопользовательской системе гораздо легче работать при наличии прав администратора.)

Что бы Вы не делали, не сохраняйте данную директорию (или любую другую) в Вашей папке JDK (например, C:\jdk1.6.0 или C:\Program Files\Java\j2sdk1.5.0\_09). После первой инсталляции директории JDK и JRE должны остаться нетронутыми.

Внутри директории project создайте еще две директории: bin и src. (Некоторые называют их build и source, соответственно.)

Затем, в директории src создайте иерархию, которая будет отражать иерархию Вашего пакета. Например, при наличии класса, именуемого com.elharo.math.Fraction, я бы разместил директорию com в директории src. Потом в директории com я бы создал директорию elharo. Затем директорию math в директории elharo. И, наконец, поместил бы Fraction.java внутри данной директории math, как показано на рисунке 2:

Рисунок 2. Структура директорий, повторяющая структуру пакета



Очень важно: Никогда не размещайте в своей директории src ничего, кроме исходного кода. Обычно в ней находятся только .java файлы. Иногда в данной директории можно разместить .html файлы (для JavaDoc) или другие типы файлов исходного кода. Но никогда не помещайте в данной структуре .class файлы или другие скомпилированные, сгенерированные артефакты. Это прямой путь к катастрофе. К сожалению, если Вы потеряете бдительность, то компилятор javac сделает именно это. В следующем разделе я покажу, как это преодолеть.

Компиляция

Компиляция Java-кода - это весьма каверзная процедура, поскольку Вам необходимо следить за ходом нескольких связанных, но отличных друг от друга вещей:

**Обратные и прямые слэши**

Программисты UNIX, которые изобрели язык Java, позволили компилятору признавать прямые слэши, характерные для UNIX, вместо используемых в Windows обратных слэшей. Следовательно, следующая команда будет также работать:

Однако некоторые команды не допускают использования прямых слэшей и требуют обратные. При работе в Windows проще всего всегда использовать обратные слэши.

* Целевой файл, который Вы компилируете.
* Директория, где компилятор ищет .java файлы, импортируемые целевым файлом.
* Директория, где компилятор ищет .class файлы, импортируемые целевым файлом.
* Директория, куда компилятор помещает скомпилированный результат.

По умолчанию компилятор javac считает, что все они являются текущей рабочей директорией, что почти никогда не соответствует Вашим нуждам. Следовательно, при компиляции Вам необходимо точно указать каждый из этих элементов.

Файл для компилирования

Первое, что Вы указываете, это .java файл, который Вы собираетесь скомпилировать. Необходимо указать путь к данному файлу из текущей рабочей директории. Например, вы находитесь в директории project, показанной на [рисунке 1](https://www.ibm.com/developerworks/ru/library/j-classpath-windows/index.html#fig1). Данная директория содержит директорию src. Директория src содержит директорию com, в которой, в свою очередь, находится директория example, а в ней хранится файл Fraction.java. Компиляция выполняется следующей командной строкой:

|  |  |
| --- | --- |
| 1 | C:\project> javac src\com\elharo\math\Fraction.java |

Если путь является неверным, то Вы получите сообщение об ошибке такого типа:

|  |  |
| --- | --- |
| 1 | error: cannot read: src\com\example\mtah\Fraction.java |

Если у Вас появилось сообщение об ошибке, то проверьте правильность указания путь. В данном случае, ошибка говорит о том, что переставлены местами вторая и третья буквы в "math".

Если Вы не нашли орфографической ошибки, то убедитесь, что файл действительно находится там, где он должен быть, выполнив команду dir, как показано ниже:

|  |  |
| --- | --- |
| 1  2 | C:\project\src> dir src\com\example\math  ls: src/com/example/math: No such file or directory |

Данное сообщение обычно указывает на опечатку в указании пути, но она также может означать, что Вы находитесь не в той директории, в которой предполагаете. В данном примере Вам необходимо удостовериться, что текущей рабочей директорией является именно директория project. Проверьте текст между C: и > в командной строке, чтобы удостовериться, что Вы находитесь там, где должны быть. В данном примере Вы увидите, что в действительности я нахожусь в C:\project\src, хотя должен быть в C:\project.

