Использование средств Opentool для расширения возможностей IDE JBuilder. Часть 1

Николай Жишкевич

1. Введение

Одним из ключей к успеху почти любого программного продукта, является возможность расширения тех возможностей, которые были заложены его проектировщиками и разработчиками. Не возможно создать программный продукт, который бы не требовал доработки и модификации, устранения ошибок и добавления новых возможностей, зачастую тех, о которых на стадии проектирования не было ничего известно. Еще одним из ключей к успеху приложения является возможность дать его пользователям в руки эффективный инструмент для доработки приложения. Таким образом, когда мне для работы требуется новая функция программы, мне не нужно ждать пока разработчики сами соизволят сделать ее. А можно используя некоторый api разработать модуль, plugin, компонент и легко интегрировать его в среду, связать и научить работать совместно с другими компонентами, которые быть может, были разработаны такими же энтузиастами или же другими коммерческими организациями.

Если посмотреть внимательно на рынок ПО, то можно увидеть, что подобный подход реализован в самых различных приложениях (photoshop, 3dsmax, msoffice, far и другие). Достаточно большой объем программных продуктов строится по принципу: создается ядро приложения, набор ключевых библиотек и некоторый специальный язык (набор компонентов, правила создания, форматы), в общем, благодаря этому инструментарию создается набор надстроек, компонент которые по сути своей и реализуют функциональность приложения. В мире средств разработки для Java такой

принцип также используется, наиболее известными представителями являются JBuilder с моделью Opentool, Eclipse — соответственно со средствами PDE (Plug-in Development Environment).

2. Opentool и JBuilder. Техническая информация

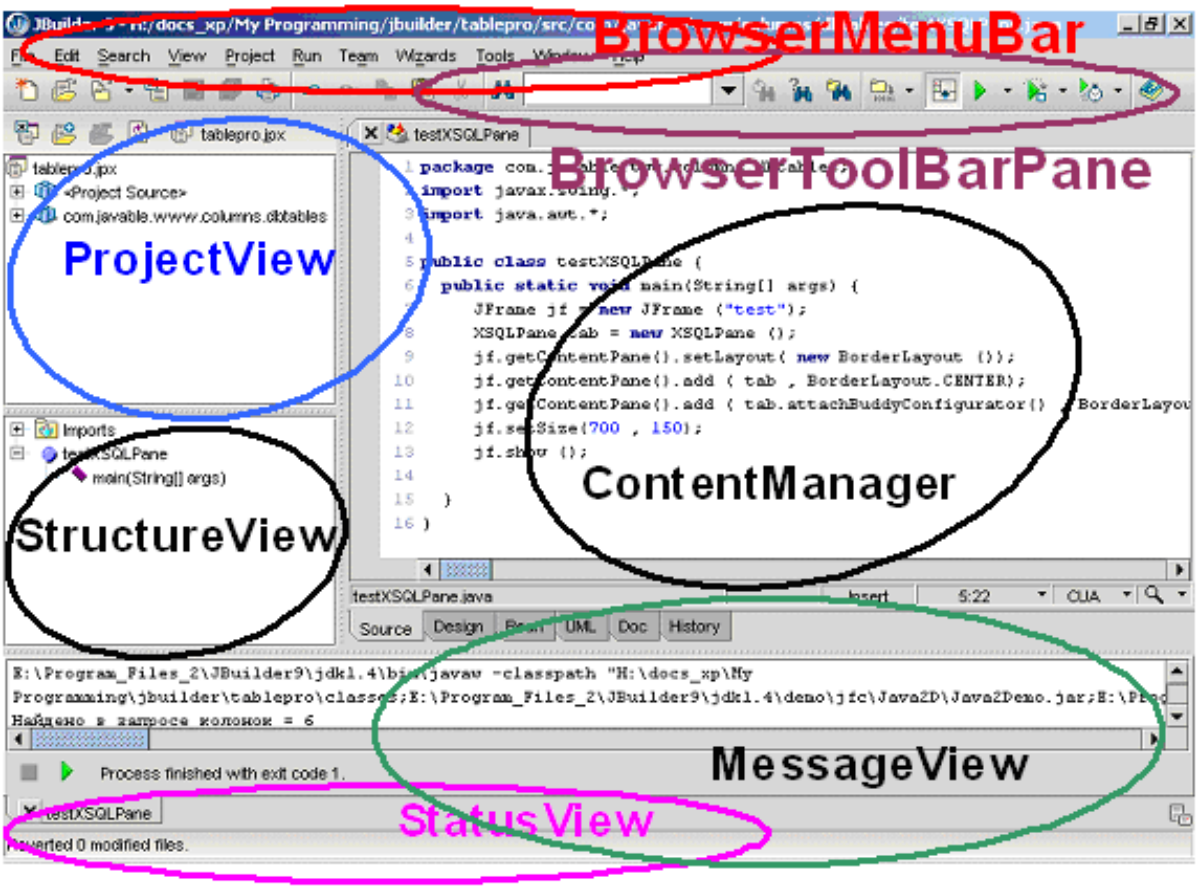
Как известно JBuilder — IDE для создания приложения на Java, и сам целиком написан с помощью java. Подобный ход имеет множество преимуществ, прежде всего не стоит забывать, что Java — по определению кроссплатформенна, а следовательно созданная на Java IDE с минимальными затратами может быть перенесена на другую аппаратную платформу, под другую OS. Стоит также держать в уме, что идея Opentool начинала свой путь именно как инструмент для разработки JBuilder, хотя в настоящий момент ничто не мешает воспользоваться возможностями данного api для создания отличных от JBuilder продуктов. Например, недавно вышедший CbuilderX построен на тех же принципах и компонентах. И хотя мы разрабатывать собственную линию IDE не будем, но держать данный факт в памяти стоит. Надо сказать, что создание полезных Opentool практически невозможно без обращения к объектной модели среды JBuilder. Поэтому прежде чем мы начнем дальнейший рассказ я пожалуй опишу основные классы и сразу после этого мы напишем пример на использование этих классов.

Классы и интерфейсы составляющие возможности Opentools размещаются в двух пакетах, это либо `com.borland.primetime` или `com.borland.JBuilder`. Каждый их этих пакетов имеет свою специализацию, так пакет `com.borland.primetime` содержит в себе все необходимое для создания графического интерфейса среды IDE JBuilder. А пакет `com.borland.jbuilder` содержит набор классов для управления Java-кодом (редактирование, всевозможные подсказки, подсветки и т.д.). И, следовательно, как говорится в справочной системе по Opentool существует возможность использовать пакет `com.borland.primetime` для создания GUI пользователя для другой совместимой IDE. Также можно будет добавить в другую IDE средства управления Java кодом с помощью пакета `com.borland.jbuilder`.

Наибольший интерес из всех классов и интерфейсов для нас предствляет класс `com.borland.primetime.ide.Browser`. Данный класс является по сути своей центром всего интерфейса пользователя среды JBuilder. Если посмотреть на экран с

запущенным JBuilder то можно выделить набор компонент образующих IDE. Каждый из этих компонент представляется конкретным классом. На приведенной ниже картинке я приведу пример окна JBuilder с подписанными и обведенными рамкой элементами.

