Глава 2. Объектно-ориентированное проектирование и платформа NetBeans

2.1.Процедурное и объектно-ориентированное программирование. Инкапсуляция

Объектно-ориентированное программирование (ООП) - это методология программирования, опирающаяся на три базовых принципа:

- *инкапсуляцию*,
- *наследование*,
- *полиморфизм*.

Язык Java является объектно-ориентированным и в полном объёме использует эти принципы. В данном параграфе рассматривается принцип инкапсуляции, наследованию и полиморфизму посвящены отдельные параграфы.

Построение программ, основанных на ООП, принципиально отличается от более ранней методики *процедурного программирования*, в которой основой построения программы служили *подпрограммы*.

Программа – это набор инструкций процессору и данных, объединённых в единую функционально законченную последовательность, позволяющую выполнять какую-нибудь конкретную деятельность.

Подпрограмма – это набор инструкций и данных, объединённых в относительно самостоятельную последовательность, позволяющую выполнять какую-нибудь конкретную деятельность внутри программы. При этом подпрограмма не может работать самостоятельно - она запускается из программы, и может получать из неё данные или передавать их в программу.

Подпрограммы принято делить на подпрограммы-процедуры и подпрограммы-функции. Подпрограммы-процедуры вызываются для выполнения каких-либо действий, например – распечатки текста на принтере. Подпрограммы-функции выполняют какие-либо действия и возвращают некоторое значение. Например, проводится последовательность действий по вычислению синуса, и возвращается вычисленное значение. Или создаётся сложно устроенный объект и возвращается ссылка на него (адрес ячейки, в которой он находится).

Программы, написанные в соответствии с принципами процедурного программирования, состоят из набора подпрограмм, причём для решения конкретной задачи программист явно указывает на каждом шагу, *что* делать и *как* делать. Эти программы практически полностью (процентов на девяносто) состоят из решения конкретных задач.

Программы, написанные в соответствии с принципами ООП, пишутся совершенно иначе. В них основное время занимает продумывание и описание того, как устроены классы. Код с описанием классов предназначен для многократного использования без внесения каких-либо изменений. И только небольшая часть времени посвящается решению конкретной задачи – написания такого кода с использованием классов, который в других задачах такого типа не применить. Именно благодаря такому подходу объектное программирование приобрело огромную популярность – при необходимости решения сходных задач можно использовать уже готовый код, модифицировав только ту часть программы, которая относится к решению конкретной задачи.

Подробный разбор принципов ООП будет дан позже. Пока же в общих чертах разъясним их суть.

Самым простым из указанных в начале параграфа принципов является *инкапсуляция*. Это слово в общем случае означает “заключение внутрь капсулы”. То есть ограничение

доступа к внутреннему содержимому снаружи и отсутствие такого ограничения внутри капсулы. В объектном программировании “инкапсуляция” означает использование *классов* – таких *типов*, в которых кроме *данных* описаны подпрограммы, позволяющие работать с этими данными, а также выполнять другие действия. Такие подпрограммы, инкапсулированные в класс, называются *методами*. Поля данных и методы, заданные в классе, часто называют *членами класса* (class members).

Класс – это описание того, как будет устроен *объект*, являющийся *экземпляром данного класса*, а также какие методы объект может вызывать. Заметим, что методы, в отличие от других подпрограмм, могут напрямую обращаться к данным своего объекта. Так как экземплярами классов (“воплощением” в реальность того, что описано в классе) являются объекты, классы называют *объектными типами*.

Все объекты, являющиеся экземплярами некоторого класса, имеют одинаковые наборы полей данных (*атрибуты* объекта) – но со значениями этих данных, которые свои для каждого объекта. Поля данных это переменные, заданные на уровне описания класса, а не при описании метода. В процессе жизни объекта эти значения могут изменяться. Значения полей данных объекта задают его *состояние*. А методы задают *поведение* объекта. Причём в общем случае на это поведение влияет состояние объекта – методы пользуются значениями его полей данных.

Классы в Java задаются следующим образом. Сначала пишется зарезервированное слово class, затем имя класса, после чего в фигурных скобках пишется реализация класса – задаются его поля (*глобальные переменные*) и методы.

Объектные переменные – такие переменные, которые имеют объектный тип. В Java объектные переменные – это не сами объекты, а только *ссылки* на них. То есть все объектные типы являются ссылочными.

Объявление объектной переменной осуществляется так же, как и для других типов переменных. Сначала пишется тип, а затем через пробел имя объявляемой переменной. Например, если мы задаём переменную obj1 типа Circle, “окружность”, её задание осуществляется так :

```
Circle obj1;
```

Связывание объектной переменной с объектом осуществляется путём присваивания. В правой части присваивания можно указать либо функцию, возвращающую ссылку на объект (адрес объекта), либо имя другой объектной переменной. Если объектной переменной не присвоено ссылки, в ней хранится значение null. Объектные переменные можно сравнивать на равенство, в том числе на равенство null. При этом сравниваются не сами объекты, а их адреса, хранящиеся в объектных переменных.

Создаётся объект с помощью вызова специальной подпрограммы, задаваемой в классе и называемой *конструктором*. Конструктор возвращает ссылку на созданный объект. Имя конструктора в Java *всегда совпадает с именем класса*, экземпляр которого создаётся. Перед именем конструктора во время вызова ставится оператор new – “новый”, означающий, что создаётся новый объект. Например, вызов

```
obj1=new  Circle();
```

означает, что создаётся новый объект типа Circle, “окружность”, и ссылка на него (адрес объекта) записывается в переменную obj1. Переменная obj1 до этого уже должна быть объявлена. Оператор new отвечает за динамическое выделение памяти под создаваемый объект.

Часто совмещают задание объектной переменной и назначение ей объекта. В нашем случае оно будет выглядеть как

```
Circle obj1=new  Circle();
```

У конструктора, как и у любой подпрограммы, может быть список параметров. Они нужны для того, чтобы задать начальное состояние объекта при его создании. Например, мы хотим, чтобы у создаваемой окружности можно было при вызове конструктора задать координаты x, y её центра и радиус r. Тогда при написании класса Circle можно предусмотреть

конструктор, в котором первым параметром задаётся координата x, вторым – y, третьим – радиус окружности r. Тогда задание переменной obj1 может выглядеть так:

```
Circle obj1=new  Circle(130,120,50);
```

Оно означает, что создаётся объект-окружность, имеющий центр в точке с координатами x=130, y=120, и у которой радиус r=50.

Если разработчики класса не создали ни одного конструктора, в реализации класса автоматически создаётся конструктор по умолчанию, имеющий пустой список параметров. И его можно вызывать в программе так, как мы это первоначально делали для класса Circle.

Отметим ещё одно правило, касающееся используемых имён. Как мы помним, имена объектных типов принято писать с заглавной буквы, а имена объектных переменных – с маленькой. Если объектная переменная имеет тип Circle, она служит ссылкой на объекты-окружности. Поэтому имя obj1 не очень удачно – мы используем его только для того, чтобы подчеркнуть, что именно с помощью этой переменной осуществляется связь с объектом, и чтобы читатель не путал тип переменной, её имя и имя конструктора. В Java принято называть объектные переменные так же, как их типы, но начинать имя со строчной буквы. Поэтому предыдущий оператор мог бы выглядеть так:

```
Circle circle=new  Circle(130,120,50);
```

Если требуется работа с несколькими объектными переменными одного типа, их принято называть в соответствии с указанным выше правилом, но добавлять порядковый номер. Следующие строки программного кода создают два независимых объекта с одинаковыми начальными параметрами:

```
Circle circle1=new  Circle(130,120,50);
Circle circle2=new  Circle(130,120,50);
```

С помощью объектных переменных осуществляется доступ к полям данных или методам объекта: сначала указывается имя переменной, затем точка, после чего пишется имя поля данных или метода. Например, если имя объектной переменной obj1, а имя целочисленного поля данных x, то присваивание ему нового значения будет выглядеть как

```
obj1.x=5;
```

А если имя подпрограммы show, у неё нет параметров и она не возвращает никакого значения, то её вызов будет выглядеть как

```
obj1.show();
```

Методы делятся на *методы объектов* и *методы классов*. Чаще всего пользуются методами объектов. Они так называются потому, что пользуются полями данных объектов, и поэтому их можно вызывать только из самих объектов. Методы классов, напротив, не пользуются полями данных объектов, и могут работать при отсутствии объекта. Поэтому их можно вызывать как из классов, так и из объектов. Формат вызова: *имяКласса.имяМетода(список параметров)* или *имяОбъекта. имяМетода(список параметров)*.

При задании метода класса перед его именем необходимо поставить модификатор static – “статический”. Это крайне неудачное название, пришедшее в язык Java из C++. Мы никогда не будем называть такие методы статическими, а будем называть их методами класса, как это принято в теории программирования.

Точно так же переменные (поля данных) делятся на *переменные объектов* и *переменные классов*. При задании переменной класса перед её именем необходимо поставить модификатор static. Переменные класса, как и методы класса, можно вызывать как из классов, так и из объектов. Формат вызова: *имяКласса.имяПеременной* или *имяОбъекта.имяПеременной*.

Не следует путать классы, объекты и объектные переменные. Класс – это тип, то есть описание того, как устроена ячейка памяти, в которой будут располагаться поля данных объекта. Объект – это содержимое данной ячейки памяти. А в переменной объектного типа содержится адрес объекта, то есть адрес ячейки памяти. Сказанное относится только к языкам с динамической объектной моделью, каким, в частности, является Java. В C++ это не

так.

Как уже было сказано, кроме полей данных в классе описываются методы. Несмотря на схожесть задания в классе полей и методов их реальное размещение во время работы программы отличается. Методы не хранятся в объектах, но объекты могут их вызывать. Каждый объект имеет свой комплект полей данных – “носит свои данные с собой”. Если имеется сотня объектов одного типа, то есть являющихся экземплярами одного и того же класса, в памяти компьютера будет иметься сотня ячеек памяти, устроенных так, как это описано в классе. Причём у каждого объекта значения этих данных могут быть свои. Например, если объект является окружностью, отрисовываемой на экране, у каждой окружности будет свой набор координат, радиусов и цветов. Если мы будем отрисовывать окружности с помощью метода show(), нет необходимости в каждом объекте хранить код этого метода – он для всех ста объектов будет одним и тем же. Поэтому методы не хранятся в объектах – они хранятся в классах. Класс – более общая сущность, чем объект, и до того, как во время работы программы в памяти компьютера будет создан объект, сначала должен быть загружен в память соответствующий ему класс. В Java имеется возможность создавать переменные типа “класс”, и с их помощью обращаться к классам таким образом, как будто это объекты особого рода. Но, в отличие от обычных объектов, такие “объекты” не могут существовать в нескольких экземплярах, и правила работы с ними принципиально отличаются от работы с объектами. Такие сущности называются *метаобъектами*.

Объявление переменных может осуществляться либо в классе, либо в методе. В первом случае мы будем говорить, что переменная является полем данных объекта, или *глобальной* переменной. Во втором – что она является *локальной* переменной.

2.2. Работа со ссылочными переменными. Сборка мусора

Объекты, как мы уже знаем, являются экземплярами ссылочных типов. Работа со ссылочными переменными имеет специфику, принципиально отличающую её от работы с переменными примитивного типа. В каждой переменной примитивного типа содержится своё значение, и изменение этого значения не влияет на те значения, которые можно получить с помощью других переменных. Причём имя переменной примитивного типа можно рассматривать как имя ячейки с данными, и у такой ячейки может быть только одно имя, которое не может меняться по ходу работы программы. Для ссылочных переменных это не так.

Переменные ссылочного типа содержат адреса данных, а не сами данные. Поэтому присваивания для таких переменных меняют адреса, но не данные. Кроме того, из-за этого под них выделяется одинаковое количество памяти независимо от типа объектов, на которые они ссылаются. А имена ссылочных переменных можно рассматривать как псевдонимы имён ячеек с данными – у одной и той же ячейки с данными может быть сколько угодно псевдонимов, так как адрес одной и той же ячейки можно копировать в произвольное число переменных соответствующего типа. И все они будут ссылаться на одну и ту же ячейку с данными.

В Java ссылочные переменные используются для работы с объектами: в этом языке программирования используется *динамическая объектная модель*, и все объекты создаются динамически, с явным указанием в программе момента их создания. Это отличает Java от C++, языка со *статической объектной моделью*. В C++ могут существовать как статически заданные объекты, так и динамически создаваемые. Но это не преимущество, как могло бы показаться, а проблема, так как в ряде случаев создаёт принципиально неразрешимые ситуации.

Следует отметить, что сами объекты безымянны, и доступ к ним осуществляется только через ссылочные переменные. Часто говорят про ссылочную переменную как про сам объект, поскольку долго и неудобно произносить “объект, на который ссылается данная переменная”.

Но этого по мере возможности следует избегать.
Ссылочной переменной любого типа может быть присвоено значение `null`, означающее, что она никуда не ссылается. Попытка доступа к объекту через такую переменную вызовет ошибку. В Java все ссылочные переменные первоначально инициируются значением `null`, если им не назначена ссылка на объект прямо в месте объявления. Очень часто встречающаяся ошибка – попытка доступа к полю или методу с помощью ссылочной переменной, которой не сопоставлен объект. Такая ошибка не может быть обнаружена на этапе компиляции и является ошибкой времени выполнения (Run-time error). При этом приложение Java генерирует исключительную ситуацию (ошибку) попытки доступа к объекту через ссылку null с сообщением следующего вида:
Exception in thread "AWT-EventQueue-0" java.lang.NullPointerException.

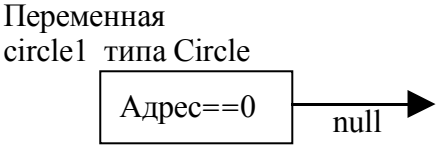
Последовательность действий со ссылочными переменными и объектами удобно описывать с помощью рисунков, на которых каждой такой переменной и каждому объекту сопоставлен прямоугольник – символическое изображение ячейки. Около ячейки пишется её имя, если оно есть, а внутри – значение, содержащееся в ячейке.

Если переменная является ссылочной, из неё выходит стрелочка-ссылка. Она кончается на том объекте, на который указывает ссылка. Если в ссылочной переменной содержится значение `null` (ссылка “в никуда”, адрес==0), рисуется “висящая” короткая стрелка, у которой находится надпись “`null`”. Отметим, что в Java символом равенства является “==”, а не символ “=”, который используется для оператора присваивания.

Если ссылка перещёлкивается, то либо создаётся новый рисунок (в книжке), либо перечёркивается крестиком прежняя стрелочка и рисуется новая (на листе бумаги или на доске). При новом перещёлкивании эта “старая” стрелка перечёркивается двумя крестиками, а “более свежая”, которая была перещёлкнута – одним крестиком, и так далее. Такая система обозначений позволяет наглядно представить, что происходит при работе со ссылками, и не запутаться в том, куда они указывают и с какими ссылками в какой момент связаны динамически создаваемые объекты.

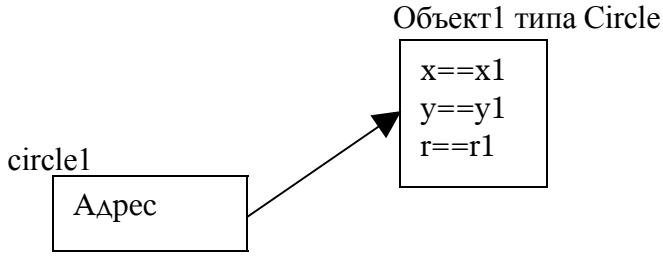
При выполнении оператора `new` , после которого указан вызов конструктора, динамически выделяется новая безымянная ячейка памяти, имеющая тип, соответствующий типу конструктора, а сам конструктор после окончания работы возвращает адрес этой ячейки. Если у нас в левой части присваивания стоит ссылочная переменная, то в результате ссылка “перещёлкивается” на динамически созданную ячейку.

Рассмотрим этот процесс подробнее. Будем считать, что сначала в ячейке circle1 типа Circle хранится нулевой адрес (значение ссылки равно `null`). Будем изображать это как стрелку в никуда с надписью `null`.