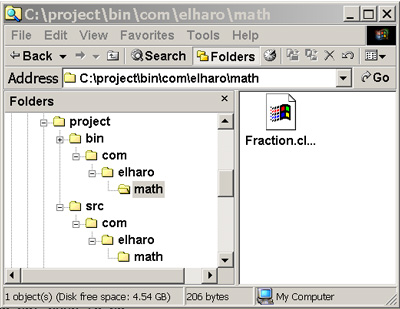
Куда отправляется результат

Если предположить, что нет никаких синтаксических ошибок, то javac разместит скомпилированный .class файл в той же директории, что и соответствующий .java файл. Вам это не нужно. Размещение .class и .java файлов в одной директории создает большие трудности при очистке скомпилированных файлов без случайного удаления .java файлов, которые Вы бы хотели сохранить. Это создает проблемы при создании чистых сборок и зачастую приводит к проблеме появления версий, а также усложняет согласование только что скомпилированных .class файлов при распределении двоичных файлов. Вам необходимо указать компилятору, чтобы он поместил скомпилированный результат в совершенно другую директорию. Параметр -d указывает конечную директорию (обычно называемую bin, build или classes):

|  |  |
| --- | --- |
| 1 | C:\project> javac -d bin src\com\elharo\math\Fraction.java |

Теперь результат выглядит так, как показано на рисунке 3. Обратите внимание, что javac создал полную иерархию директорий com\elharo\math. Вам не надо делать этого вручную.

Рисунок 3. Параллельные исходные и скомпилированные иерархии



Sourcepath

Директория, в которой Java ищет исходные файлы, называется *sourcepath*. В показанной здесь схеме это директория src. Это директория, которая содержит *иерархию* исходных файлов, расположенных в своих собственных директориях. Это *не* директория com или src\com\elharo\math.

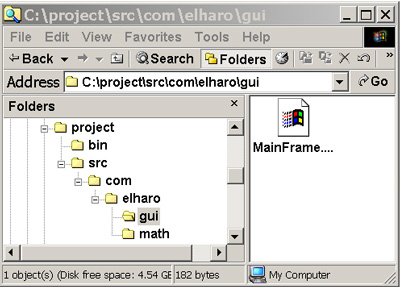
Большинство проектов использует более одного класса и более одного пакета. Они связаны с помощью операторов import и полностью пакетно-классифицированных имен класса. Например, предположим, что Вы создаете новый класс MainFrame в пакете com.elharo.gui, как показано в листинге 1:

Листинг 1. Класс в одном пакете может импортировать класс в другом пакете

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | package com.elharo.gui;    import com.elharo.math.\*;    public class MainFrame {      public static void main(String[] args) {      Fraction f = new Fraction();      // ...    }    } |

Этот класс использует класс com.elharo.math.Fraction в пакете, отличном от класса MainFrame. Исходная настройка показана на рисунке 4. (я удалил скомпилированный результат, полученный на предыдущем этапе, так как всегда могу скомпилировать его снова.)

Рисунок 4. Исходная структура для нескольких пакетов



Теперь давайте посмотрим, что случится, если я попытаюсь скомпилировать MainFrame.java так, как я делал это ранее.

Листинг 2. Компиляция MainFrame.java

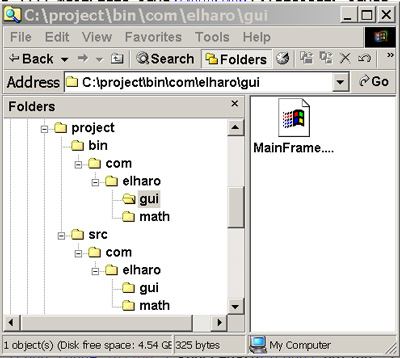
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | C:\project> javac -d bin src\com\elharo\gui\MainFrame.java  src\com\elharo\gui\MainFrame.java:3: package com.elharo.math does not exist  import com.elharo.math.\*;  ^  src\com\elharo\gui\MainFrame.java:7: cannot find symbol  symbol  : class Fraction  location: class com.elharo.gui.MainFrame    private Fraction f = new Fraction();            ^  src\com\elharo\gui\MainFrame.java:7: cannot find symbol  symbol  : class Fraction  location: class com.elharo.gui.MainFrame    private Fraction f = new Fraction();                             ^  3 errors |

Ошибки в листинге 2 произошли из-за того, что хотя javac и знал, где найти MainFrame.java, но он не знал, где находится Fraction.java. (Вы думаете, что было бы вполне разумно заметить совпадающие иерархии пакета, но это не так.) Чтобы прояснить это, мне нужно задать значение *sourcepath*, которое указывает директории, в которых компилятор должен искать иерархию исходных файлов. В листинге 2, это src. Следовательно, я использую -sourcepath вот таким образом:

|  |  |
| --- | --- |
| 1 | C:\project> javac -d bin -sourcepath src src\com\elharo\gui\MainFrame.java |

Теперь программа компилируется без ошибок и выдает результат, показанный на рисунке 5. Обратите внимание, что javac также скомпилировал файл Fraction.java, на который ссылается скомпилированный мной файл.