Однако, давайте сначала рассмотрим наиболее полезные возможности класса `Browser`. Среди его методов есть методы для управления внешним видом окна браузера. Например, можно изменять заголовок окна, цветовое оформление. Можно назначить слушателей событий, например закрытия окна, или изменения размеров и т.д. Кстати, все, что я перечислил (вернее, малую толику всех возможностей) данный класс получил в наследство от своего предка `JFrame`. Больший интерес представляют нам именно возможности класса `Browser`, а среди них есть средства для управления внешним видом приложения, его меню, панелями кнопок, а также управления проектами. На приведенном ниже изображении я показал вид окна среды JBuilder. На этом рисунке условно показаны основные компоненты интерфейса IDE.



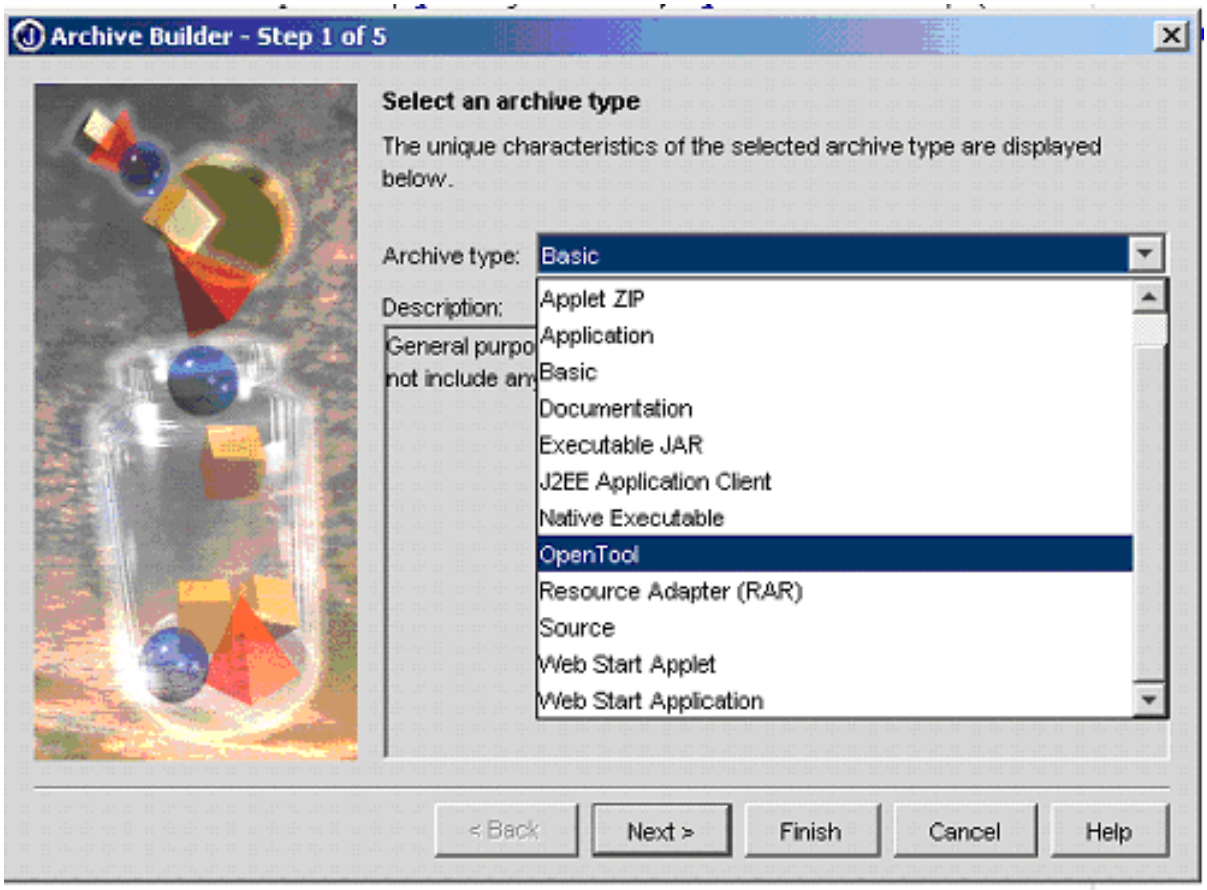
3. Самый простой Opentool. Инструкция по разработке

Начнем с самого простого: мы поставим перед собой задачу создать несложный Opentool, единственная цель существования которого будет выводить на стадии загрузки JBuilder некоторое сообщение, нечто похожее на классический первый пример для любого программиста "hello world".

Для разработки данного Opentool при создании нового проекта необходимо подключить библиотеку "Opentool SDK". Далее нам следует создать специальный класс, играющий роль инсталлятора нашего plugin в среду JBuilder. Существуют несложные правила которых следует придерживаться. Прежде всего, класс, играющий роль инсталлятора, должен содержать статический метод со следующим заголовком.

```
public static void initOpentool(byte majorVersion, byte minorVersion)
```

В качестве параметров данному методу будет передан номер версии среды Opentool. Так как со временем появляются новые возможности или изменяются старые методы и классы. К слову сказать, что если посмотреть на документацию по тому же классу Browser, то можно увидеть что львиная доля интерфейса данного класса имеет пометку "Deprecated" (не рекомендуется к использованию). Так что сделать лишнюю проверку номеров версий будет совсем не лишним. Итак, мы создали класс инсталлятора, реализовав в нем специальный метод " initOpentool". Следующим шагом будет необходимо воспользоваться мастером создания архива. И при этом следует указать что тип архива "Opentool".