После оператора

```
circle1=new Circle(x1,y1,r1);
```
в динамически выделенной безымянной ячейке памяти будет создан объект-окружность с координатами центра x1, y1 и радиусом r1 (это какие-то значения, конкретная величина которых в данном случае не имеет значения):



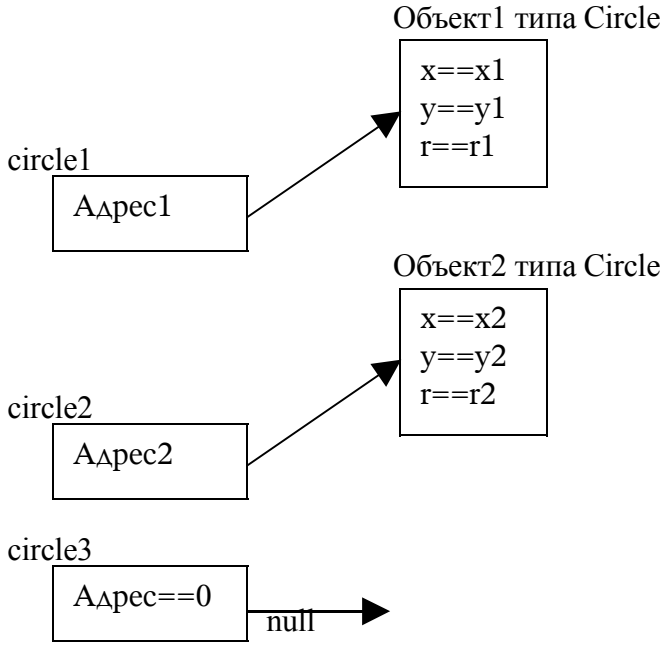
Поля объекта доступны через ссылку как по чтению, так и по записи. До тех пор, пока ссылочная переменная circle1 содержит адрес Объекта1, имя circle1 является псевдонимом, заменяющим имя этого объекта. Его можно использовать так же, как имя обычной переменной в любых выражениях и операторах, не связанных с изменением адреса в переменной circle1.

Поскольку между ссылочными переменными одного типа разрешены присваивания, переменная по ходу программы может сменить объект, на который она ссылается. Если circle2 также имеет тип Circle, то допустимы присваивания вида

```
circle2= circle1;
```
Такие присваивания изменяют адреса в ячейках ссылочных переменных, но не меняют содержания объектов.

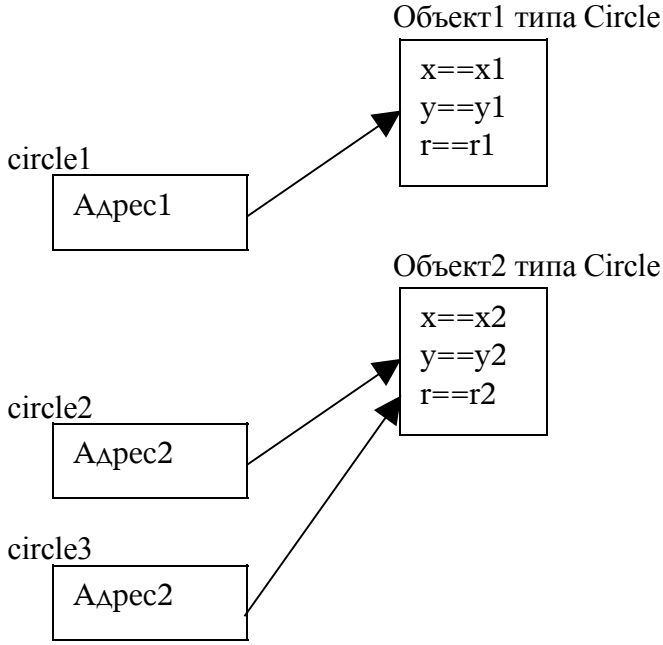
Рассмотрим следующий участок кода:

```
Circle circle1=new Circle(x1,y1,r1);
Circle circle2=new Circle(x2,y2,r2);
Circle circle3;
```
Ему соответствует следующий рисунок:



Проведём присваивание

```
circle3=circle2;
```
В результате получится такая картинка:

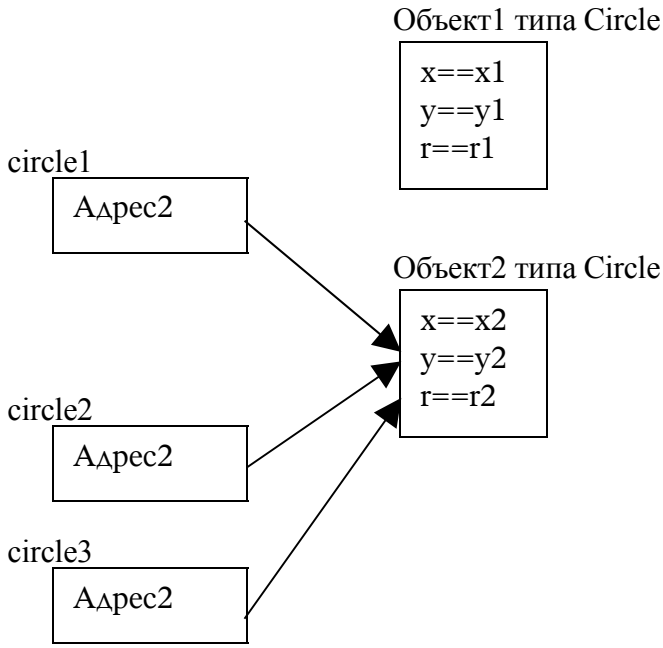


Обе переменные, как circle2, так и circle3, теперь ссылаются на один и тот же объект – в них находится один и тот же Адрес2. То есть оба имени – синоним имени Объекта2. Напомним, что сам объект, как все динамически создаваемые величины, безымянный. Таким образом, circle2.x даст значение x2, точно так же, как и circle3.x. Более того, если мы изменим значение circle2.x, это приведёт к изменению circle3.x – ведь это одно и то же поле x нашего Объекта2.

Рассмотрим теперь, что произойдёт при присваивании

```
circle1=circle2;
```

Этот случай отличается от предыдущего только тем, что переменная circle1 до присваивания уже была связана с объектом.



В результате у Объекта2 окажется сразу три ссылочные переменные, которые с ним связаны, и имена которых являются его псевдонимами в данном месте программы: circle1,

circle2 и circle3. При этом программная связь с Объектом1 окажется утеряна – он занимает место в памяти компьютера, но программный доступ к нему невозможен, поскольку адрес этого объекта программой утерян. Таким образом, он является бесполезным и напрасно занимает ресурсы компьютера.

Про такие ячейки памяти говорят, что они являются *мусором*. В Java предусмотрен механизм высвобождения памяти, занятой такими бесполезными объектами. Он называется сборкой мусора (garbage collection) и работает автоматически. Этим в фоновом режиме занимается специальная часть виртуальной Java-машины, *сборщик мусора*. При программировании на Java, отличие от таких языков как C/C++ или Object PASCAL, программисту нет необходимости самому заботиться о высвобождении памяти, занятой под динамически создаваемые объекты.

Следует подчеркнуть, что намеренная потеря связи ссылочной переменной с ненужным уже объектом – это одно, а непреднамеренная – совсем другое. Если вы не планировали потерю связи с объектом, а она произошла, это логическая ошибка. И хотя она не приведёт к зависанию программы или её неожиданному закрытию, такая программа будет работать не так, как вы предполагали, то есть неправильно или не совсем правильно. Что иногда ещё опасней, так как ошибку можно не заметить или, если заметили, очень трудно понять её причину.

2.3. Проекты NetBeans. Пакеты. Уровни видимости классов. Импорт классов

Современное программное обеспечение построено по модульному (блочному) принципу. Программы давно перестали состоять из одного файла. Поэтому вместо слова “программа” лучше употреблять слово “проект”. Тем более что термин “программа”, как уже говорилось, неоднозначен.

Идеология Java подразумевает работу в компьютерных сетях и возможность подгрузки в необходимый момент через сеть требуемых классов и ресурсов, в которых нуждается программа, и которые не были до того загружены. Для обеспечения такого рода работы приложения Java разрабатываются и распространяются в виде большого числа независимых классов. Однако такой способ разработки приводит к чрезвычайно высокой фрагментации программы. Даже небольшие учебные проекты часто состоят из десятков классов, а реальные проекты – из сотен. При этом каждому общедоступному (public) классу соответствует свой файл, имеющий то же имя. Для того чтобы справиться с таким обилием файлов, в Java предусмотрено специальное средство группировки классов, называемое пакетом (package). Пакеты обеспечивают независимые пространства имён (namespaces), а также ограничение доступа к классам.

Классы всегда задаются в каком-либо пакете. Пакеты могут быть вложенными с произвольным уровнем вложения (ограничивается только операционной системой и, как правило, не менее 256). Каждому пакету соответствует папка с исходными кодами соответствующих классов, при этом пакету, вложенному в другой, соответствует папка, вложенная в другую.

Для того чтобы поместить класс в пакет, требуется продекларировать имя пакета в начале файла, в котором объявлен класс, в виде

```
package имя_пакета;
```

Кроме того, необходимо поместить исходный код класса в папку, соответствующую пакету.

Если декларация имени пакета отсутствует, считается, что класс принадлежит пакету с именем default.

Вложенным пакетам соответствуют составные имена. Например, если мы имеем пакет с именем pkg1, в который вложен пакет с именем pkg2, в который вложен пакет с именем pkg3, то объявление, что класс с именем MyClass1 находится в пакете pkg3, будет выглядеть

как

```
package pkg1.pkg2.pkg3;
class MyClass1 {
….
}
```

Внутри фигурных скобок должно содержатся описание класса. Оно заменено многоточием.

В качестве разделителя имён пакетов в программе используется точка независимо от типа операционной системы. Хотя в разных операционных системах вложенность папок будет обозначаться по-разному:

в *MS Windows*®: pkg1\pkg2\pkg3\
в *Unix* и *Linux*: pkg1/pkg2/pkg3/
в *Mac OS*: pkg1:pkg2:pkg3:

При создании проекта в среде NetBeans помещение класса в пакет происходит автоматически.

При декларации класса можно указывать, что он общедоступен, с помощью *модификатора доступа* public:

```
public class MyClass2 {
….
}
```

В этом случае возможен доступ к данному классу из других пакетов.

Если же модификатор public отсутствует, как в случае MyClass1, то доступ к классу разрешён только из классов, находящихся с ним в одном пакете. Про такие файлы говорят, что у них *пакетный* вариант доступа (в C++ аналогичный вид доступа называется “дружественным” - friend).

В файле .java можно располагать только один общедоступный класс и произвольное число классов с пакетным уровнем видимости.

Класс может использовать общедоступные (public) классы из других пакетов напрямую, с указанием полного имени класса в пространстве имён, включающего имя пакета. Например, доступ к классу MyClass2 в таком варианте осуществляется как

```
pkg1.pkg2.pkg3.MyClass2
```

Для того, чтобы задать переменную объектного типа, надо до того указать её класс. В нашем случае это будет выглядеть так:

```
pkg1.pkg2.pkg3.MyClass2 myObject;
```

Для того чтобы отличать имена классов от имён пакетов, в Java принято имена пакетов писать только строчными буквами, имена классов начинать с заглавной буквы, а имена полей данных (в том числе имена объектных переменных) и методов начинать со строчной буквы. Если имя класса, поля данных или метода (но не пакета!) состоит из нескольких слов, каждое новое слово принято или писать с заглавной буквы. Новое слово также можно отделять от предыдущего символом подчёркивания. Таким образом, из названия `javax.swing.JMenuItem` понятно, что `javax` и `swing` – пакеты, а `JMenuItem` – имя класса.

Существует способ доступа к именам из другого пакета “напрямую”, без указания каждый раз полного пути в пространстве имён. Это делается с помощью оператора import. Если мы хотим импортировать имя класса `MyClass2` из пакета `pkg3`, то после объявления имени нашего пакета (например, `mypack1`), но до объявления нашего класса (например, `MyClass3`) пишется

```
import pkg1.pkg2.pkg3.MyClass2;
```

При этом в классе `MyClass3` имя `MyClass2` можно использовать напрямую, без указания

перед ним имени пакета `pkg1.pkg2.pkg3`. При этом задание переменной будет выглядеть так:

```
MyClass2 myObject;
```

Но если мы импортируем пакеты, содержащие классы с одинаковыми именами, требуется указание полного имени класса – квалифицированного именем пакета.

Если мы хотим импортировать имена всех классов из пакета, в операторе import после имени пакета вместо имени класса следует написать * . Пример:

```
import pkg1.pkg2.pkg3.*;
```

Заметим, что импортируются только имена файлов, находящихся точно на уровне указанного пакета. Импорта имён из вложенных в него пакетов не происходит. Например, если записать `import pkg1.*;` или `import pkg1.pkg2.*;` , то имя класса `MyClass2` не будет импортировано, так как хотя он и находится внутри `pkg1` и `pkg2`, но не непосредственно, а в пакете `pkg3`.

Имеется одно исключение из правила для импорта: классы ядра языка Java, содержащиеся в пакете `java.lang`, импортируются автоматически без указания имени пакета.

Пример: объявление графического объекта g, имеющего тип , может проводиться тремя способами.

Во-первых, напрямую, с указанием имени пакета и класса:

```
java.awt.Graphics g;
```

Во-вторых, с предварительным импортом класса `Graphics из пакета java.awt` и последующим указанием имени этого класса без его спецификации именем пакета:

```
import java.awt.Graphics;
…
Graphics g;
```

В-третьих, с предварительным импортом всех классов (в том числе `Graphics`) `из пакета java.awt` и последующим указанием имени этого класса без его спецификации именем пакета:

```
import java.awt.*;
…
Graphics g;
```

2.4. Базовые пакеты и классы Java

В пакете java находятся следующие пакеты и классы:

Пакет, класс	Краткое описание
java.applet	Поддержка работы с апплетами.
java.awt	Базовый пакет работы с графическим пользовательским интерфейсом (Abstract Window Toolkit - Абстрактный Инструментарий графического Окна).
java.beens	Поддержка компонентной модели JavaBeans.
java.io	Поддержка базовых средств ввода-вывода.
java.lang	Содержит базовые классы языка Java. Автоматически импортируется в любую программу без указания имени пакета.
java.lang.reflect	Поддерживает механизм доступа к классам как метаобъектам, обеспечивающий возможность динамического выяснения программой, какие возможности поддерживает класс. Данный механизм называется reflection - “отражение”.
java.lang.Math	Класс, обеспечивающий поддержку основных математических функций, а также простейшее средство генерации псевдослучайных чисел.

java.math	Поддержка вычислений с целыми числами произвольной длины, а также числами в формате с плавающей точкой произвольной точности.
java.net	Поддержка работы в Интернет, а также соединений через сокеты (sockets).
java.nio	Содержит классы и пакеты для поддержки сетевых соединений, расширяющие возможности пакета java.io . В частности, содержит классы контейнеров (буферов) для создания списков с данными различных примитивных типов, а также пакеты channels (“каналы соединения, коннекции”) и charset (“национальный набор символов”). Пакет charset обеспечивает поддержку перекодирования из символов Unicode в последовательность байт для передачи через канал связи, а также обратное преобразование.
java.rmi	Поддержка вызовов удалённых методов.
java.security	Поддержка специальных средств, обеспечивающих безопасность приложения, в том числе при работе в компьютерных сетях (списки доступа, сертификаты безопасности, шифрование и т.д.).
java.sql	Поддержка SQL-запросов к базам данных.
java.text	Поддержка специальных средств, обеспечивающих локализацию программ – классы, обеспечивающие настройки для работы с текстом, датами, текстовым представлением чисел. Кроме того, содержит средства для зависимого от локализации сравнения строк.
java.util	Содержит важнейшие классы для работы со структурами данных (в том числе – необходимых для работы с событиями и датами). В частности – поддержку работы с массивами (сортировка, поиск), а также расширенные средства генерации псевдослучайных чисел.
java.util.jar	Поддержка работы с jar-архивами (базовым видом архивов в Java).
java.util.zip	Поддержка работы с zip-архивами.