Рисунок 5. Мультиклассовый результат



Компиляция нескольких директорий в sourcepath

В Вашем sourcepath может быть несколько директорий, отделенных точкой с запятой, хотя обычно в этом нет необходимости. Например, если Вы хотите включить как локальную директорию src, так и директорию C:\Projects\XOM\src, где хранится исходный код для другого проекта, то можно скомпилировать следующее:

|  |  |
| --- | --- |
| 1  2 | C:\project> javac -d bin -sourcepath src;C:\Projects\XOM\src    src/com/elharo/gui/MainFrame.java |

Данная команда не компилирует каждый файл, найденный в какой-либо из этих иерархий. Она компилирует только файлы, на которые прямо или косвенно ссылается единичный файл .java, который я и попросил скомпилировать.

Пробелы в именах директорий

Ни имена классов Java, ни имена пакетов не могут содержать пробелов. Однако иногда директория, содержащая исходный файл или директорию пакета Java все-таки содержит пробел. Самым наглядным примером служит Documents and Settings. Если Вам необходимо включить одну из директорий в путь, то необходимо указать соответствующий аргумент командной строки в двойных кавычках. Например, если Вы находитесь в корневой директории C:, а папка src находится в C:\Documents and Settings\Administrator\project, то необходимо выполнять компиляцию следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | C:\> javac -d bin -   sourcepath "C:\Documents and Settings\    Administrator\project"   -classpath C:\lib\classes   "C:\Documents and Settings\   Administrator\project\src\com\   elharo\gui\MainFrame.java" |

В большинстве случаев этого можно избежать, перейдя в директорию project, до компиляции или запуска программы.

Гораздо чаще у Вас будет единственная исходная директория для .java файлов, и различные директории для классов или JAR-архивов, где размещаются предварительно скомпилированные сторонние библиотеки. В этом и заключается роль classpath.

Настройка classpath

В больших проектах повторная компиляция каждого файла может быть достаточно затратной с точки зрения времени. Вы можете облегчить нагрузку путем раздельного компилирования и хранения отдельных частей одного и того же проекта в различных директориях bin. Данные директории добавляются в classpath.

Существует несколько способов добавления класса в classpath. Однако следует использовать только переключатель командной строки -classpath. Например, предположим, что я хочу импортировать файлы из другого проекта, которые я предварительно скомпилировал в директории C:\lib\classes. Тогда я добавлю -classpath C:\lib\classes в командную строку следующим образом:

|  |  |
| --- | --- |
| 1  2 | C:\project> javac -d bin -sourcepath src -classpath C:\lib\classes    src\com\elharo\gui\MainFrame.java |

Теперь предположим, что мне необходимо добавить две директории, C:\project1\classes и C:\project2\classes. Тогда укажем их, разделив точкой с запятой следующим образом:

|  |  |
| --- | --- |
| 1  2  3 | C:\project> javac -d bin -sourcepath src    -classpath C:\project1\classes;C:\project2\classes    src\com\elharo\gui\MainFrame.java |

Конечно, при желании Вы можете использовать различные варианты относительного пути. Например, если project1 и project2 являются одноуровневыми узлами текущей рабочей директории (то есть, у них одна и та же родительская директория), то можно указать их следующим образом:

|  |  |
| --- | --- |
| 1  2  3 | C:\project> javac -d bin -sourcepath src    -classpath ..\project1\classes;..\project2\classes    src\com\elharo\gui\MainFrame.java |

До сих пор я предполагал, что программа является сама по себе полной и не использует никаких отдельно скомпилированных сторонних библиотек. А если использует, то Вам потребуется также указать их в classpath. Обычно библиотеки классифицируются как JAR-файлы, например, junit.jar или icu4j.jar. В данном случае, в classpath Вы указываете непосредственно JAR-файл, а не содержащую их директорию. (В сущности, JAR-файл работает как директория, содержащая скомпилированные .class файлы.) Например, следующая команда добавляет в classpath директорию C:\classes, файл icu4j.jar в текущей рабочей директории, и файл junit.jar in E:\lib:

|  |  |
| --- | --- |
| 1  2  3 | C:\project> javac -d bin -sourcepath src    -classpath C:\classes;icu4j.jar;E:\lib\junit.jar    src\com\elharo\gui\MainFrame.java |

JAR-файлы используются только для .class файлов и classpath, а не для .java файлов и sourcepath.