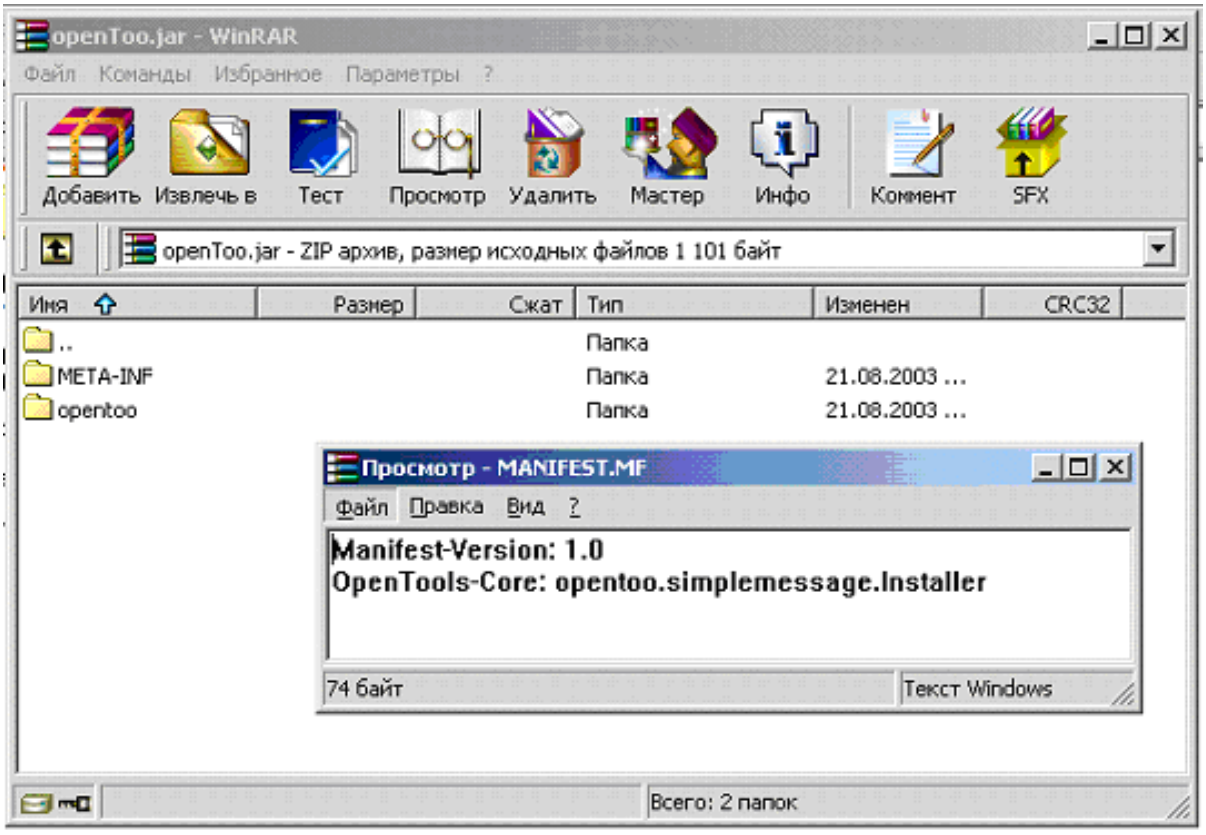


В созданном мастере архиве должен находиться файл манифеста, содержащий описание содержимого данного архива. Содержимое данного текстового файла очень

простое. В общем случае он содержит множество строк, каждая из которых имеет вид: "ВидOpentool: имя класса", например:

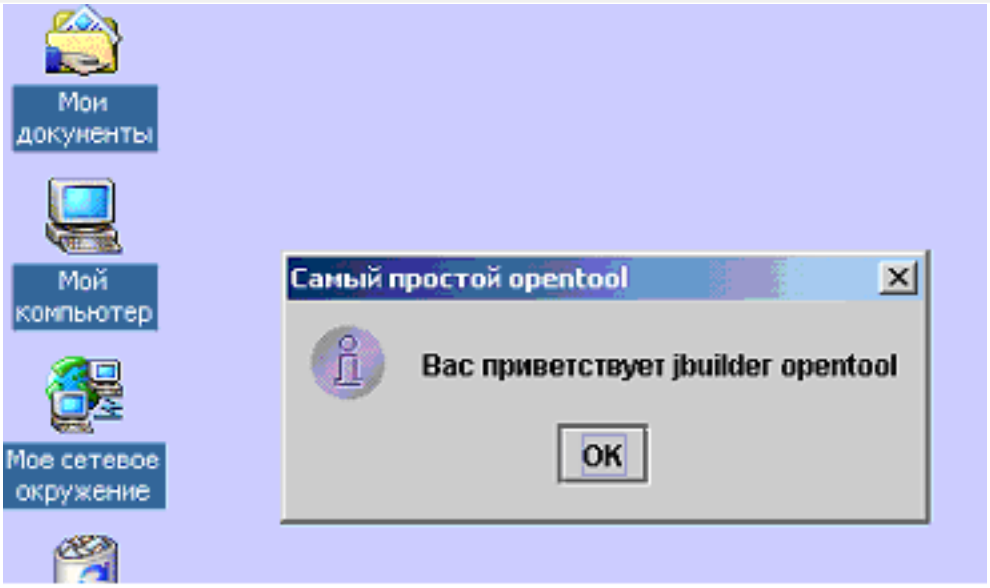
```
Opentools-Core: opentoo.simplemessage.ClockInstaller
Opentools-UI: opentoo.simplemessage.StatInstaller
Opentools-Wizard: opentoo.simplemessage.WizInstaller
```

Указание типа или вида Opentool имеет большое значение. Например если вы создаете Opentool который работает с графическим интерфейсом пользователя, то следует указать его тип как "UI", если же вы укажите его тип "Core", то помните, что раз данный тип Opentool стартует до подготовки графического интерфейса пользователя, то у вас возможны проблемы при обращении к нему. В нашем случае мы укажем тип "Core", а следовательно окно с сообщением появится до самого главного окна JBuilder.



Что же я привожу пример исходного кода и как демонстрацию результат выполнения.

```
package com.javable.www.jbuilder.opentool;

import javax.swing.*;

public class Installer
{
    public static void initOpentool(byte majorVersion, byte minorVersion)
    {
        JOptionPane.showMessageDialog(null,
                                      "Вас приветствует JBuilder Opentool",
                                      "Самый простой Opentool",
                                      JOptionPane.INFORMATION_MESSAGE);
    }
}
Содержимое файла MANIFEST.MF
Opentools-Core: com.javable.www.jbuilder.opentool.Installer
```