Пакет javax обеспечивает поддержку новых возможностей, введённых в Java 2. В нём находятся следующие пакеты:

Пакет, класс	**Краткое описание**
javax.accessibility	Обеспечивает настройку специальных возможностей представления информации для людей с плохим зрением, слухом и т.п., а также других случаев, когда требуется специализированный доступ для управления информационными объектами.
javax.activity	Вспомогательный пакет для работы с компонентами.
javax.crypto	Поддержка шифрования-расшифровки данных.
javax.imageio	Поддержка работы с изображениями (ввод-вывод).
javax.management	Поддержка работы с управляющими компонентами (MBean – Management Bean).
javax.naming	Поддержка работы с пространством имён компонентов.
javax.net	Поддержка работы в Интернет, а также соединений через сокеты (sockets). – Расширение возможностей java.net
javax.print	Поддержка работы с печатью документов.
javax.rmi	Поддержка вызовов удалённых методов. – Расширение возможностей java.rmi

javax.security	Поддержка специальных средств, обеспечивающих безопасность приложения. – Расширение возможностей java.security
javax.sound	Поддержка работы со звуковыми потоками и файлами.
javax.sql	Поддержка SQL-запросов к базам данных. – Расширение возможностей java.sql
javax.swing	Библиотека основных графических компонентов в Java 2.
javax.transaction	Поддержка работы с транзакциями.
javax.xml	Поддержка работы с XML документами и парсерами.

Пакет com.sun от фирмы Sun Microsystems в основном обеспечивает расширение возможностей пакета javax. В нём находятся следующие пакеты:

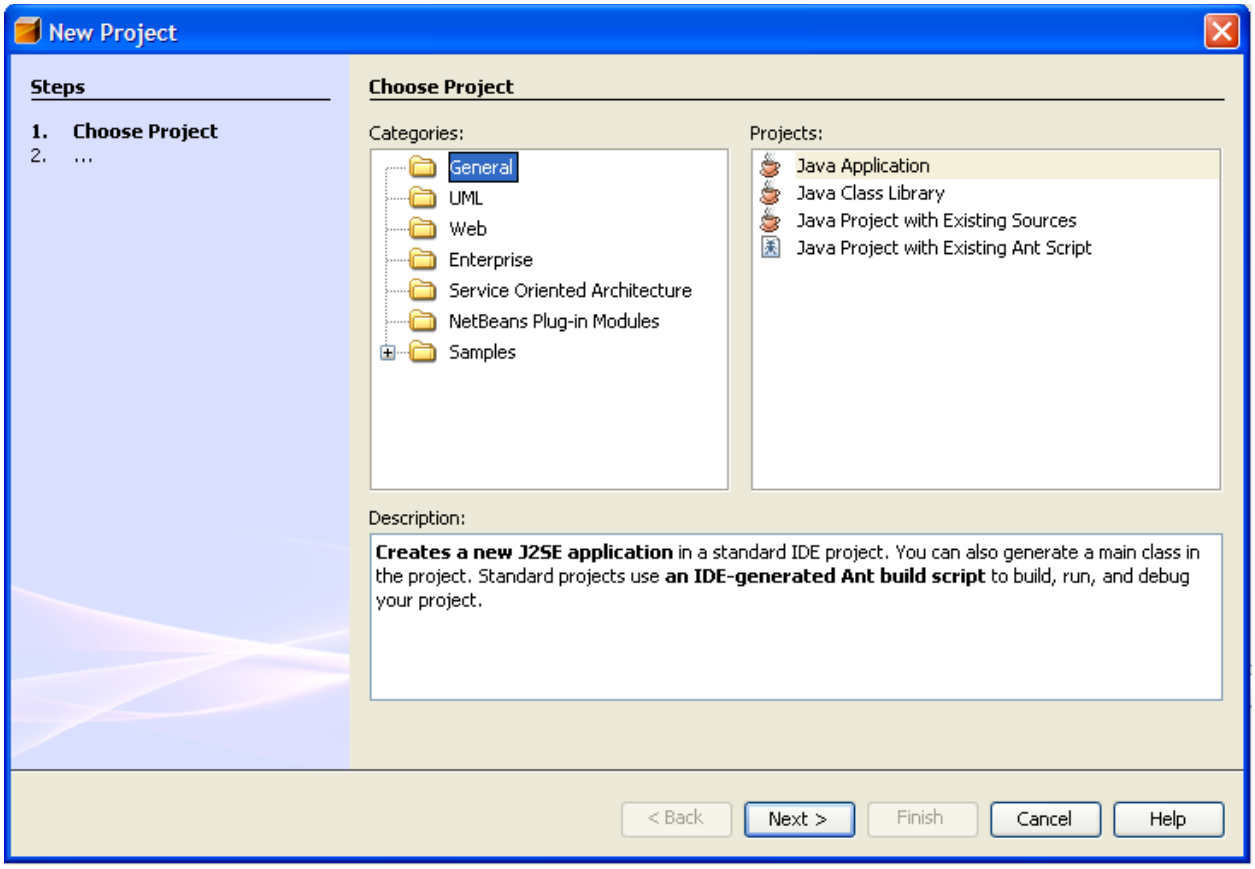
Пакет, класс	Краткое описание
com.sun.accessibility	Дополнение к пакету javax.accessibility
com.sun.beans	Дополнение к пакету java.beens
com.sun.corba	Поддержка работы в компьютерных сетях с базами данных по технологии CORBA (Common Object Request Broker Architecture).
com.sun.crypto	Дополнение к пакету javax.crypto
com.sun.image	Поддержка работы с изображениями
com.sun.imageio	Дополнение к пакету javax.imageio
com.sun.java	Поддержка стилей показа приложений (см.раздел “Внешний вид приложения”), а также служебные утилиты для работы с браузерами и WWW-документами.
com.sun.java_cup	Поддержка технологии JavaCup
com.sun.jlex	Поддержка работы лексического анализатора.
com.sun.jmx	Дополнение к пакету javax.management
com.sun.jndi	Пакет в процессе разработки.
com.sun.management	Дополнение к пакету javax.management
com.sun.media	Поддержка работы со звуком.
com.sun.naming	Дополнение к пакету javax.naming
com.sun.net	Дополнение к пакету javax.net
com.sun.org	Поддержка взаимодействия с сервером Apache, средства работы с базами данных по технологии CORBA.
com.sun.rmi	Дополнение к пакету javax.rmi

В пакете org находятся следующие пакеты, предоставляемые свободным сообществом разработчиков:

Пакет, класс	Краткое описание
org.ietf	Поддержка защищенных соединений по протоколу GSS (Kerberos v5).
org.jdesktop	Менеджер размещения GroupLayout.
org.omg	Средства для использования из программ на Java технологии CORBA, применяемой для создания распределенных объектных приложений.
org.w3c	Интерфейсы для работы с XML-документами в соответствии со спецификацией DOM.
org.xml	Поддержка работы с XML-документами.

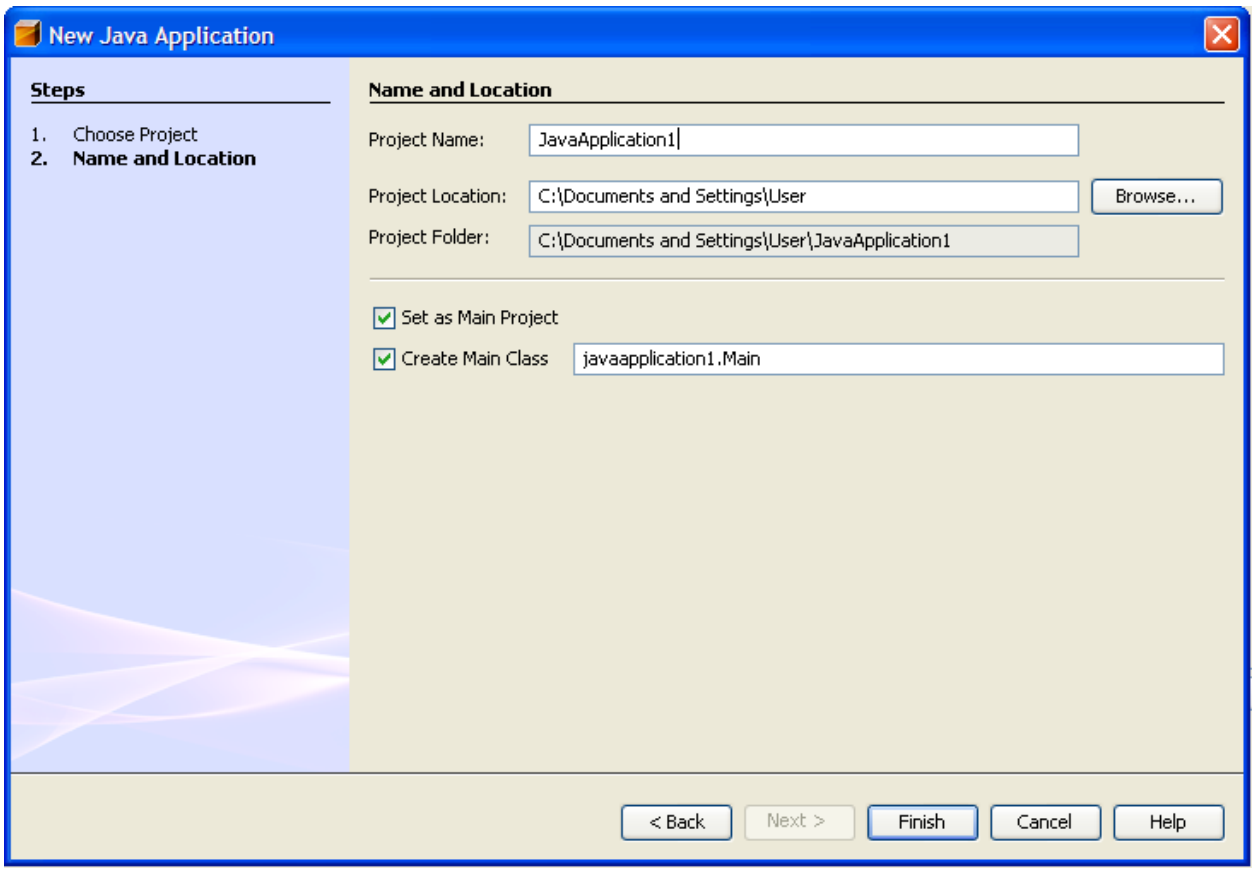
2.5. Создание в NetBeans простейшего приложения Java

Создадим с помощью среды NetBeans приложение Java. Для этого запустим интегрированную среду разработки (IDE) NetBeans , и выберем в главном меню **File/New Project…** В открывшемся диалоге выберем **General / Java Application / Next>**



Создание нового проекта. Шаг 1.

После чего можно нажимать кнопку **Finish** – значения по умолчанию для начала менять не стоит. Это можно будет делать потом, когда вы освоитесь со средой разработки.



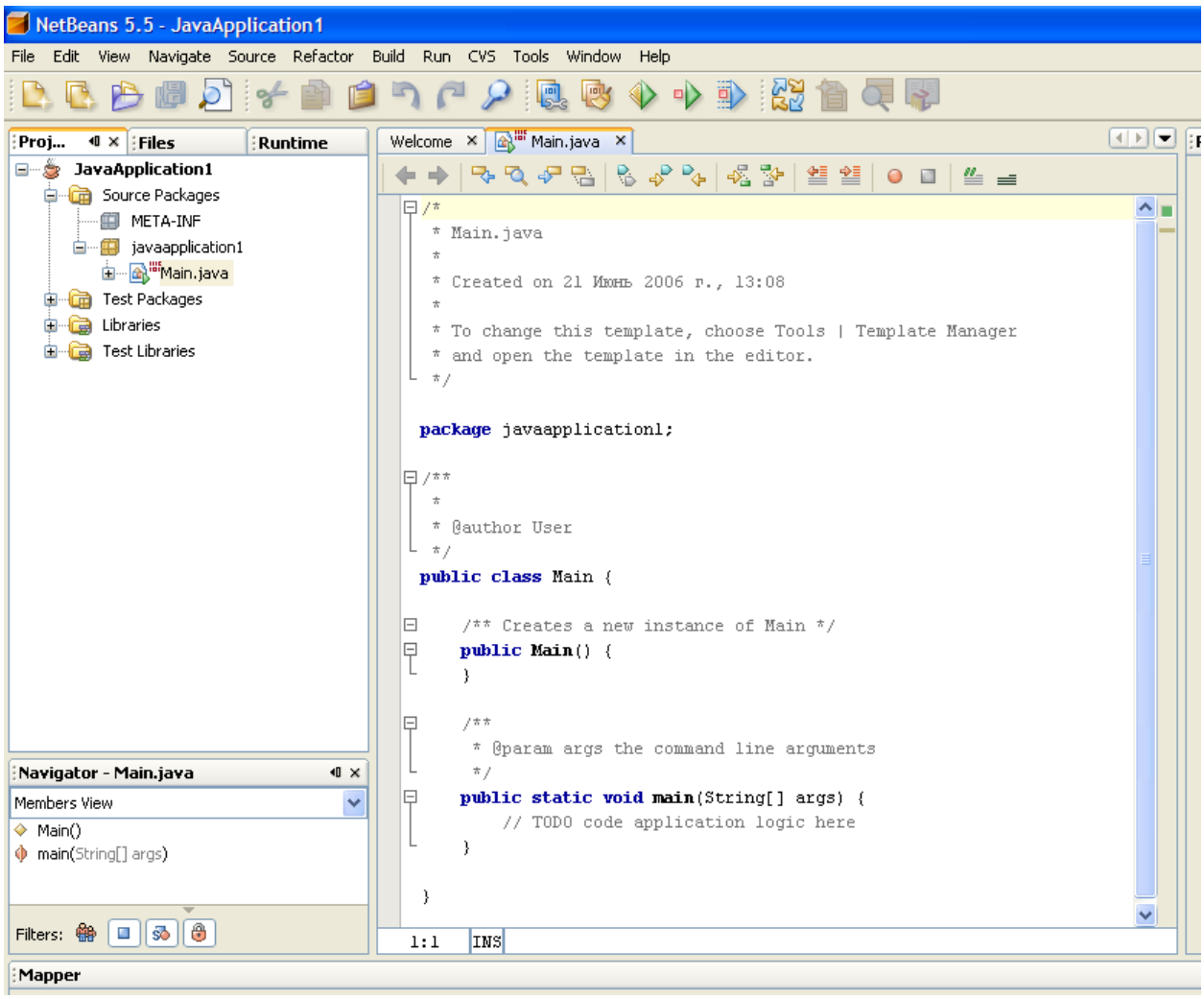
Создание нового проекта. Шаг 2.

На следующем рисунке показано, как выглядит редактирование исходного кода приложения в среде NetBeans.

В левом верхнем окне “Projects” показывается дерево проектов. В нашем случае это дерево для проекта JavaApplication1. Это окно может быть использовано для одновременного показа произвольного числа проектов. По умолчанию все деревья свёрнуты, и нужные узлы следует разворачивать щелчком по узлу с “плюсиком” или двойным щелчком по соответствующему имени.

В правом окне “Source” показывается исходный код проекта.

В левом нижнем окне “Navigator” показывается список имён членов класса приложения – имена переменных и подпрограмм. Двойной щелчок по имени приводит к тому, что в окне редактора исходного кода происходит переход на то место, где задана соответствующая переменная или подпрограмма.



Редактирование исходного кода приложения

Рассмотрим, как выглядит сгенерированный исходный код нашего приложения Java:

```
/*
 * Main.java
 *
 * Created on 21 Июнь 2006 г., 13:08
 *
 * To change this template, choose Tools | Template Manager
 * and open the template in the editor.
 */

package javaapplication1;

/**
 *
 * @author User
 */
public class Main {

    /** Creates a new instance of Main */
    public Main() {
    }
```

```
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }

}
```

Сначала идёт многострочный комментарий /* … */ Он содержит информацию об имени класса и времени его создания.