**Директории верхнего уровня**

Обратите внимание, что все упоминаемые здесь директории являются директориями верхнего уровня, содержащими иерархию пакета. Директории, имена которых совпадают с именами пакетов (com, elharo, math и т.д.) никогда не указываются в sourcepath или classpath напрямую.

Запуск программы

Итак, теперь Вы успешно скомпилировали программу и готовы ее запустить. Это похоже на компиляцию, но гораздо проще. При запуске программы необходимо указать только:

* Сlasspath.
* Полное пакетно-классифицированное имя класса, содержащего Ваш метод main() .

Не нужно указывать sourcepath.

Обычно classpath является тем же самым classpath, который Вы использовали для компиляции программы, с добавлением директории, где был размещен скомпилированный результат. Например, если команда компиляции была следующей:

|  |  |
| --- | --- |
| 1  2  3 | C:\project> javac -d bin -sourcepath src    -classpath C:\classes;E:\lib\junit.jar    src\com\elharo\gui\MainFrame.java |

а метод main() находился в классе the class com.elharo.gui.MainFrame, то необходимо запустить программу следующим образом:

|  |  |
| --- | --- |
| 1  2 | C:\project> java -classpath bin;C:\classes;E:\lib\junit.jar      com.elharo.gui.MainFrame |

Обратите особое внимание на то, что последний элемент командной строки является *именем класса*, а не именем файла. Оно не заканчивается на .java или .class. Данный класс должен быть найден где-нибудь в classpath.

Другие места хранения классов

Я настоятельно рекомендую Вам всегда четко указывать classpath, и при компиляции, и при запуске. Есть и другие места, где Вы можете разместить файлы, так чтобы они добавлялись в classpath и находились как javac компилятором, так и интерпретатором java. Однако данные варианты сокращают объем набираемой с клавиатуры информации и делают это за счет большого количества отладок, когда - а не если - Вы случайно поместите в classpath старую версию класса.

В данном разделе я покажу Вам несколько мест, где можно найти скрытые файлы, которые внезапно появляются в Вашем classpath и создают проблемы. Чаще всего это происходит на тех компьютерах, которые Вы не контролируете, например на сервере.

Текущая рабочая директория

Компилятор всегда вносит текущую рабочую директорию (.) в classpath, вне зависимости от того, просите ли Вы об этом в явной форме или нет. Слишком легко забыть, что находится или не находится в той же директории, что и Вы. Однако старайтесь избегать размещения каких-либо классов или иерархий в своих директориях project или home. Вместо этого, всегда аккуратно храните все для .java файлов в директориях src и для .class файлов - в директориях bin.

CLASSPATH

Через некоторое время Вы устанете от набора вручную директорий bin и JAR-архивов. Используйте переменную среды CLASSPATH, в которой можно один раз указать директории и JAR-архивы. Тогда Вам не придется при каждом запуске javac или java указывать их пути.

Не поддавайтесь искушению, так как это может привести к возникновению проблем при загрузке неверного класса или неверной версии класса. В итоге Вы потратите в сотни раз больше времени на устранение неполадок, вызванных случайной загрузкой неверных классов, чем сэкономите на наборе. Существуют и лучшие способы автоматизации classpaths и избавления от ввода с клавиатуры.

jre\lib\ext

JAR-архивы, размещенные в директории jre\lib\ext, добавляются в classpath всех приложений, запущенных на данной виртуальной машине. Хотя это и кажется удобным, но также является и ошибкой, аналогичной добавлению директорий в переменную среды CLASSPATH. Рано или поздно (а скорее всего рано), Вы загрузите неверную версию класса из такого места, о котором Вы даже и не думали, и потратите долгие часы на отладку.

Эта проблема может оказаться особенно серьезной при развертывании серверных приложений. Будьте особенно внимательны к тому, чтобы на том сервере, на котором Вы разворачиваете, не было каких-либо дополнительных JAR в директории jre\lib\ext. Проблемы, вызванные неверной версией JAR-архива в classpath, могут оказаться очень сложными с точки зрения устранения, если Вы не распознаете симптомы и не будете знать, что искать. Во избежание таких проблем, некоторые среды разработки продвинулись так далеко, как написание своих собственных загрузчиков класса, которые обходят обычные загрузочные механизмы класса кода Java.

jre\lib\endorsed

JAR-файлы в директории jre\lib\endorsed также добавляются в classpath всех приложений, работающих на данной виртуальной машине. Отличие заключается в том, что данные файлы фактически добавляются скорее в*bootclasspath*, чем в обычный classpath и могут заменить стандартные классы, имеющиеся в JDK. Данный подход оказывается особенно полезным для модернизации анализатора XML и устранения неполадок в VM.