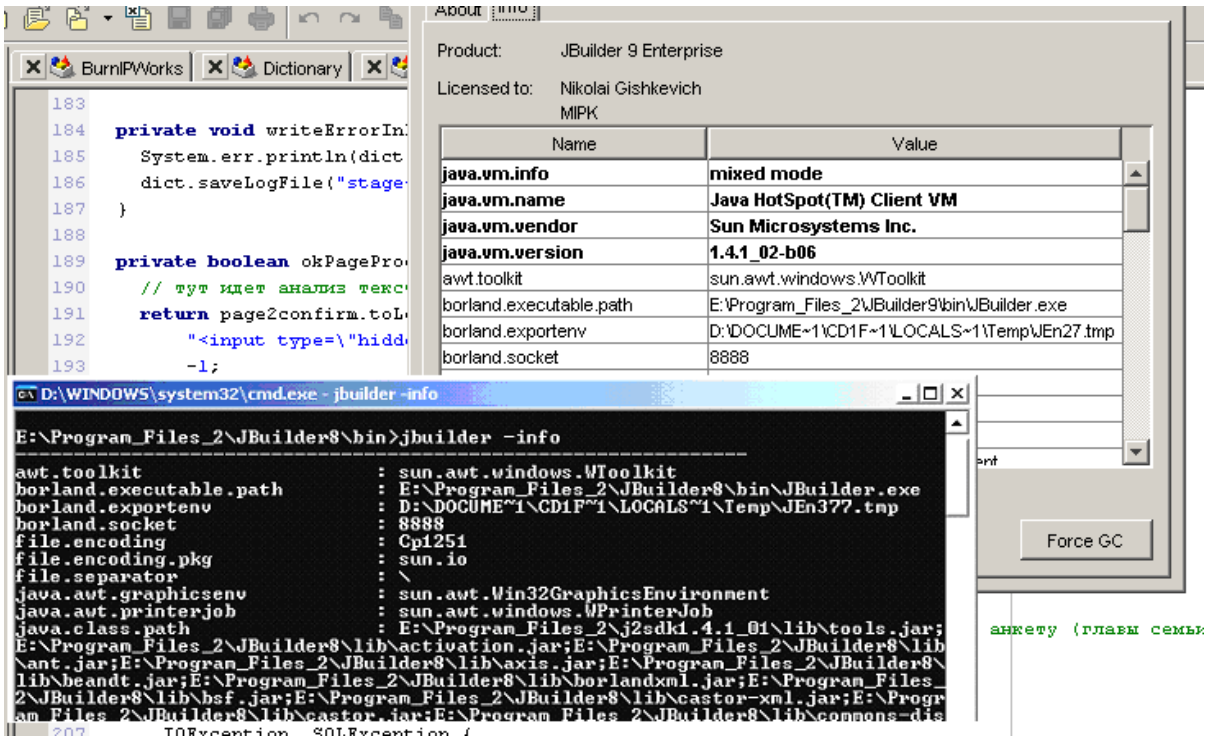


Для того чтобы попробовать в работе созданный нами Opentool необходимо поместить его в специальную директорию "там, где у вас установлен JBuilder"\lib\ext. Кстати по завершению опытов с Opentool бывает очень полезно вернуть ее в исходное положение. Как-то был замечательный случай: я разработал несложный Opentool который при запуске и своей работе выводил отладочные сообщения и записывал их в файл лога. Создал, проинсталлировал и забыл. А спустя пару месяцев я заметил, что

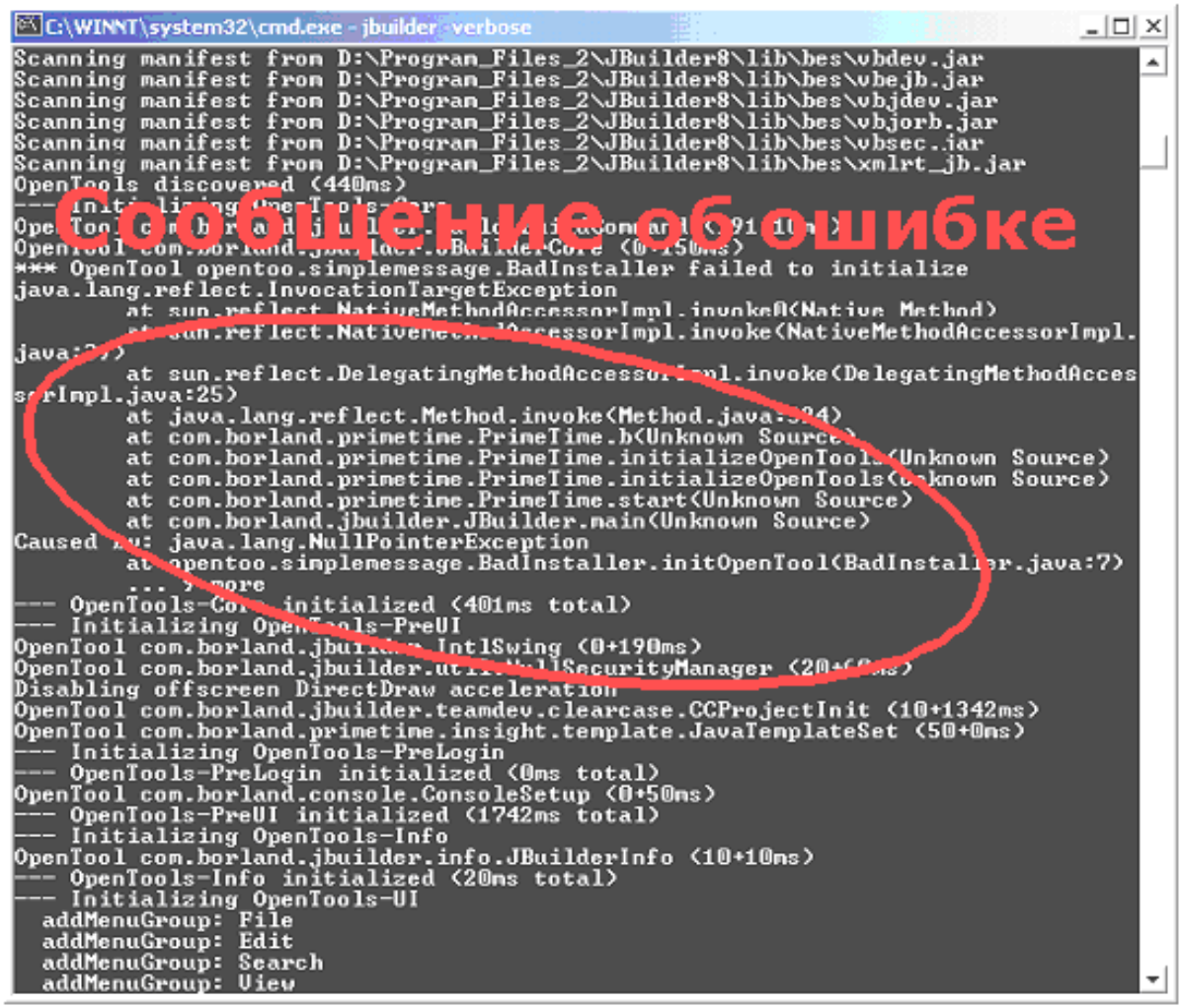
папка JBuilder потяжелела на много десятков мегабайт. Не смертельно, но не приятно. Тот, кто внимательно читает данный опус, уже задался вопросом, а как быть по поводу отладки. Хотя создаваемые нами Opentool вряд ли настолько сложны, что в них можно допустить какую-то ошибку, но, тем не менее, я посвящу пару строки и этой проблеме. Прежде всего, при разработке Opentool забудьте, что вы запускаете JBuilder через ярлык на рабочем столе или в меню "пуск", только из командной строки и только файл "JBuilder.exe". При подобном запуске мы можем, а, следовательно, будут ситуации, когда мы должны будем указывать специальные параметры командной строки.

Но пока давайте просто перечислим варианты. Итак, при запуске из командной строки возможно указать следующий набор опций:

- -build: с помощью данной опции можно выполнить компиляцию проекта не запуская самой графической оболочки.
- -help: думаю, что вы без моей помощи догадались, что делает данная опция.
- -info: отображение на экране информации о конфигурации JBuilder.
- -license: — запуск менеджера лицензирования.
- -nosplash: отключить splash — окно
- -verbose: подробная выдача диагностических сообщений о том, что происходит при загрузке JBuilder.



Наибольший интерес для нас представляет именно последняя опция "-verbose". Как небольшую демонстрацию возможности отслеживать события происходящие при загрузке JBuilder и инициализации Opentool, я создам Opentool с ошибкой, а затем посмотрим какую информацию мы получим при попытке его инсталляции.



Следует обратить внимание на лог в котором выводится список сканируемых файлов, так при загрузке среди них есть jar файлы в каталоге "там, где у вас установлен JBuilder"/lib, однако размещение файлов Opentool в данном каталоге не рекомендуется.