Затем объявляется, что наш класс будет находиться в пакете javaapplication1. После этого идёт многострочный комментарий /** … */ , предназначенный для автоматического создания документации по классу. В нём присутствует инструкция задания метаданных с помощью выражения @author – информация об авторе проекта для утилиты создания документации javadoc. Метаданные – это некая информация, которая не относится к работе программы и не включается в неё при компиляции, но сопровождает программу и может быть использована другими программами для проверки прав на доступ к ней или её распространения, проверки совместимости с другими программами, указания параметров для запуска класса и т.п. В данном месте исходного кода имя “User” берётся средой разработки из операционной системы по имени папки пользователя. Его следует заменить на имя реального автора, т.е. в нашем случае на “Вадим Монахов”.

Далее следует объявление класса Main, который является главным классом приложения. В нём объявлены две общедоступных (public) подпрограммы. Первой из них является конструктор:

```
    public Main() {
    }
```

Его имя совпадает с именем класса. Он занимается созданием объектов типа Main. Обычно такой конструктор вызывается из метода main, и с его помощью создаётся всего один объект, “олицетворяющий” собой приложение. Но, вообще говоря, таких объектов в простых программах может и не создаваться, как это и происходит в нашем случае.

Все классы и объекты приложения вызываются и управляются из метода main, который объявлен далее и выглядит следующим образом:

```
public static void main(String[] args) {
 }
```

Он является методом класса, и поэтому для его работы нет необходимости в создании объекта, являющегося экземпляром класса Main. Хотя если этот объект создаётся, это происходит во время работы метода main.

Метод main является главным методом приложения и управляет работой запускаемой программы. Он автоматически вызывается при запуска приложения. Параметром `args` этого метода является массив строк, имеющий тип `String[]`.Это параметры командной строки, которые передаются в приложение при его запуске. Слово String означает “Строка”, а квадратные скобки используются для обозначения того, что это массив строк.

После окончания выполнения метода main приложение завершает свою работу.

При объявлении любого метода в Java сначала указывается модификатор видимости, указывающий права доступа к методу, затем другие модификаторы, после чего следует тип возвращаемого методом значения. Если модификатор видимости не указан, считается, что это private (читается “прайвит”)– “закрытый, частный”, не позволяющий доступ к методу из других классов.

Конструкторы представляют особый случай, в них имя типа и имя метода совпадают.

В Java они не считаются методами, хотя в других языках программирования такого тонкого различия не делается.

Далее следует имя метода, после чего в круглых скобках идёт список параметров (аргументов), передаваемых в данный метод при его вызове. После этого в фигурных скобках идёт *тело метода*, то есть его *реализация*– пишется тот алгоритм, который будет выполняться при вызове метода.

В языке Java, как и в C/C++ подпрограммы всегда являются подпрограммами-функциями, возвращающими какое-либо значение. Если надо написать подпрограмму-процедуру, в которой не надо возвращать никакого значения, в C/C++/Java пользуются подпрограммами-функциями с типом возвращаемого значения void – "пустота, пустое пространство". Как и происходит в случае метода main.

Среда NetBeans создаёт заготовку методов – в них имеется пустое тело. Для осуществления методом какой-либо деятельности следует дописать свой собственный код. Напишем традиционный пример – вывод сообщения "Привет!". Для этого вместо комментария

```
// TODO code application logic here
```

("описать тут логику работы приложения") напишем строку вывода текста

```
System.out.println("Привет!");
```

Класс `System`, "система", имеет поле `out`, "наружу". Это объект, предназначенный для поддержки вывода. У него есть метод `println`, предназначенный для вывода текста в режиме консоли.

Консольный ввод-вывод ранее широко применялся в операционных системах, ориентированных на работу в режиме командной строки. При этом основным средством взаимодействия пользователей с программами служила текстовая консоль ("пульт управления"). В ней устройством ввода служила клавиатура, а устройством вывода – окно операционной системы, обеспечивающее вывод текста в режиме пишущей машинки (системным шрифтом с буквами, имеющими одинаковую ширину). Очень много примеров программ в учебных курсах по Java ориентированы на работу в таком режиме. В настоящее время в связи с тем, что подавляющее большинство пользователей работают с программами в графическом режиме, работу в консольном режиме нельзя рассматривать как основную форму ввода-вывода. Тем более, что NetBeans позволяет без особых усилий создавать графический пользовательский интерфейс (GUI – Graphics User Interface) приложения. А консольный режим следует применять только как промежуточный, удобный в отладочном режиме как средство вывода вспомогательной информации.

2.6. Компиляция файлов проекта и запуск приложения

В современных средах разработки используется два режима компиляции – compile ("скомпилировать") и build ("построить"). В режиме "compile" происходит компиляция только тех файлов проекта, которые были изменены в процессе редактирования после последней компиляции. А в режиме "build" перекомпилируются заново все файлы.

Для компиляции проекта следует выбрать в меню среды разработки **Build/ Build Main Project** (или, что то же, клавиша <**F11**>, или на панели инструментов иконка с голубой книжкой и гаечным ключом). При этом будут заново скомпилированы из исходных кодов все классы проекта.

Пункт **Build/ Clean and Build Main Project** (или, что то же, комбинация клавиш <**Shift**> <**F11**>, или на панели инструментов иконка с оранжевой книжкой и веником) удаляет все выходные файлы проекта (очищает папки build и dist), после чего по новой компилируются все классы проекта.

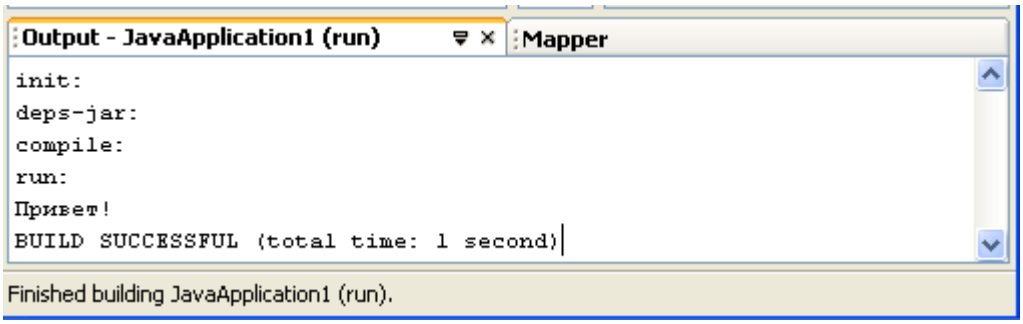
Пункт **Build/ Generate Javadoc for "JavaApplication1"** запускает создание

документации по проекту. При этом из исходных кодов классов проекта выбирается информация, заключённая в документационные комментарии /** … */, и на её основе создаётся гипертекстовый HTML-документ.

Пункт **Build/ Complile "Main.java"** (или, что то же, клавиша <**F9**>) компилирует выбранный файл проекта – в нашем случае файл Main.java, в котором хранятся исходные коды класса Main.

Для того чтобы запустить скомпилированное приложение из среды разработки, следует выбрать в меню среды разработки **Run/ Run Main Project** (или, что то же, клавиша <**F6**>, или на панели инструментов иконка с зелёным и жёлтыми треугольниками). При запуске приложение всегда автоматически компилируется (но не "строится"), так что после внесения изменений для запуска обычно достаточно нажать <**F6**>.

После запуска нашего проекта в выходной консоли, которая находится в нижней части окна проекта, появится служебная информация о ходе компиляции и запуска:



Информация о ходе компиляции и запуска в выходной консоли.

В неё же осуществляется вывод методов `System.out.print` и `System.out.println`.

Метод `System.out.print` отличается от метода `System.out.println` только тем, что в `println` после вывода осуществляется автоматический переход на новую строку, а в `print` продолжается вывод в ту же строку консоли. Поэтому вывод

```
System.out.println("Привет!");
System.out.println("Привет!");
```

Даст текст

```
Привет!
Привет!
```

а

```
System.out.print("Привет!");
System.out.print("Привет!");
```

даст

```
Привет!Привет!
```

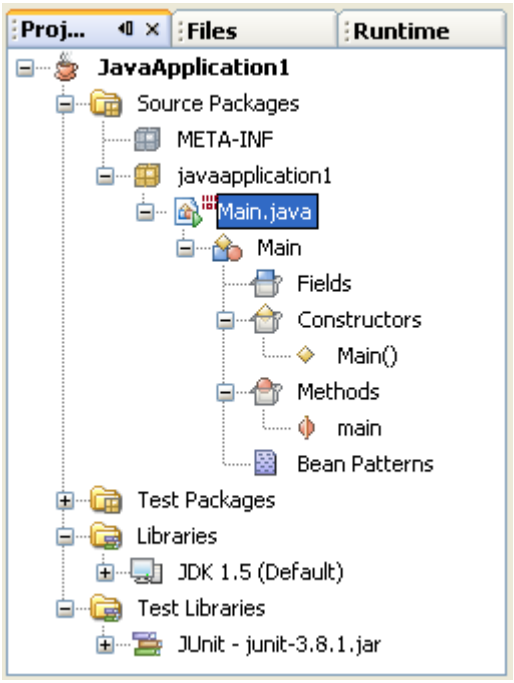
2.7. Структура проекта NetBeans

Рассмотрим, из каких частей состоит проект NetBeans. На рисунке показаны основные элементы, отображаемые в среде разработки.

Это Source Packages (пакеты исходного кода), Test Packages (пакеты тестирования), Libraries (библиотеки) и Test Libraries (библиотеки поддержки тестирования). Ветви дерева представления элементов проекта можно разворачивать или сворачивать путём нажатия на узлы, отмеченные плюсами и минусами. Мы пока будем пользоваться только пакетами исходного кода.

В компонентной модели NetBeans пакеты приложения объединяются в единую конструкцию – модуль. Модули NetBeans являются базовой конструкцией не только для создания приложений, но и для написания библиотек. Они представляют собой оболочку над пакетами (а также могут включать в себя другие модули).

В отличии от библиотек Java скомпилированный модуль – это не набор большого количества файлов, а всего один файл, архив JAR (Java Archive, архив Java). В нашем случае он имеет то же имя, что и приложение, и расширение .jar : это файл JavaApplication1.jar. Модули NetBeans гораздо лучше подходят для распространения, поскольку не только обеспечивают целостность комплекта взаимосвязанных файлов, но и хранят их в заархивированном виде в одном файле, что намного ускоряет копирование и уменьшает объём занимаемого места на носителях.



Отметим не очень удобную особенность NetBeans – после сохранения проекта и закрытия среды разработки не сохраняется конфигурация открытых окон и развёрнутых деревьев проекта - деревья проектов показываются в свёрнутом виде. Поэтому для того, чтобы вновь попасть в режим редактирования исходного кода нашего приложения, в окне Projects, “Проекты” (левом верхнем окне среды разработки) следует развернуть последовательность узлов JavaApplication1/Source Packages/javaapplication1/ . Это делается нажатием на плюсики в соответствующих узлах или двойным щелчком по имени узла. Затем надо сделать двойной щелчок с помощью левой кнопкой мыши по имени узла Main.java, либо с помощью щелчка правой кнопкой мыши по этому имени открыть всплывающее меню и выбрать в нём первый пункт – “Open”.

Имеется и более простой способ. По умолчанию сначала открывается окно Welcome (“Привет”, “Приветствие”). Но среда разработки сохраняет список открытых окон, и в верхней части окна редактирования кода щелчком мыши можно выбрать нужное имя окна. Хотя при этом не видна структура проекта, так что первый способ во многих случаях может быть предпочтительным.

Если вы открываете новый проект, старый не закрывается. И в дереве проектов видны все открытые проекты. То же относится и к списку открытых окон. Это позволяет работать сразу с несколькими проектами, например – копировать в текущий проект участки кода из других проектов. Один из открытых проектов является главным (Main Project) – именно он будет запускаться на исполнение по **Run/ Run Main Project**. Для того, чтобы установить какой-либо из открытых проектов в качестве главного, следует в дереве проектов с помощью правой кнопкой мыши щелкнуть по имени проекта и выбрать пункт меню Set Main Project. Аналогично, для того, чтобы закрыть какой-либо из открытых проектов, следует в дереве проектов с помощью правой кнопкой мыши щелкнуть по имени проекта и выбрать пункт меню Close Project.

Рассмотрим теперь структуру папок проекта NetBeans. По умолчанию головная папка проекта располагается в папке пользователя. В операционной системе Windows® XP проект по умолчанию располагается в папке C:\Documents and Settings\ИмяПользователя\ . Дальнейшее расположение папок и файлов приведено ниже, при этом имена папок выделены жирным шрифтом, а имена вложенных папок и файлов записаны под именами их головных папок и сдвинуты относительно них вправо.

build
  classes
    javaapplication1
      Main.class

… .class
META-INF
dist
javadoc
lib
JavaApplication1jar
README.TXT
nbproject
src
javaapplication1
Main.java
… .java
… .form
META-INF
test
build.xml
manifest.mf

- В папке **build** хранятся скомпилированные файлы классов, имеющие расширение .class.
- В папке **dist** - файлы, предназначенные для распространения как результат компиляции (модуль JAR приложения или библиотеки, а также документация к нему).
- В папке **nbproject** находится служебная информация по проекту.
- В папке **src** - исходные коды классов. Кроме того, там же хранится информация об экранных формах (которые будут видны на экране в виде окон с кнопками, текстом и т.п.). Она содержится в XML-файлах, имеющих расширение .form.
- В папке **test** - сопроводительные тесты, предназначенные для проверки правильности работы классов проекта.

Приведём перевод файла README.TXT, находящегося в папке **dist** - там же, где архив JAR, предназначенный для распространения как файл приложения:

```
=======================
ОПИСАНИЕ ВЫВОДА КОМПИЛЯЦИИ
=======================
Когда Вы компилируете проект приложения Java, которое имеет главный класс, среда разработки (IDE)
автоматически копирует все файлы JAR-архивов, указанные в classpath ваших проектов,
в папку dist/lib. Среда разработки также автоматически прибавляет путь к каждому из этих архивов в
файл манифеста приложения (MANIFEST.MF).

Чтобы запустить проект в режиме командной строки, зайдите в папку dist и
наберите в режиме командной строки следующий текст:
    java -jar "JavaApplication3.jar"

Чтобы распространять этот проект, заархивируйте папку dist (включая папку lib), и распространяйте ZIP-
архив.

Замечания:

* Если два JAR-архива, указанные в classpath ваших проектов, имеют одинаковое имя, в папку lib будет
скопирован только первый из них.
* Если в classpath указана папка с классами или ресурсами, ни один из
элементов classpath не будет скопирован в папку dist.
* Если в библиотеке, указанной в classpath, также имеется элемент classpath, указанные в нём
элементы должны быть указаны в пути classpath времени выполнения проектов.
* Для того чтобы установить главный класс в стандартном проекте Java, щёлкните правой кнопкой
мыши в окне Projects и выберите Properties. Затем выберите Run и введите данные
```

о названии класса в поле Main Class. Кроме того, Вы можете вручную ввести название класса в элементе Main-Class манифеста.

2.8. Создание в NetBeans приложения Java с графическим интерфейсом

Экранной формой называется область, которая видна на экране в виде окна с различными элементами - кнопками, текстом, выпадающими списками и т.п. А сами эти элементы называются компонентами.

Среды, позволяющие в процессе разработки приложения в интерактивном режиме размещать на формы компоненты и задавать их параметры, называются RAD-средами. RAD расшифровывается как Rapid Application Development - быстрая разработка приложений.