Еще раз повторю, что хотя и данный подход и кажется удобным, но по этой же причине представляет собой достаточно длительную с точки зрения устранения ошибку. Если Вам необходимо заменить классы JDK, то при запуске используйте опцию -Xbootclasspath/p во избежание случайной загрузки неверной версии класса:

|  |  |
| --- | --- |
| 1  2 | C:\project> java -classpath C:\classes         -Xbootclasspath/p:xercesImpl.jar com.elharo.gui.MainFrame |

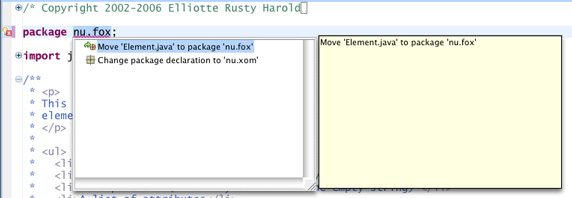
Автоматизация управления classpath

Вы должны научиться пользоваться обычным молотком, прежде чем хвататься за пневматический "пистолет". Аналогично, Вы должны уверенно управлять классами вручную, прежде чем начинать пользоваться более мощными инструментами. Хотя, как только Вы освоите набор инструментов командной строки, Вы захотите использовать инструмент, автоматизирующий часть рутинной работы с sourcepath и classpath. В основном данные инструменты выполняют работу по организации файлов согласно линиям, которые я уже проложил в данной статье.

IDE

Большинство интегрированных сред разработки, таких как Eclipse и NetBeans, автоматизируют и оказывают помощь в некоторых аспектах управления classpath. Например, когда Вы изменяете имя пакета, Eclipse предлагает переместить соответствующий .java файл для согласования, как показано на рисунке 6:

Рисунок 6. Быстрая настройка для classpath в Eclipse



Имейте в виду, что данные IDE все еще находятся на вершине файловой системы, которая должна быть задана надлежащим способом, особенно если Вам необходимо интегрировать в другие инструменты и другие IDE. Основное преимущество данных инструментов заключается в том, что древовидные списки, таблицы и диалоговые окна GUI заменяют переключатели командной строки, но основная файловая структура остается такой же.

Ant

Ant является де-факто стандартным инструментом для автоматизации процесса конструирования. Вместо размещения директорий в jre\lib\ext или в переменной среды CLASSPATH, Ant действительно позволяет создать одноэтапные процессы конструирования. Однако все равно нужно настроить classpath в Вашем файле Ant build.xml и вручную разместить исходные файлы в верных директориях. Однако хотя бы не нужно повторно вводить их каждый раз при компилировании.

Maven

Maven пошел еще дальше, чем Ant в организации и автоматизации процесса конструирования classpath и связанных с ним задач. Maven по умолчанию предлагает рациональную настройку, которая позволяет конструировать простые проекты с помощью всего лишь нескольких строк кода, если только Вы разместили исходные файлы там, где Maven будет их искать. Вам все еще придется координировать иерархию файловой системы и иерархию пакета. Maven особенно хорош для управления взаимозависимостями посторонних библиотек, хотя он и не так легко приспосабливается, как Ant.

Заключение

Хотя использование classpath и является очень хлопотным, но Вы можете приручить его с помощью нескольких нехитрых правил. А именно:

* Размещайте каждый класс в пакете.
* Строго соблюдайте правила наименования пакетов и классов, и использования прописных символов.
* Убедитесь, что Ваша иерархия пакета соответствует иерархии директории.
* Всегда используйте опцию -d в javac.
* Никогда ничего не размещайте в jre\lib\ext.
* Никогда ничего не размещайте в jre\lib\endorsed.
* Никогда не размещайте .java файлы в той же директории, что и .class файлы.
* Никогда не размещайте какие-либо .java или .class файлы в текущей рабочей директории.

И самая последняя рекомендация: большинство отнимающих много времени проблем с classpath вращаются вокруг таких простых ошибок, как ошибочное написание имени директории или компилирование из неверной директории. Если Вы не можете сразу определить, в чем проблема, обратитесь к друзьям или коллегам, чтобы они на нее посмотрели. Зачастую я нахожусь слишком близко к проблеме, чтобы разглядеть неполадку в моей настройке, которая совершенно очевидна кому-то еще. Вторая пара глаз оказывается очень эффективным методом отладки.

Конечно, classpath совсем не прост, но есть управа и на его сумасбродство, которая делает его послушным. Немного аккуратности и внимания, потраченных на соглашение по присвоению имен, аргументы командной строки и структуры директорий позволят Вам скомпилировать и запустить программы при минимуме суеты.