4. Учимся взаимодействовать с объектной средой Browser

Следующее, что мы сделаем, более полно используя объектную модель Browser (согласитесь, что в предыдущем примере мы услугами классов пакетов primetime или

JBuilder совершенно не пользовались) — это попытаемся изменить некоторые характеристики главного окна JBuilder. Давайте создадим Opentool, который будет отображать в заголовке окна JBuilder текущее системное время, а также время, в течение которого мы работаем с JBuilder, и, пожалуй, если время работы будет превышать некоторую величину, например, 60 минут, то будем отображать окно с сообщением о необходимости сделать перерыв. Пример насколько бесполезный настолько и сложный, но надо ведь с чего-то начинать. Для реализации данного Opentool мы воспользуемся двумя статическими методами, принадлежащими классу Browser:

- `public static java.lang.String getBrowserTitle()` — получение текущего заголовка окна браузера.
- `public static void setBrowserTitle(java.lang.String title)` — установка нового значения заголовка окна браузера.

На стадии инициализации создаваемого нами Opentool необходимо будет создать поток (Thread), который с периодом в одну секунду будет устанавливать новое значение заголовка окна браузера. А вот и код примера.

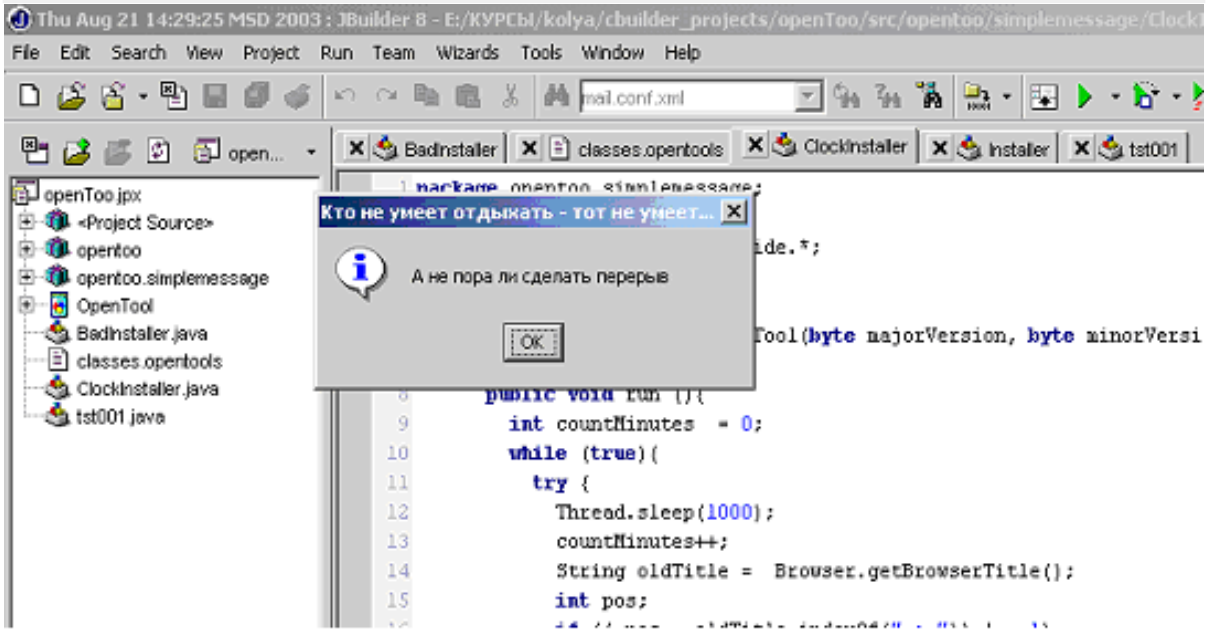
```
package com.javable.www.jbuilder.opentool;
import com.borland.primetime.*;
import com.borland.primetime.ide.*;

import java.io.*;

import java.text.*;

import java.util.*;

import javax.swing.*;


public class SimpleClockInstaller
{
    public static void initOpentool(byte majorVersion,
                                     byte minorVersion)
    {
        new Thread() {
            public void run()
```

```
        {

            int countMinutes = 0;

            while (true)
            {

                try
                {
                    Thread.sleep(1000);
                    countMinutes++;

                    String oldTitle = Browser.getBrowserTitle();
                    int pos;

                    if ((pos = oldTitle.indexOf(
                                       " : ")) != -1)
                    {
                        oldTitle = oldTitle.substring(
                                            pos + 3);
                    }

                    oldTitle = " : " +
                               oldTitle;
                    Browser.setBrowserTitle(
                            new Date() +
                            oldTitle);

                    if (countMinutes >= 60000)
                    {
                        countMinutes = 0;
                        JOptionPane.showMessageDialog
                         (Browser.getActiveBrowser(),
                           "А не пора ли сделать перерыв",
                           "Кто не умеет отдыхать — тот не умеет работать",
                            JOptionPane.INFORMATION_MESSAGE);
                    }
                }
                catch (InterruptedException ex)
                {
```

```
                        }
                    }
                }
            }.start();
        }
}
```

А вот и примерный вид файла манифеста содержимого архива.

```
Opentools-UI: com.javable.www.jbuilder.opentool.SimpleClockInstaller
```



Давайте рассмотрим пример передачи нашему Opentool параметров через командную строку. Для иллюстрации мы разовьем приведенный выше пример, добавив возможность указания различных форматов отображения. Влиять на используемый формат представления даты/времени можно будет с помощью указания параметров командной строки. Например, можно будет запустить JBuilder с помощью следующей командной строки:

```
JBuilder -clock hh:mm:ss
```

И соответственно в заголовке окна JBuilder будет отображаться время в формате "часы:минуты:секунды".

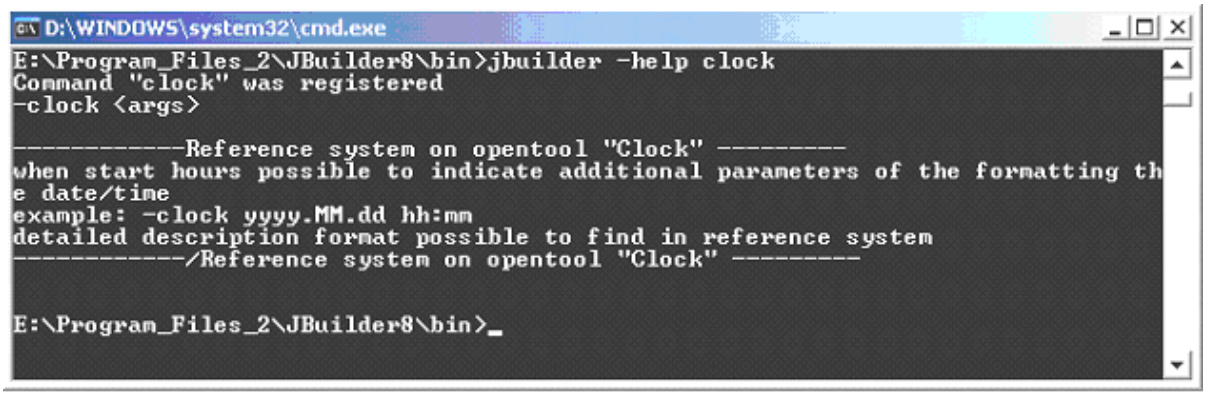
Для того, чтобы дать возможность запускать JBuilder с различными параметрами