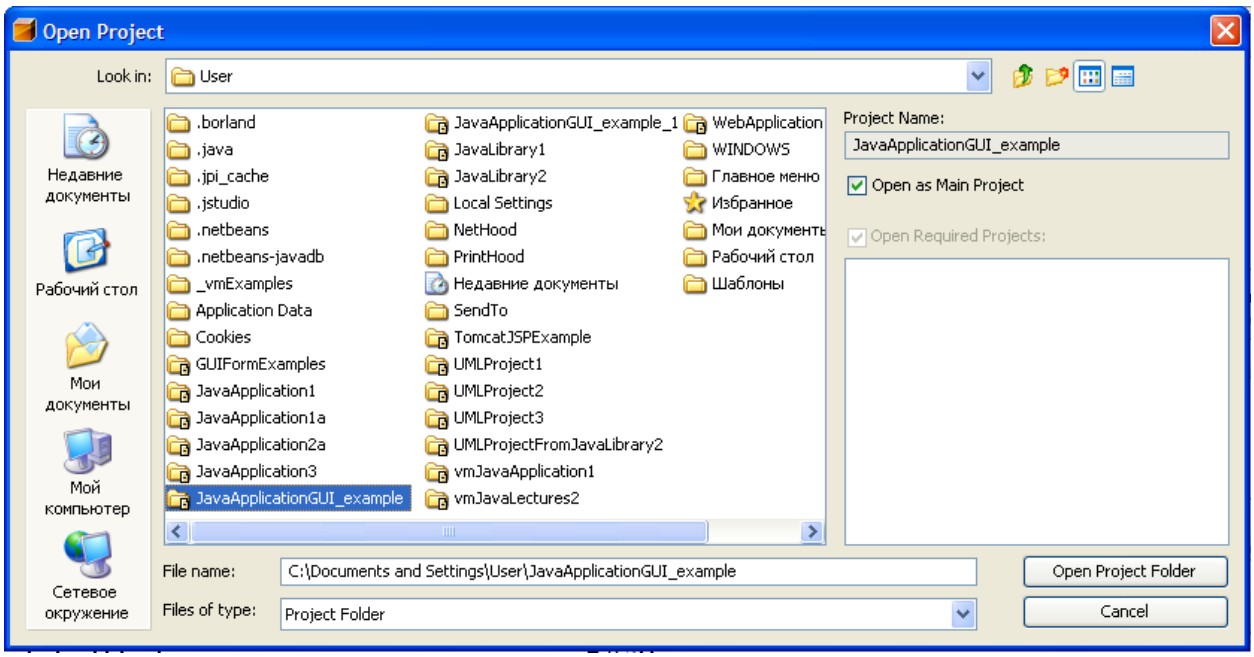
В NetBeans и других современных средах разработки такой процесс основан на объектной модели компонентов, поэтому он называется Объектно-Ориентированным Дизайном (OOD – Object-Oriented Design).

NetBeans является RAD-средой и позволяет быстро и удобно создавать приложения с развитым графическим пользовательским интерфейсом (GUI). Хотя языковые конструкции Java, позволяющие это делать, не очень просты, на начальном этапе работы с экранными формами и их элементами нет необходимости вникать в эти тонкости. Достаточно знать основные принципы работы с такими проектами.

С точки зрения автора изучение того, как создавать приложения с графическим интерфейсом, весьма важно для начинающих программистов, и это следует делать с самых первых шагов по изучению Java.

Во-первых, с самого начала осваивается создание полноценных приложений, которые можно использовать в полезных целях. Трудно месяцами изучать абстрактные концепции, и только став профессионалом иметь возможность сделать что-то такое, что можно показать окружающим. Гораздо интереснее и полезнее сразу начать применять полученные знания на практике.

Во-вторых, такой интерфейс при решении какой-либо задачи позволяет лучше сформулировать, какие параметры надо вводить, какие действия и в какой последовательности выполнять, и что в конце концов получается. И отобразить всё это на экране: вводимым параметрам будут соответствовать пункты ввода текста, действиям – кнопки и пункты меню, результатам – пункты вывода текста.



Пример открытия проекта с существующим исходным кодом.

В NetBeans 5.0 имелся хороший пример GUI-приложения, однако в NetBeans 5.5 он отсутствует. Поэтому для дальнейшей работы следует скопировать аналогичный пример с сайта автора или сайта, на котором выложен данный учебный курс. Пример называется JavaApplicationGUI_example.

Сначала следует распаковать zip-архив, и извлечь находящуюся в нём папку с файлами проекта в папку с вашими проектами (например, C:\Documents and Settings\User). Затем запустить среду NetBeans, если она не была запущена, и закрыть имеющиеся открытые проекты, чтобы они не мешали. После чего выбрать в меню File/Open Project, либо или на панели инструментов иконку с открывающейся фиолетовой папочкой, либо нажать комбинацию клавиш <Shift>+<Ctrl>+O. В открывшемся диалоге выбрать папку **JavaApplicationGUI_example** (лучше в неё не заходить, а просто установить выделение на эту папку), после чего нажать кнопку Open Project Folder.

При этом, если не снимать галочку “Open as Main Project”, проект автоматически становится главным.

В окне редактора исходного кода появится следующий текст:

```
/*
 * GUI_application.java
 *
 * Created on 22 Июня 2006 г., 13:41
 */

package java_gui_example;

/**
 *
 * @author  Вадим Монахов
 */

public class GUI_application extends javax.swing.JFrame {
    /**
     * Creates new form GUI_application
     */
    public GUI_application() {
        initComponents();
     }
```

```
    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    +Generated Code

    private void exitMenuItemActionPerformed(java.awt.event.ActionEvent evt)
    {
        System.exit(0);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new GUI_application().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JMenuItem aboutMenuItem;
    private javax.swing.JMenuItem contentsMenuItem;
    private javax.swing.JMenuItem copyMenuItem;
    private javax.swing.JMenuItem cutMenuItem;
    private javax.swing.JMenuItem deleteMenuItem;
    private javax.swing.JMenu editMenu;
    private javax.swing.JMenuItem exitMenuItem;
    private javax.swing.JMenu fileMenu;
    private javax.swing.JMenu helpMenu;
    private javax.swing.JMenuBar menuBar;
    private javax.swing.JMenuItem openMenuItem;
    private javax.swing.JMenuItem pasteMenuItem;
    private javax.swing.JMenuItem saveAsMenuItem;
    private javax.swing.JMenuItem saveMenuItem;
    // End of variables declaration

}
```

Поясним некоторые его части. Указание пакета `java_gui_example`, в котором будет располагаться код класса приложения, нам уже знакомо. Декларация самого класса `GUI_application` в данном случае несколько сложнее, чем раньше:

```
public class GUI_application extends javax.swing.JFrame
```

Она означает, что задаётся общедоступный класс `GUI_application`, который является наследником класса `JFrame`, заданного в пакете `swing`, вложенном в пакет `javax`. Слово `extends` переводится как “расширяет” (класс-наследник всегда расширяет возможности класса-прародителя).

Общедоступный конструктор `GUI_application()` создаёт объект приложения и инициализирует все его компоненты, методом `initComponents()`, автоматически генерируемом средой разработки и скрываемом в исходном коде узлом `+Generated Code`. Развернув узел, можно увидеть реализацию этого метода, но изменить код нельзя. Мы не будем останавливаться на том, что в нём делается.

Далее следует закрытый (private) метод

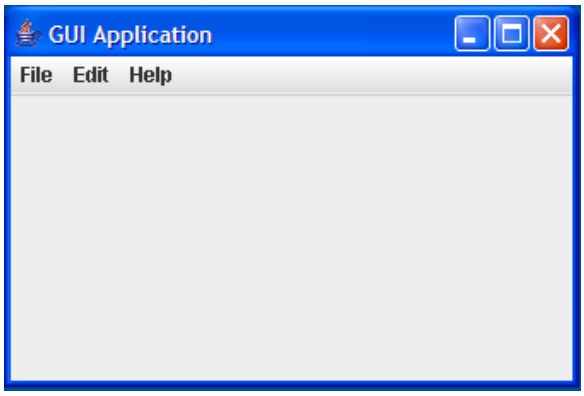
```
private void exitMenuItemActionPerformed
```

Он будет обсуждаться чуть позже. Метод

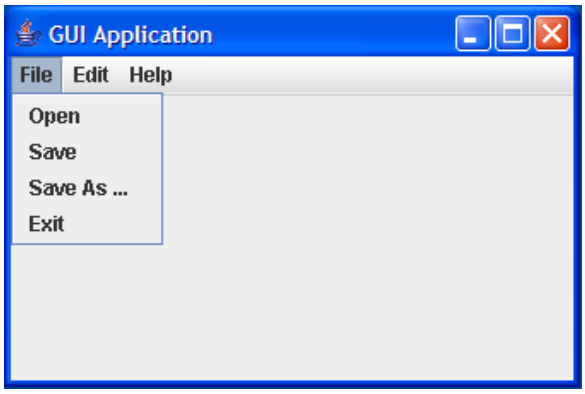
```
public static void main(String[] args)
```

нам уже знаком – это главный метод приложения. Он является методом класса нашего приложения и автоматически выполняется Java-машиной при запуске приложения. В данном примере метод создаёт экранную форму приложения и делает её видимой. Для того, чтобы понять, как это делается, потребуется изучить довольно много материала в рамках данного курса.

Далее следует область объявления компонентов– пунктов меню нашей формы. Она автоматически создаётся в исходном коде редактором экранных форм и недоступна для изменения в редакторе исходного кода.



Запущенное приложение.



Приложение с раскрытым меню.

При запуске приложения экранная форма выглядит так, как показано на рисунке. В ней уже имеется заготовка меню, которое способно разворачиваться и сворачиваться, и даже работает пункт Exit – “Выход”. При нажатии на него происходит выход из приложения.

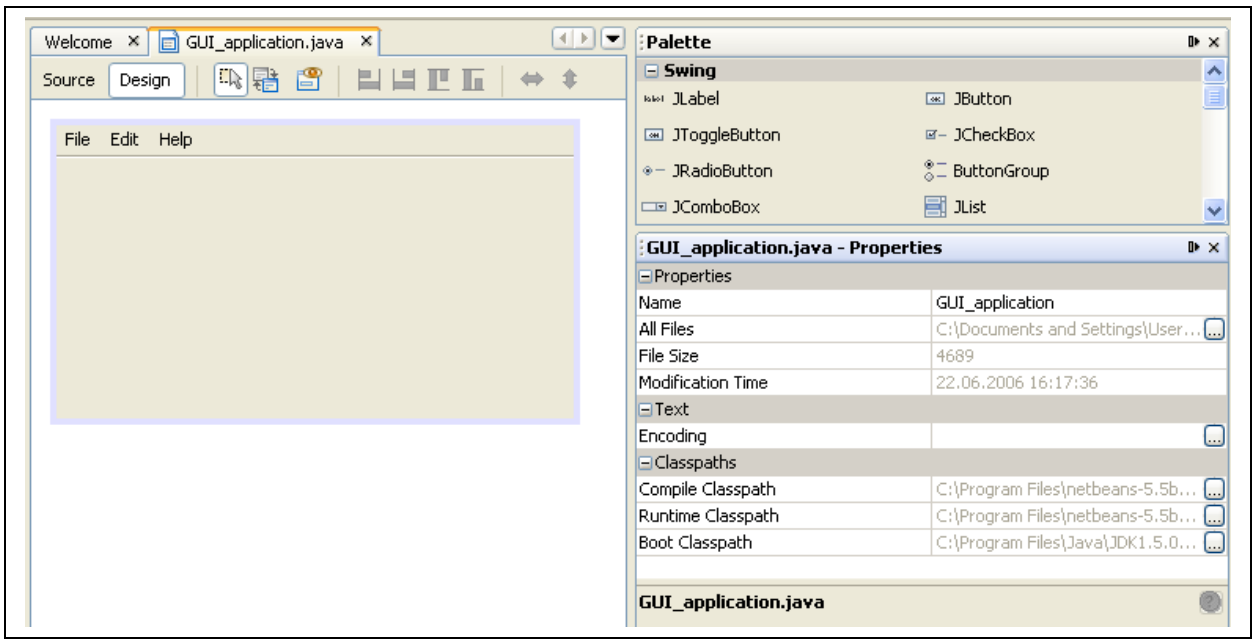
Именно за нажатие на этот пункт меню несёт ответственность оператор `exitMenuItemActionPerformed`. При проектировании экранной формы он назначен в качестве *обработчика события* – подпрограммы, которая выполняется при наступлении события. В нашем случае событием является выбор пункта меню Exit, и при этом вызывается обработчик `exitMenuItemActionPerformed`. Внутри него имеется всего одна строчка

```
System.exit(0);
```

Она вызывает прекращение выполнения метода main и выход из приложения с нулевым кодом завершения. Как правило, ненулевой код завершения возвращают при аварийном завершении приложения для того, чтобы по его значению можно было выяснить причины “вылета” программы.

2.9. Редактор экранных форм

Нажмём закладку Design (“дизайн”) в левой верхней части редактора исходного кода. При этом мы переключимся из режима редактирования исходного кода (активна закладка Source – “исходный код”) в режим редактирования экранной формы, как это показано на рисунке.



Редактирование экранной формы.

Вместо исходного кода показывается внешний вид экранной формы и находящиеся на ней компоненты. Справа от окна, в котором показывается экранная форма в режиме редактирования, расположены окна Palette (“палитра”) палитры компонентов и окно Properties (“свойства”) показа и редактирования свойств текущего компонента.

Свойство – это поле данных, которое после изменения значения может проделать какое-либо действие. Например, при изменении значения ширины компонента отрисовать на экране компонент с новой шириной. “Обычное” поле данных на такое не способно. Таким образом, свойство – это “умное поле данных”.

Палитра компонентов предназначена для выбора типа компонента, который нужен программисту для размещения на экранной форме. Например, добавим на нашу форму компонент типа JButton (сокращение от Java Button – “кнопка Java”). Для этого щёлкнем мышью по пункту JButton на палитре и передвинем мышь в нужное место экранной формы. При попадании мыши в область экранной формы на ней появляется кнопка стандартного размера, которая передвигается вместе с мышью. Щелчок в нужном месте формы приводит к тому, что кнопка остаётся в этом месте. Вокруг неё показываются рамка и маленькие квадратики, обозначающие, что наш компонент является выделенным. Для него осуществляется показ и редактирование свойств в окне Properties.

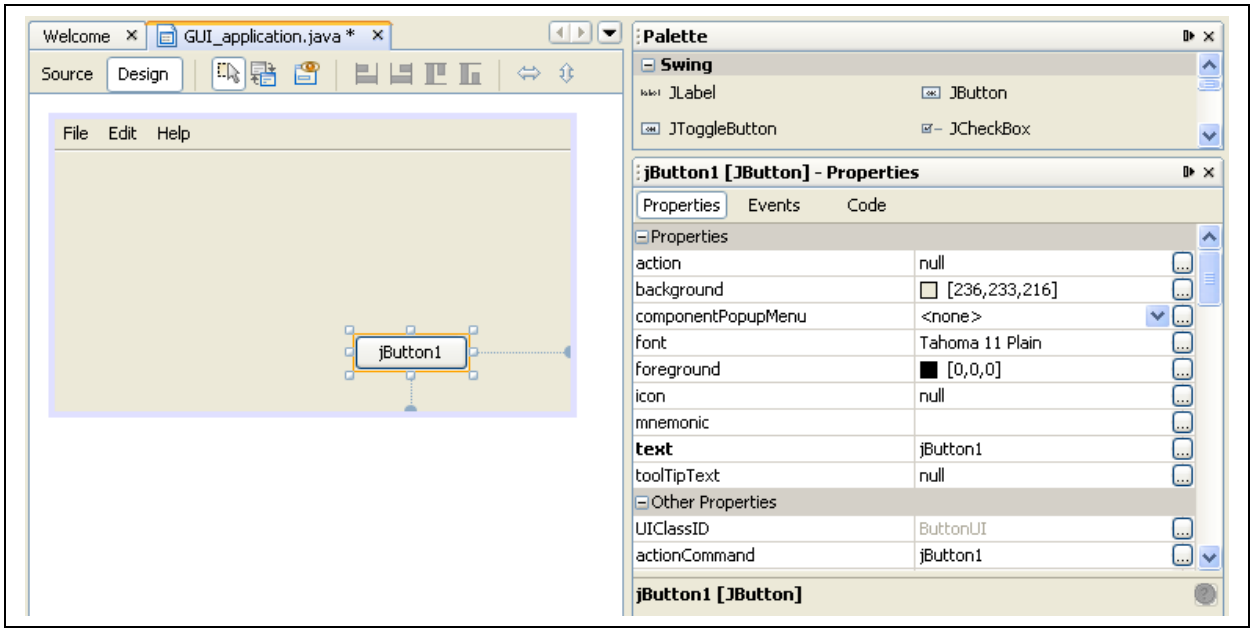
Кроме того, от выделенного компонента исходят линии, к которым идет привязка для задания положения компонента на форме.