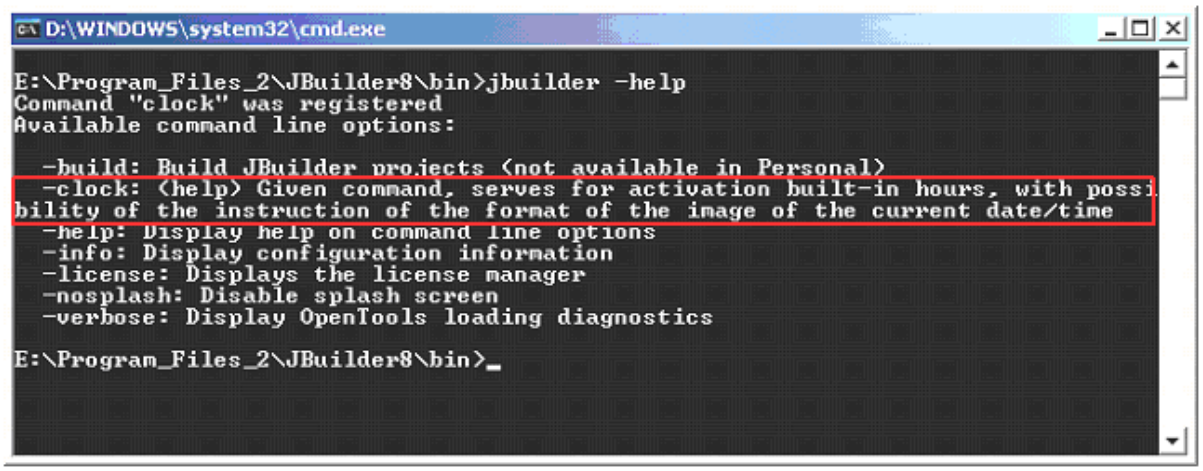
командной строки нам потребуется выполнить регистрацию новой команды "clock". Технически это выполняется с помощью вызова метода.

```
PrimeTime.registerCommand("clock", clockCommand);
```

В данном примере переменная clockCommand представляет собой экземпляр класса реализующего функциональность некоторой команды, к данному классу представляется требование реализовать интерфейс `com.borland.primetime.Command`. Данный интерфейс содержит всего 4 метода, краткое описание которых я привожу ниже.

- `public java.lang.String getCommandDescription()` — данный метод возвращает строку содержащую описание команды, данная строка будет использована справочной системой JBuilder при его запуске с указанием параметра командной строки "-help". На приведенной ниже иллюстрации я показываю как возможно с помощью команды "-help" получить информацию о всех командах, как стандартных так и добавленных как Opentool.
- `public void printCommandHelp(java.io.PrintStream printStream)` — данный метод служит для печати более подробной информации о параметрах которых получает создаваемая нами команда. Снова на изображении ниже как пример я вызываю подробную информацию о созданной мною команде с помощью "JBuilder –help clock".
- `public boolean takesCommandArguments()` — данный метод возвращает логическое значение, признак того получает ли данная команда параметры или нет. Если возвращается "true" — то параметры для этой команды допустимы, иначе — нет.
- `public void invokeCommand(java.lang.String[] args)` – данный метод непосредственно вызывается при указании параметра командной строки. В качестве аргументов ему передается все, что было указано в командной строке после имени команды.





Для форматирования текущей даты мы воспользуемся классом `SimpleDateFormat` общая схема использования которого приведена ниже.

```
SimpleDateFormat frm = new SimpleDateFormat ("yyyy.MM.dd hh:mm");
String formattedDate = formatter.format(new Date ());
```

В качестве параметров конструктора для `SimpleDateFormat` мы передаем формат, описывающий внешний вид результата преобразования. Каждый символ представляет собой компоненту даты/времени. Полное соответствие можно посмотреть в документации. Так что сосредоточимся лучше на вопросе получения параметров командной строки. Прежде всего, стоит понимать, что все параметры командной строки рассматриваются как объекты command, которые должны быть зарегистрированы с помощью вызова

```
PrimeTime.registerCommand("имя команды", new ОбъектРеализующийФункциональностьКоманды()
```

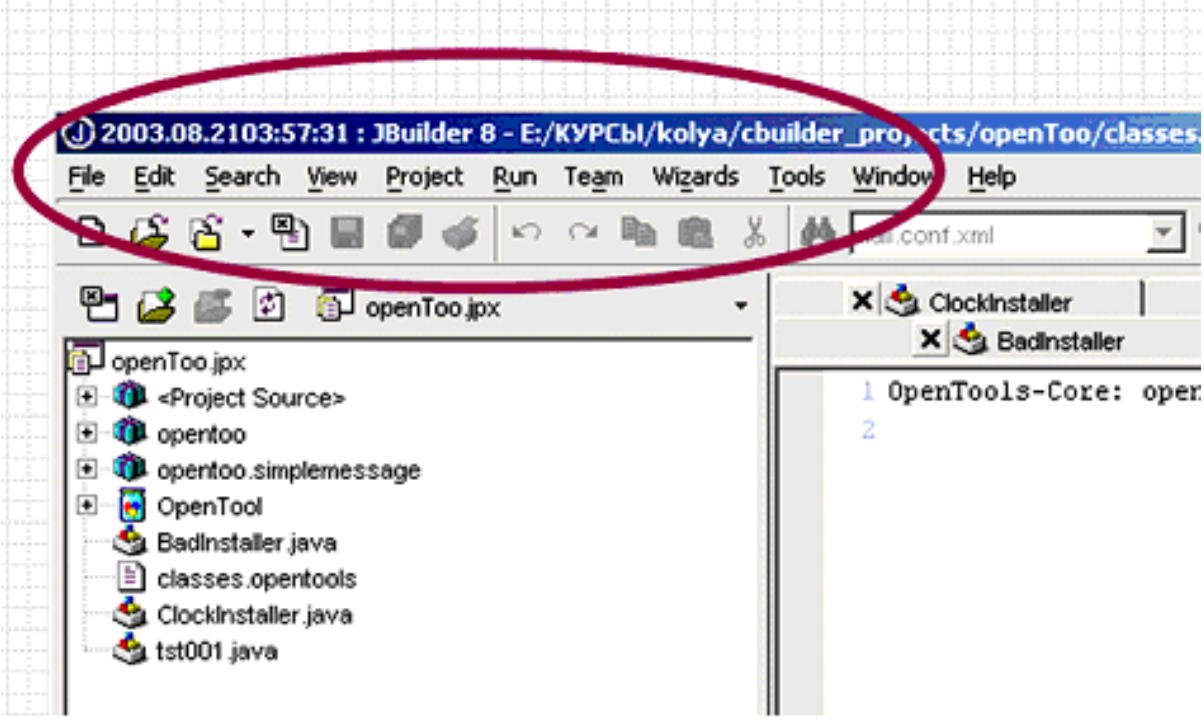
В противном случае команды будут проигнорированы. И попытка их указания при запуске JBuilder приведет к возникновению ошибки. Далее я привожу исходный код примера, в котором регистрируется новая команда для управления форматом отображения даты/времени в заголовке окна JBuilder.

```
package com.javable.www.jbuilder.opentool;
import com.borland.primetime.*;
import com.borland.primetime.ide.*;

import java.io.*;

import java.text.*;

import java.util.*;

import javax.swing.*;


public class ClockInstaller
{
    static class ClockCommand
        implements Command
    {

        String format = "yyyy.MM.dd hh:mm:ss";
        SimpleDateFormat sdf = new SimpleDateFormat(
                                        format);
        ;public String formatIt()
        {

            try
            {

                return sdf.format(new Date());
            }
            catch (Exception ex)
            {

                return "Ошибка отображения даты/времени";
            }
```

```
        }

        public boolean takesCommandArguments()
        {

            return true;
        }

        public void invokeCommand(String[] parm1)
        {

            String res = "";

            for (int i = 0; i < parm1.length; i++)
                res += "" + parm1[i];

            try
            {
                sdf = new SimpleDateFormat(res);
            }
            catch (Exception ex)
            {
                System.err.println(
                        "Mistake, value specified by your line is not a
                        real format of the date/time:" +
                        res);
            }
        }

        public String getCommandDescription()
        {

            return "(help) Given command, serves for activation built-in hours,
            with possibility of the instruction of the format
            of the image of the current date/time";
        }

        public void printCommandHelp(PrintStream parm1)
        {
            parm1.println(
```

```
                "------------Reference system on Opentool \"Clock\" ---------");
        parm1.println(
                "when start hours possible to indicate additional parameters of
                the formatting the date/time");
        parm1.println(
                "example: -clock yyyy.MM.dd hh:mm");
        parm1.println(
                "detailed description format possible to find in reference system")

        // конечно делать подобный help несколько неправильно,
        // но учитывая что пользоваться JBuilder
        // будут более или менее опытные программисты
        // (так и хотелось написать пользователи)
        // то думаю, найдут
        parm1.println(
                "------------/Reference system on Opentool \"Clock\" ---------");
    }
}

public static void initOpentool(byte majorVersion,
                                byte minorVersion)
{

    final ClockCommand clockCommand = new ClockCommand();
    PrimeTime.registerCommand("clock",
                              clockCommand);
    System.out.println(
            "Command \"clock\" was registered");
    new Thread() {
        public void run()
        {

            int countMinutes = 0;

            while (true)
            {

                try
                {
                    Thread.sleep(1000);
```

```
                        countMinutes++;

                        String oldTitle = Browser.getBrowserTitle();
                        int pos;

                        if ((pos = oldTitle.indexOf(
                                          " : ")) != -1)
                        {
                            oldTitle = oldTitle.substring(
                                               pos + 3);
                        }

                        oldTitle = " : " +
                                   oldTitle;
                        Browser.setBrowserTitle(
                                clockCommand.formatIt() +
                                oldTitle);

                        if (countMinutes >= 60)
                        {
                            countMinutes = 0;
                            JOptionPane.
                             showMessageDialog(Browser.getActiveBrowser(),
                                               "А не пора ли сделать перерыв",
                                               "Кто не умеет отдыхать — тот не умеет ра
                                               JOptionPane.INFORMATION_MESSAGE);
                        }
                    }
                    catch (InterruptedException ex)
                    {
                    }
                }
            }
        }.start();
    }
}
```