По умолчанию надписи на компонентах задаются как имя типа, после которого идёт номер компонента. Но вместо заглавной буквы, в отличие от имени типа, используется строчная. Поэтому первая кнопка будет иметь надпись jButton1, вторая – jButton2, и так далее. Такие же имена будут приобретать автоматически создаваемые в исходном коде переменные, соответствующие кнопкам.

Изменить надпись на кнопке можно несколькими способами. Во-первых, сделав по ней двойной щелчок, и отредактировав текст. Во-вторых, перейдя в окно Properties, изменив значение свойства Text и нажав <Enter> для завершения ввода. В-третьих, изменив аналогичным образом свойство label. Наконец, можно в окне Properties отредактировать текст не в однострочном поле ввода значений для свойств Text или label, а открыв многострочный редактор путём нажатия на кнопку, находящуюся справа от пункта редактирования значения свойства. Однако многострочность редактора не помогает сделать

надпись на кнопке многострочной.

Введём на кнопке надпись “OK” – используем эту кнопку для выхода из программы.



Редактирование свойств компонента

Размер компонента задаётся мышью путём хватания за рамку и расширения или сужения по соответствующим направлениям. Установка на новое место – перетаскиванием компонента мышью.

Некоторые свойства выделенного компонента (его размер, положение, текст) можно изменять непосредственно в области экранной формы. Однако большинство свойств просматривают и меняют в окне редактирования свойств. Оно состоит из двух столбцов: в левом показываются имена свойств, в правом – их значения. Значения, стоящие в правом столбце, во многих случаях могут быть отредактированы непосредственно в ячейках таблицы. При этом ввод оканчивается нажатием на <Enter> или выходом из редактируемой ячейки, а отменить результаты неоконченного ввода можно нажатием <Escape>.

В правой части каждой ячейки имеется кнопка с надписью “…” – в современных операционных системах принято добавлять три точки в названии пунктов меню и кнопок, после нажатия на которые открывается диалоговое окно. В данном случае раскрывается окно специализированного редактора соответствующего свойства, если он существует.

Если требуется просматривать и редактировать большое количество свойств компонента, бывает удобнее щёлкнуть правой кнопкой мыши по нужному компоненту и в появившемся всплывающем меню выбрать пункт “Properties”. В этом случае откроется отдельное окно редактирования свойств компонента. Можно держать открытыми одновременно произвольное количество таких окон.

Булевские свойства в колонке значений свойств показываются в виде кнопок выбора checkbox – квадратиков с возможностью установки галочки внутри. Если галочки нет, значение свойства false, если есть – true.

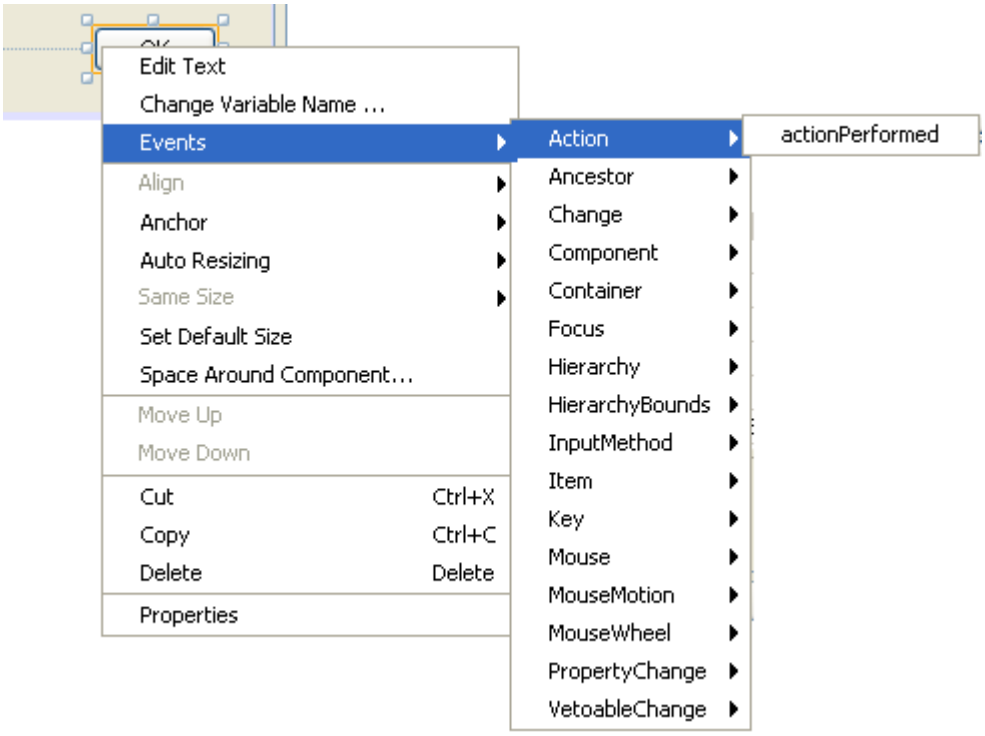
Перечислим на примере кнопки ряд некоторых важнейших свойств, которые можно устанавливать для компонентов. Многие из них относятся и к другим компонентам.

Название свойства	Что оно задаёт
background	Цвет фона
componentPopupMenu	Позволяет назначать всплывающее меню, появляющееся по нажатию правой кнопкой мыши в области компонента.

font	Фонт, которым делается надпись на компоненте.
foreground	Цвет фонта, которым делается надпись на компоненте.
icon	Картинка, которая рисуется на компоненте рядом с текстом.
text	Текст (надпись) на компоненте.
toolTipText	Всплывающая подсказка, появляющаяся через некоторое время при наведении курсора мыши на компонент.
border	Тип рамки вокруг компонента.
borderPainted	Рисуется ли рамка вокруг компонента.
contentAreaFilled	Имеется ли заполнение цветом внутренней области компонента (для кнопок оно создаёт эффект трёхмерности, без заполнения кнопка выглядит плоской).
defaultCapable	Способна ли кнопка быть “кнопкой по умолчанию”: при нажатии <Enter> автоматически происходит нажатие “кнопки по умолчанию” (такая кнопка на экранной форме должна быть одна).
enabled	Доступен ли компонент. По умолчанию все создаваемые на форме компоненты доступны. Недоступные компоненты рисуются более блеклыми красками.

В качестве примера добавим всплывающую подсказку для нашей кнопки: введём текст “Эта кнопка предназначена для выхода из программы” в поле, соответствующее свойству toolTipText. К сожалению, подсказка может быть только однострочной – символы перевода на новую строку при выводе подсказки игнорируются, даже если они заданы в строке программным путём.

Наконец, зададим действие, которое будет выполняться при нажатии на кнопку – *обработчик события* (event handler) нажатия на кнопку. Для этого сначала выделим кнопку, после чего щёлкнем по ней правой кнопкой мыши, и в появившемся всплывающем меню выберем пункт Events/Action/actionPerformed.



Назначение обработчика события

Events означает “События”, Action – “Действие”, actionPerformed – “выполненное действие”.

После этого произойдёт автоматический переход в редактор исходного кода, и там появится заготовка обработчика события:

```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
    }
```

Аналогичный результат можно получить и более быстрым способом – после того, как мы выделим кнопку в окне редактирования формы (Design), в окне Navigator показывается и выделяется имя этой кнопки. Двойной щелчок по этому имени в окне навигатора приводит к созданию заготовки обработчика события.

Рядом с обработчиком `jButton1ActionPerformed` будет расположен уже имеющийся обработчик события, срабатывающий при нажатии на пункт меню “Выход”:

```
  private void exitMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
      System.exit(0);
  }
```

Заменим в нашем обработчике события строку с комментарием на код, вызывающий выход из программы:

```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        System.exit(0);
    }
```

Теперь после запуска нашего приложения подведение курсора мыши к кнопке приведёт к появлению всплывающей подсказки, а нажатие на кнопку – к выходу из программы.

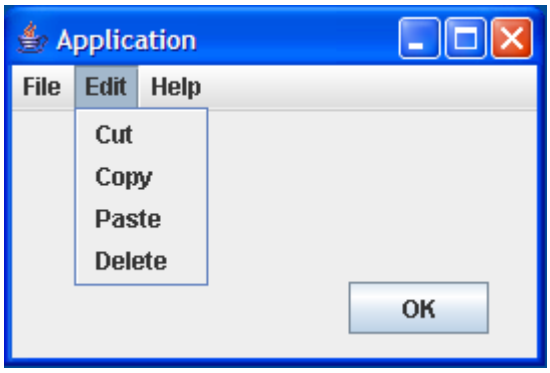
Часто встречающийся случай – показ сообщения при наступлении какого-либо события, например – нажатия на кнопку. Этом случае вызывают панель с сообщением:

javax.swing.JOptionPane.showMessageDialog(null,"Меня нажали");

Если классы пакета javax.swing импортированы, префикс javax.swing при вызове не нужен.

2.10. Внешний вид приложения

На этапе редактирования приложения внешний вид его компонентов соответствует платформе. Однако после запуска он становится совсем другим, поскольку по умолчанию все приложения Java показываются в платформо-независимом виде.:

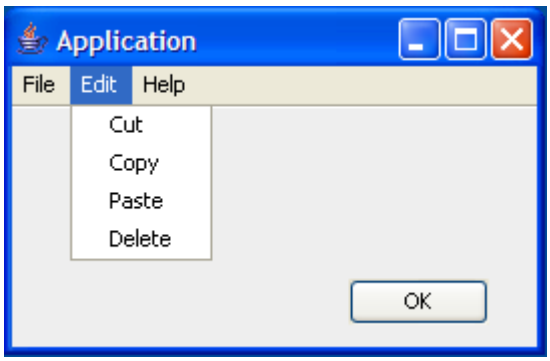


Внешний вид запущенного приложения с платформо-независимым пользовательским интерфейсом, задаваемым по умолчанию

Кроме того, наше приложение появляется в левом верхнем углу экрана, а хотелось бы, чтобы оно появлялось в центре.

Для того, чтобы показать приложение в платформо-ориентированном виде (то есть в том виде, который использует компоненты и настройки операционной системы), требуется изменить код конструктора приложения, вставив перед вызовом метода initComponents задание типа пользовательского интерфейса (User's Interface, сокращённо UI):

```
import javax.swing.*;
import java.awt.*;

...

public GUI_application() {

  try{
      UIManager.setLookAndFeel( UIManager.getSystemLookAndFeelClassName() );
     }catch(Exception e){};

  initComponents();

  Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
  Dimension frameSize = getSize();
  setLocation(new Point( (screenSize.width-frameSize.width)/2,
                         (screenSize.height-frameSize.width)/2 )
              );
}
```



Внешний вид запущенного приложения с платформо-ориентированным пользовательским интерфейсом в операционной системе Windows® XP

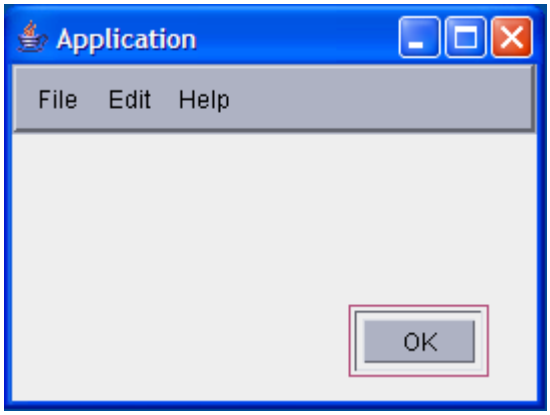
Код, следующий после вызова `initComponents()`, предназначен для установки окна приложения в центр экрана.

Имеется возможность задания ещё одного платформо-независимого вида приложения – в стиле Motiff, используемого в операционной системе Solaris®. Для установки такого вида вместо вызова

```
UIManager.setLookAndFeel( UIManager.getSystemLookAndFeelClassName()
```

Следует написать

```
UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
```



Внешний вид запущенного приложения с платформо-независимым пользовательским интерфейсом в стиле Motiff

Использованные конструкции станут понятны читателю после изучения дальнейших разделов методического пособия.

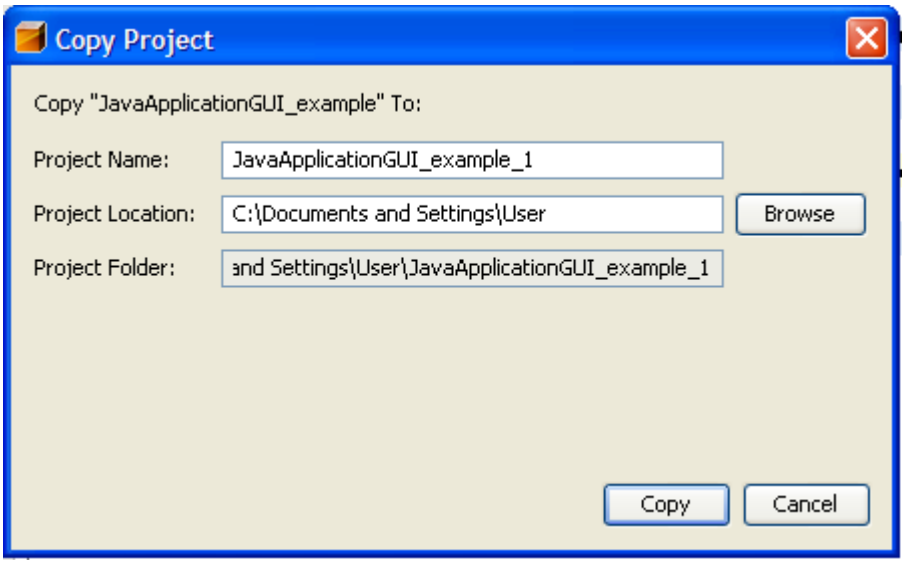
2.11. Ведение проектов

Для того, чтобы не запутаться в разных проектах и их версиях, особенно с учётом того, что учебные проекты бывает необходимо часто переносить с одного компьютера на другой, следует серьёзно отнестись к ведению проектов. Автором в результате многолетней практики работы с разными языками и средами программирования выработана следующая система (откорректированная в применении к среде NetBeans):

- Под каждый проект создаётся папка с названием проекта. Будем называть её папкой архива для данного проекта. Названия используемых папок могут быть русскоязычными, как и имена приложений и файлов.
- При создании нового проекта среда разработки предлагает ввести имя папки, где его

хранить - следует указать имя папки архива. Кроме того, предлагается ввести имя проекта. Это имя будет использовано средой NetBeans для создания папки проекта, так и для названия вашего приложения. Для того, чтобы облегчить работу с вашим приложением в разных странах, рекомендуется делать это название англоязычным. В папке проекта среда разработки автоматически создаст систему вложенных папок проекта и все его файлы. Структура папок проектов NetBeans была описана ранее.

- Если берётся проект с существующим исходным кодом, его папка копируется в папку нашего архива либо вручную, либо выбором соответствующей последовательности действий в мастере создания проектов NetBeans.
- При получении сколько-нибудь работоспособной версии проекта следует делать его архивную копию. Для этого в открытом проекте в окне “Projects” достаточно щелкнуть правой кнопкой мыши по имени проекта, и в появившемся всплывающем меню выбрать пункт “Copy Project”. Откроется диалоговая форма, в которой предлагается автоматически образованное имя копии – к первоначальному имени проекта добавляется подчёркивание и номер копии. Для первой копии это _1, для второй _2, и так далее. Причём головная папка архива по умолчанию остаётся той же, что и у первоначального проекта. Что очень удобно, поскольку даёт возможность создавать копию всего тремя щелчками мышки без набора чего-либо с клавиатуры.