5. Модель обработки событий. Узнаем о том, что происходит в среде JBuilder

Следующее, что мы попытаемся делать это попробуем отслеживать некоторые события происходящие в среде JBuilder. Самое простое, что можно сделать — это создать класс, реализующий интерфейс слушателя событий `com.borland.primetime.ide.BrowserListener`. В данном интерфейсе есть методы, вызываемые при наступлении самых разных событий, например, некоторые из них:

- `public void browserActivated(Browser browser)` — данный метод вызывается при активации окна среды JBuilder, или при создании нового окна.
- `public void browserClosing(Browser browser) throws VetoException` — закрытие окна Browser.
- `public void browserProjectActivated(Browser browser, Project project)` — активация иного проекта.

- `public void browserNodeActivated(Browser browser, Node node)` — переключение от одного документа к другому.

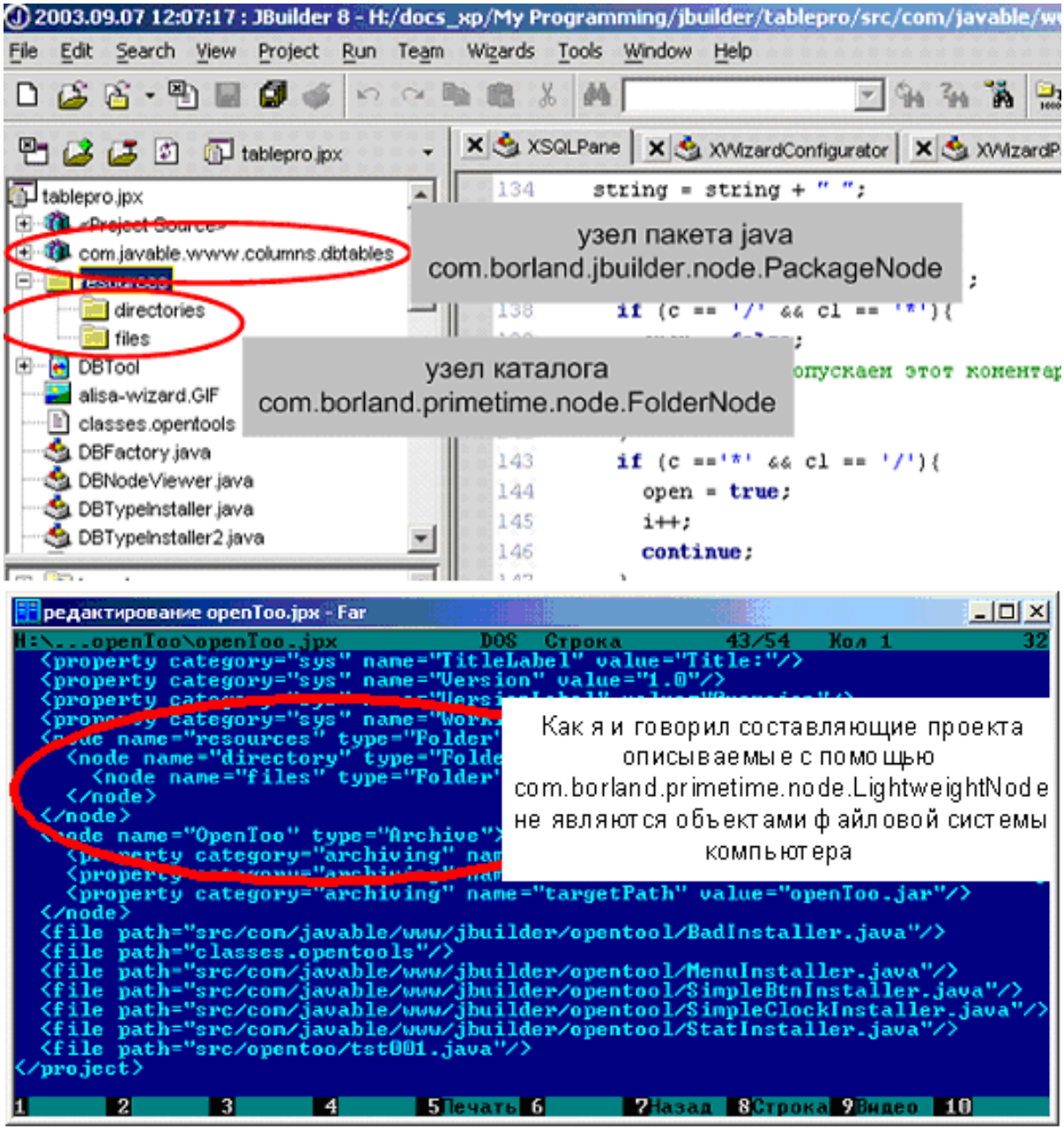
Разумеется, что это не полный список, но тут я полагаю, что те, кому эта информация потребуется, сможет почерпнуть ее в справочной системе, а мы тем временем обратим свое внимание на то, как создать Opentool, который при переключении от одного файла к другому будет выводить в строке статуса сообщение с информацией о дате последней модификации документа.

Для создания данного примера нам потребуется создать класс, который будет поддерживать и реализовывать интерфейс `BrowserListener` или будет производным от класса `com.borland.primetime.ide.BrowserAdapter`. В нашем классе следует перекрыть метод `browserViewerActivated` — данный метод вызыватеся для извещения о всяком изменении класса просмотра документа (например, при переключении из режима "Source" в режим "Design" или "UML"). В качестве параметров данному методу передается объект, реализующий интерфейс "com.borland.primetime.node.Node". Данный интерфейс служит для описания узла проекта.

Что же, как я начал говорить выше, действительно хороший (или более точно, полезный) Opentool можно будет разработать, только используя модель объектов среды JBuilder. Также, для того чтобы научиться разрабатывать хороший Opentool необходимо будет научиться реагировать на события, происходящие в среде. Следующий пример, который мы рассмотрим, как раз и будет представлять демонстрацию методики создания слушателя событий среды JBuilder. Я разработаю Opentool который будет отслеживать переход (открытие) файлов проекта и выводящего в строке статуса сообщение о дате последней модификации файла.

Для создания данного примера нам потребуется кроме знания модели слушателей реализованной в JBuilder, кратко об этой модели я упомянул ранее, также и представление о концепции представления проектов и их содержимого. Каждый проект, или находящийся в нем файл представляется узлом, для этого служит класс (Node). Потом от него были порождены классы `com.borland.primetime.node.Project` — данный класс представляет собой узел-контейнер в котором размещаются узлы описывающие различные составляющие проекта, например: `com.borland.primetime.node.UrlNode`, данный класс

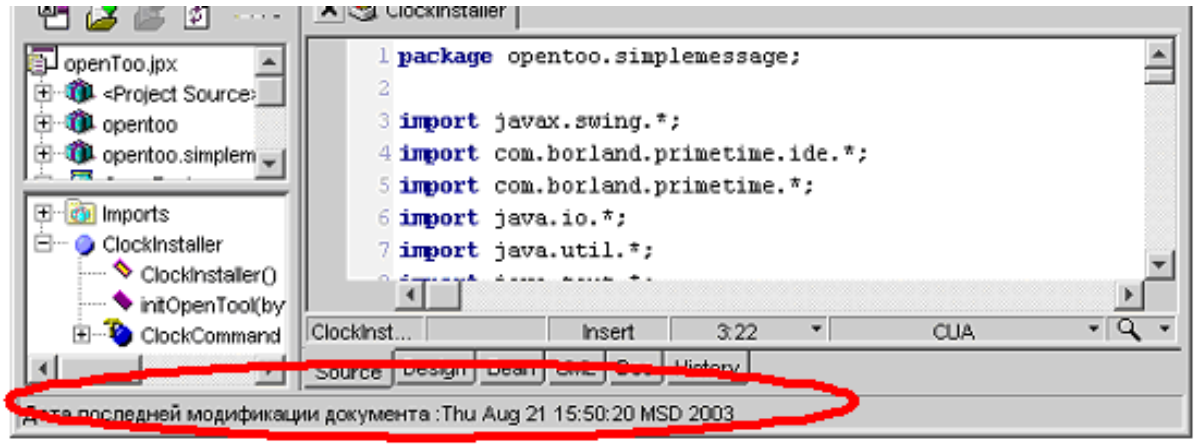
представляет собой компонент проекта имеющий представление на жестком диске, добраться до файла (его имени и местоположении) можно с помощью метода getUrl. От класса `FileNode` производного от `UrlNode` производны различные классы соответствующие различным типам файлов (`BinaryFileNode`, `ClassFileNode`, `ImageFileNode`, `JarFileNode`, `SoundFileNode`, `TextFileNode`). В свою очередь многие из этих классов имеют потомков, которые соответствуют подтипам файлов, например для класса TextFileNode есть соответствующие классы потомки для различных типов текстовых файлов (*.Java, *.txt, *.jsp, *txt и другие), это классы (`CPPFileNode`, `HTMLFileNode`, `IDLFileNode`, `JavaFileNode`, `JavaScriptFileNode`, `PropertiesFileNode`, `SchemaFileNode`, `SQLFileNode`, `SQLJFileNode`, `XMLFileNode`). И как вы догадались это еще не предел, например, класс XMLFileNode имеет еще несколько потомков, это `JnlpFileNode`, `TldFileNode`. От базового класса Node также производен класс `com.borland.primetime.node.LightweightNode`, этот класс играет основу для составляющих проекта, которые являются абстрактными компонентами, т.е. не имеют реального места хранения на жестком диске и являются средствами для организации структуры проекта. На приведенном ниже изображении я привожу пример файла проекта "*.jpx", если присмотреться, то можно заметить что ссылки на созданные мною в проекте каталоги хранятся внутри файла проекта.



А теперь давайте попробуем создать пример, о котором я говорил. Пока в нем модель узлов затронут очень поверхностно, что же значит, потом будет необходимо создать более развернутый пример.

```
package com.javable.www.jbuilder.opentool;
import com.borland.primetime.ide.*;
import com.borland.primetime.node.*;

import java.util.*;

public class StatInstaller
{
    public static void initOpentool(byte majorVersion,
                                    byte minorVersion)
    {
        Browser.addStaticBrowserListener(new BrowserAdapter() {
            public void browserViewerActivated(Browser browser,
                                               Node node,
                                               NodeViewer viewer)
            {

                Date dat = new Date(((FileNode)node).getUrl().
                        getFileObject().lastModified());
                browser.getStatusView().setText(
                        "Дата последней модификации документа :" + dat);
            }
        });
    }
}
```

Ниже я привожу результата выполнения данного фрагмента кода. В строке статуса отображается информация о дате последней модификации активного файла.



6. Заключение

Сегодня мы сделали только первый шаг в направлении очень интересной и главное полезной возможности JBuilder — возможности самим расширять его инструменты, создавая классы Opentools. Мы рассмотрели набор основных классов описывающих интерфейс пользователя среды IDE. Мы научились создавать архивы Opentools, разрабатывать файлы манифеста и отслеживать процесс загрузки Opentool. Мы попробовали создать обработчик событий для узлов проекта JBuilder.

Следующая статься будет посвящена созданию Opentools которые будут более тесно интегрированы и будут уметь взаимодействовать с панелями кнопок, управлять меню, и в конце мы попытаемся создать собственного мастера для автоматизации какой-нибудь рутинной операции.

7. Ресурсы

К сожалению, пока ничего кроме справочной системы самого JBuilder.