Создание рабочей копии проекта

Скопированный проект автоматически возникает в окне “Projects”, но не становится главным. То есть вы продолжаете работать с прежним проектом, и все его открытые окна сохраняются. Можно сразу закрыть новый проект – правой кнопкой мыши щёлкнуть по его имени, и в появившемся всплывающем меню выбрать пункт “Close Project”.

Для чего нужна такая система ведения проектов? Дело в том, что у начинающих программистов имеется обыкновение разрушать результаты собственного труда. Они развивают проект, не сохраняя архивов. Доводят его до почти работающего состояния, после чего ещё немного усовершенствуют, затем ещё – и всё перестаёт работать. А так как они вконец запутываются, восстановить работающую версию уже нет возможности. И им нечего предъявить преподавателю или начальнику!

Поэтому следует приучиться копировать в архив все промежуточные версии проекта, более работоспособные, чем уже сохранённые в архив. В реальных проектах трудно запомнить все изменения, сделанные в конкретной версии, и, что важнее, все взаимосвязи,

вызвавшие эти изменения. Поэтому даже опытным программистам время от времени приходится констатировать: “Ничего не получается!” И восстанавливать версию, в которой ещё не было тех нововведений, которые привели к путанице. Кроме того, часто бывает, что новая версия в каких-то ситуациях работает неправильно. И приходится возвращаться на десятки версий назад в поисках той, где не было таких “глюков”. А затем внимательно сравнивать работу двух версий, выясняя причину неправильной работы более поздней версии. Или убеждаться, что все предыдущие версии также работали неправильно, просто ошибку не замечали.

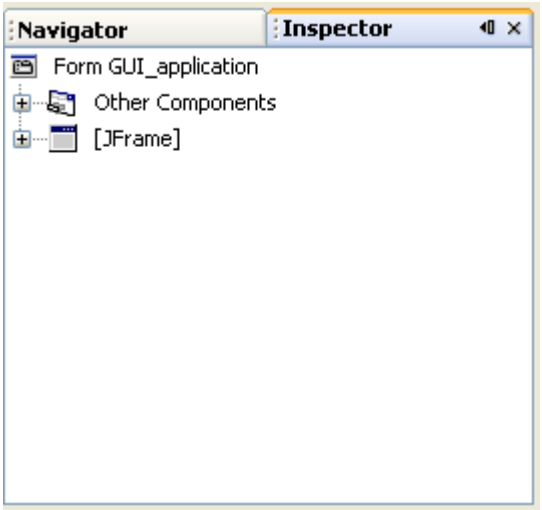
2.11. Редактирование меню экранной формы

Имеются случаи, когда процесс редактирования компонентов несколько отличается от описанного выше. Например, для главного меню экранной формы и для всплывающих меню.

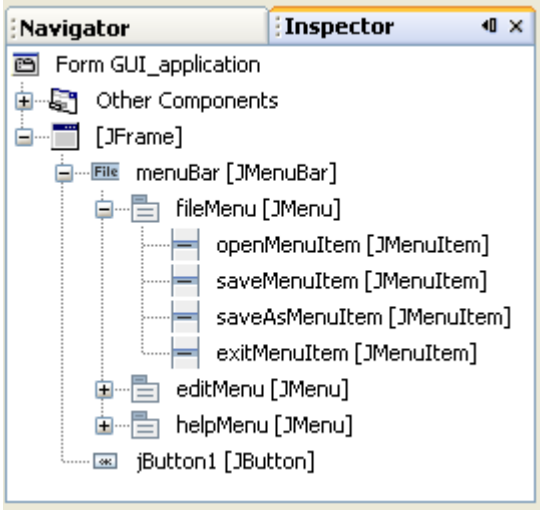
Рассмотрим, как изменить текст пунктов меню формы с английского на русский. Если щёлкнуть мышью по какому-либо пункту (item) меню, в окне редактора свойств появятся значения свойств этого пункта. И мы легко сменим “File” на “Файл”, “Edit” на “Правка”, “Help” на “Справка”. Для того, чтобы без компиляции и запуска программы посмотреть, как будет выглядеть наша экранная форма, можно нажать иконку Preview Design (третья по счёту после закладки Design в окне редактирования экранной формы).

Но вложенные пункты меню, появляющиеся при выборе любого из пунктов верхнего уровня, так отредактировать невозможно. Они редактируются немного другим путём. При переходе в режим дизайна, а также в этом режиме при щелчке в области экранной формы, в левом нижнем окне (Inspector - “инспектор компонентов”) среды разработки появляется список компонентов экранной формы. Навигатор позволяет просматривать деревья вложенности различных элементов проекта.

Сама экранная форма является экземпляром класса JFrame (от Java Frame – “окно, кадр”, предоставляемое языком Java). В окне инспектора после схематического изображения компонента и имени соответствующей ему переменной в квадратных скобках указывается тип компонента. Развернём узел для нашей формы типа JFrame, а также вложенные узлы menuBar типа JMenuBar и fileMemu типа JMenu.



Окно инспектора компонентов

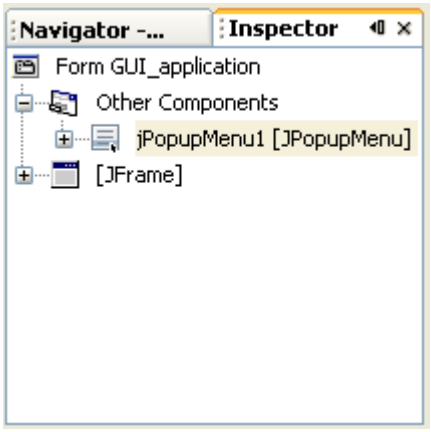


Развёрнутое дерево вложенности

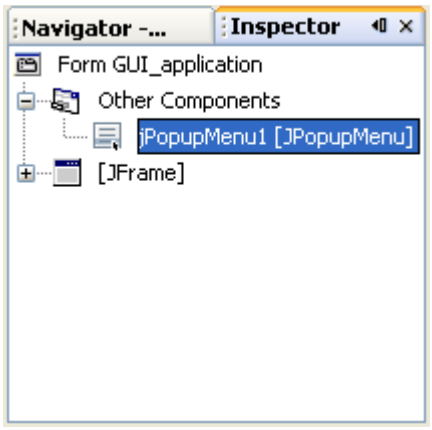
Мы увидим имена переменных, соответствующих всем пунктам меню, вложенным в файловое меню: openMenuItem, saveMenuItem, saveAsMenuItem, exitMenuItem. Щелчок по имени openMenuItem в окне инспектора компонентов приведёт к тому, что в окне редактирования свойств появятся значения свойств данного пункта меню. В поле Text

заменим слово “Open” на “Открыть”. Затем перейдём на пункт saveMenuItem, и так далее. В результате получим экранную форму с пунктами меню на русском языке.

Рассмотрим теперь создание всплывающего меню, появляющегося при щелчке по какому-либо компоненту нашей формы. Качестве примера назначим всплывающее меню кнопке выхода. Для других компонентов процесс будет абсолютно аналогичным.



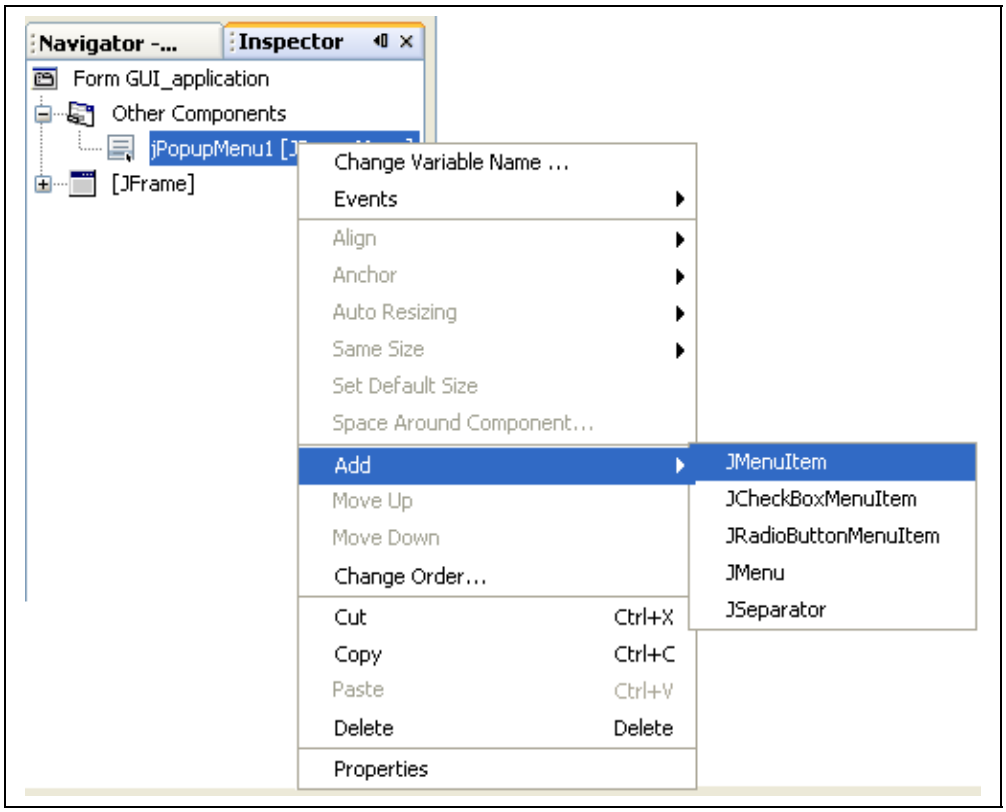
Новый узел jPopupMenu1



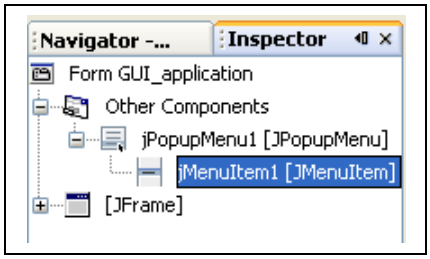
Содержание узла

В режиме дизайна (закладка Design) выберем мышью в палитре компонентов (окно Palette в правом верхнем окне) компонент JPopupMenu, и перетащим его на экранную форму. Он там не появится, но в окне инспектора компонентов в дереве Other Components возникнет новый узел jPopupMenu1[JPopupMenu]. Если щёлкнуть по узлу, окажется, что кроме самого компонента jPopupMenu1 в нём ничего нет.

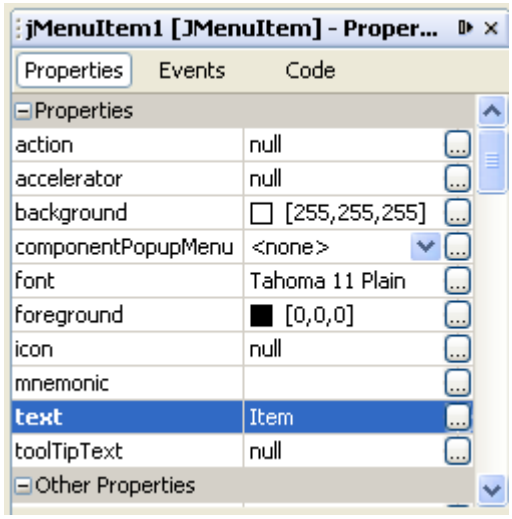
Щёлкнем правой кнопкой мыши по этому узлу, и в появившемся всплывающем меню выберем Add/JMenuItem.



После этого в дереве jPopupMenu1 возникнет узел jMenuItem1[JMenuItem], и в редакторе свойств компонентов можно задать значение свойству Text данного компонента. Введём текст “Выйти из программы”.



Узел jMenuItem1



Свойства jMenuItem1

Далее уже известным нам способом зададим обработчик нажатия на этот пункт меню – выберем во всплывающем меню, возникающем при щелчке правой кнопкой мыши по имени jMenuItem1 в окне Inspector, пункт Events/ Action/ ActionPerformed. А в обработчике напишем оператор выхода из программы

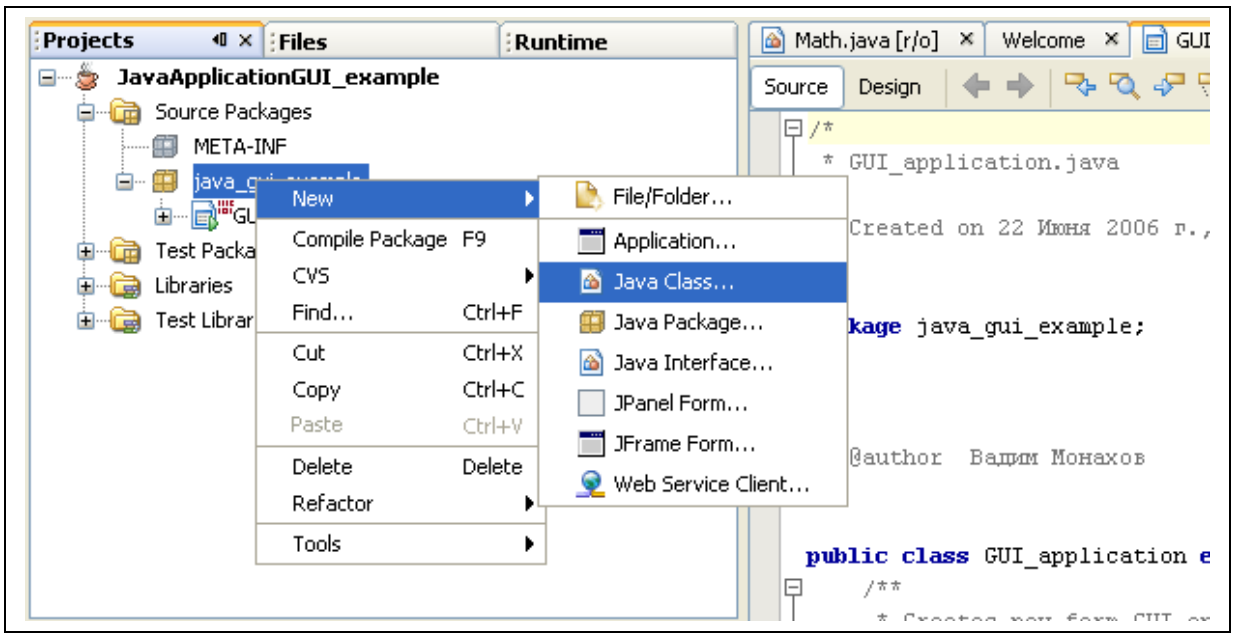
```
System.exit(0);
```

Мы пока только создали всплывающее меню, которое доступно в нашей форме, но ещё не назначили его никакому компоненту. Для того, чтобы назначить меню jPopupMenu1 кнопке JButton1, выделим её, и в редакторе свойств компонентов в пункте componentPopupMenu нажмём мышью стрелку вниз, разворачивающую выпадающий список. Кроме значения <none>, назначенного по умолчанию этому свойству для каждого компонента, там имеется имя jPopupMenu1. Его мы и выберем.

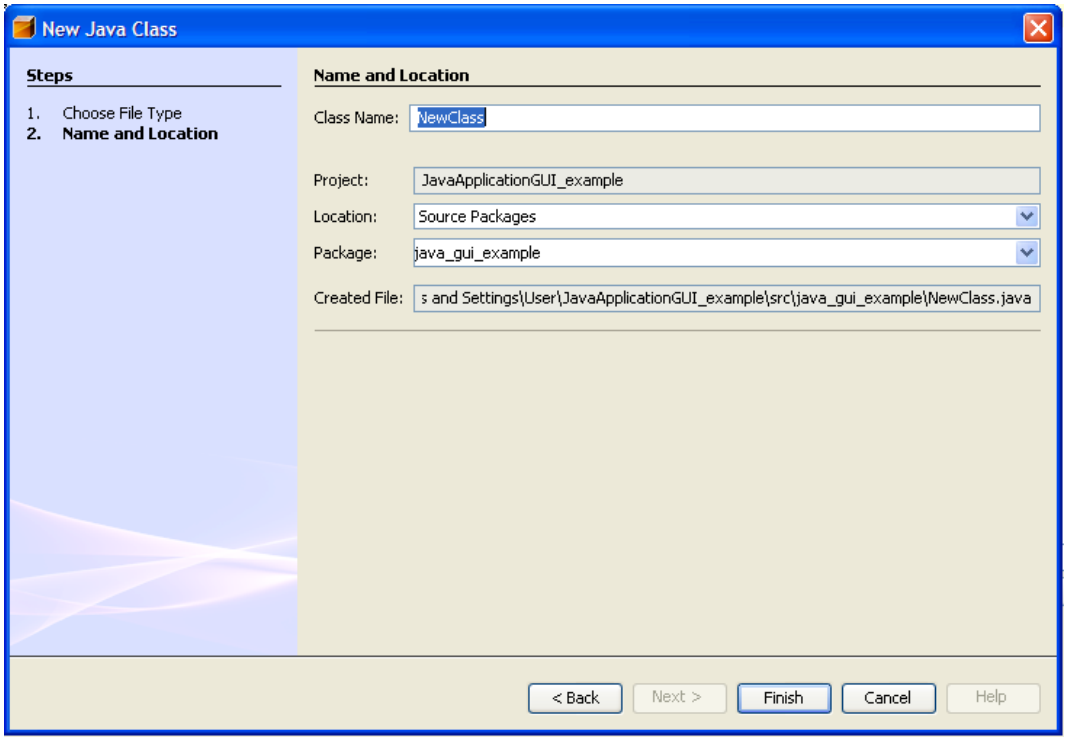
Теперь всплывающее меню, состоящее из одного пункта “ Выйти из программы ”, появится при щелчке правой кнопкой мыши по кнопке. Добавление других пунктов меню и назначение им обработчиков событий проводится абсолютно так же, как для jMenuItem1.

2.12. Создание нового класса

Пусть мы хотим создать в нашем проекте новый класс. Для этого щёлкнем правой кнопкой мыши по имени нашего пакета, и выберем в появившемся всплывающем окне пункт New/ Java Class…



Создание нового класса. Шаг 1.



Создание нового класса. Шаг 2.

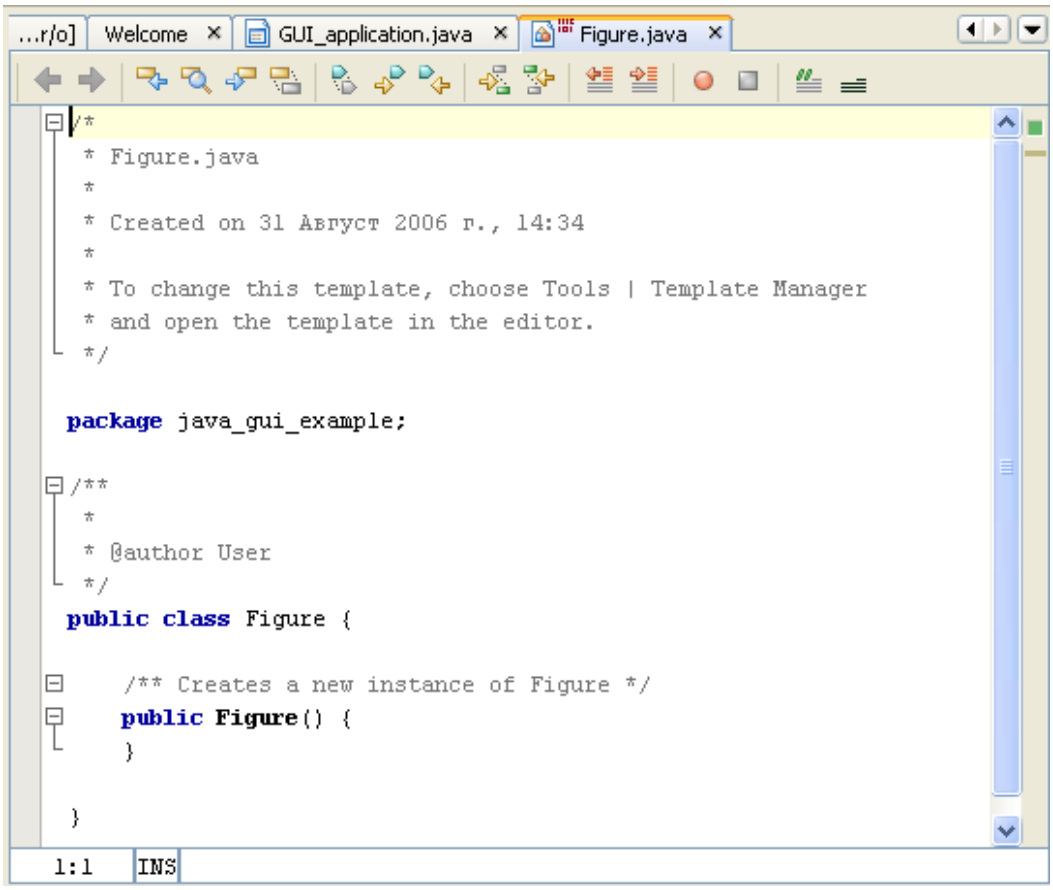
Появится диалоговое окно создания нового класса Java. В нём следует задать имя создаваемого класса, заменив имя по умолчанию. Зададим имя Figure.

В качестве пакета, в котором расположен класс, будет автоматически задан пакет нашего приложения (если первоначальный щелчок правой клавишей был по имени этого пакета).

Кроме описанной выше процедуры для создания нового класса можно воспользоваться мастером создания нового класса в главном меню среды NetBeans (File/New File…/Java Classes/Next>). В результате появится то же диалоговое окно, но в выпадающем списке придётся выбрать имя пакета .

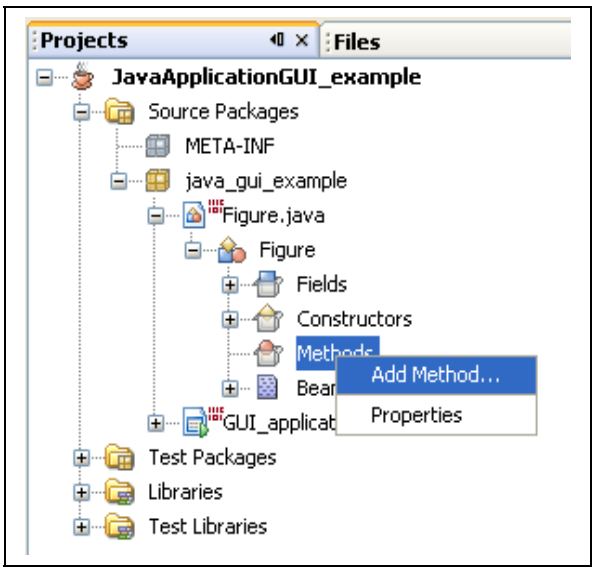
После нажатия на кнопку Finish (“закончить”) в редакторе исходного кода появляется

заготовка класса.



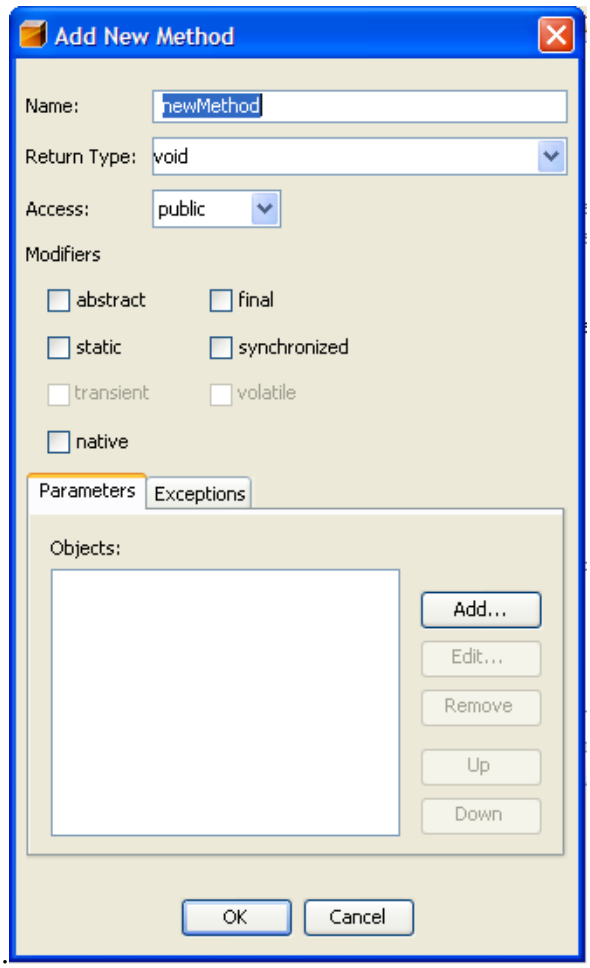
Заготовка нового класса.

Если в класс требуется добавить метод, поле данных или конструктор, можно это делать вручную. Но удобнее добавлять методы с помощью среды разработки.



Добавление в класс метода.Шаг 1.

Щелчок правой клавиши мышки в области надписи Methods и выбор пункта Add Method… всплывающего окна приводит к появлению диалога, в котором можно путём установки галочек и выбора пунктов выпадающего меню задавать нужные параметры метода



Добавление в класс метода. Шаг 2.

Аналогичным образом добавляются новые поля (Fields) и конструкторы (Constructors), но щелчок правой клавишей мыши должен делаться в области надписей Fields или Constructors.

2.13. Документирование исходного кода в Java

Одной из важнейших частей написания программного обеспечения является документирование создаваемого кода. В Java для этих целей применяется средство, обеспечивающее поддержку на уровне синтаксиса языка программирования – специализированные комментарии. Они начинаются с комбинации символов /** и заканчиваются комбинацией символов */

Часть комментариев автоматически создаёт среда разработки.

Пример:

```
/**
 * Creates new form GUI_application
 */
```

Средством обработки внедрённых в исходный код комментариев и создания для класса справочных HTML-файлов является инструмент javadoc, входящий в состав JDK. Но в среде NetBeans удобнее пользоваться вызовом через главное меню: Build/Generate Javadoc for "…".

Документационные комментарии бывают для:

- Пакетов (пока не функционируют).
- Классов.

- Интерфейсов.
- Пользовательских типов-перечислений (на уровне пакетов пока не функционируют, но можно использовать для типов, заданных в классах).
- Методов.
- Переменных.

Документационные комментарии пишутся непосредственно перед заданием соответствующей конструкции – пакета, класса, интерфейса, типа-перечисления, метода или переменной. Следует учитывать, что по умолчанию документация создаётся только для элементов, имеющих уровень видимости public или protected.

Пример фрагмента кода с документационными комментариями:

```
/**
 * Пример приложения Java с документационными комментариями <br>
 * В приложении заданы типы-перечисления Monthes и Spring и показано,
 * как с ними работать.
 * Кроме того, дан пример использования класса из другого пакета.
 * @see enumApplication.Monthes Информация о типе-перечислении Monthes
 * @see enumApplication.Spring
 * @see enumApplication#m1
 * @version Версия 0.1 Первоначальная версия, проверено при компиляции
 * в среде NetBeans 5.5
 * @author Вадим Монахов
 */
public class enumApplication extends javax.swing.JFrame {
    int i=1;
/**
  * Spring -  задаёт перечисление из 3 весенних месяцев года: march,apr,may.
  *  <ul>
  *   <li>march
  *   <li>apr
  *   <li>may
  *  </ul>
  *   Идентификатор для марта записан отличающимся от соответствующего
  * идентификатора в перечислении Monthes, а для апреля и мая записаны так
  * же - чтобы подчеркнуть, что их пространства имён независимы.
  */
    public enum Spring {march,apr,may};

    /**
     * Monthes - задаёт перечисление из 12 месяцев года: <br>
     * jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec <br>
     * (январь, февраль и т.д.)
     */
    public enum Monthes {jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec};
    Spring spr1= Spring.apr, spr2;
    /**
     *Переменная, иллюстрирующая работу с перечислениями
     */
    public Monthes m1,m2=Monthes.mar,m3;
```

Имеется два типа кода внутри блока документационного комментария – HTML-текст и метаданные (команды документации, начинающиеся с символа @). Если пишется обычный текст, он рассматривается как HTML-текст, поэтому все пробелы и переносы на новую строку при показе приводятся к одному пробелу. Для того, чтобы очередное предложение при показе начиналось с новой строки, следует вставить последовательность символов
 , называющуюся тегом HTML. Возможно использование произвольных тегов HTML, а не только тега переноса на новую строку: теги неупорядоченного списка <ul> и

, теги гиперссылок, изображений и т.д. В то же время не рекомендуется использовать заголовки и фреймы, поскольку это может привести к проблемам – javadoc создаёт на основе документационного кода собственную систему заголовков и фреймов. Кроме того, при преобразовании в HTML-документ из документационного кода удаляются символы “*”, если они стоят на первом значимом месте в строке (символы пробелов не являются значимыми).

Для более подробного изучения тегов HTML следует читать справочную или учебную литературу по этому языку разметки документов. Соответствующие ссылки и документы можно найти, например, на сайте автора

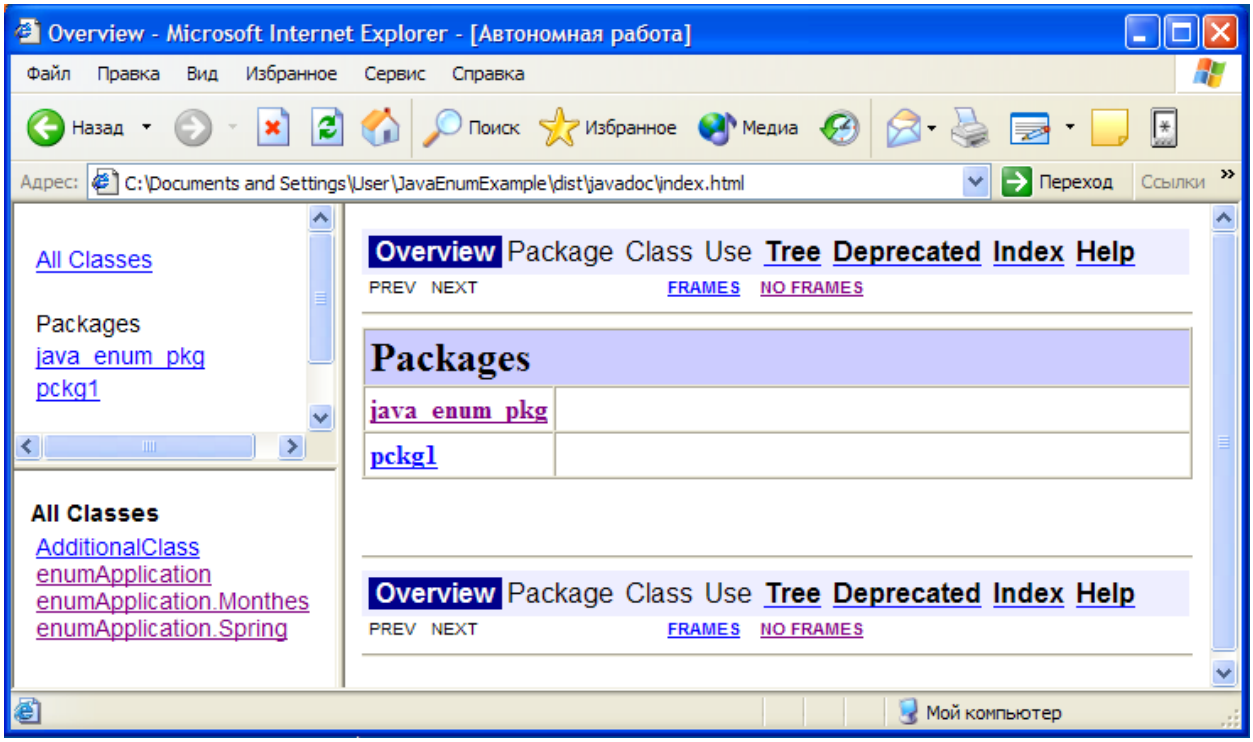
http://barsic.spbu.ru/ www/comlan/html_r.html

Команды документации (символы метаданных):

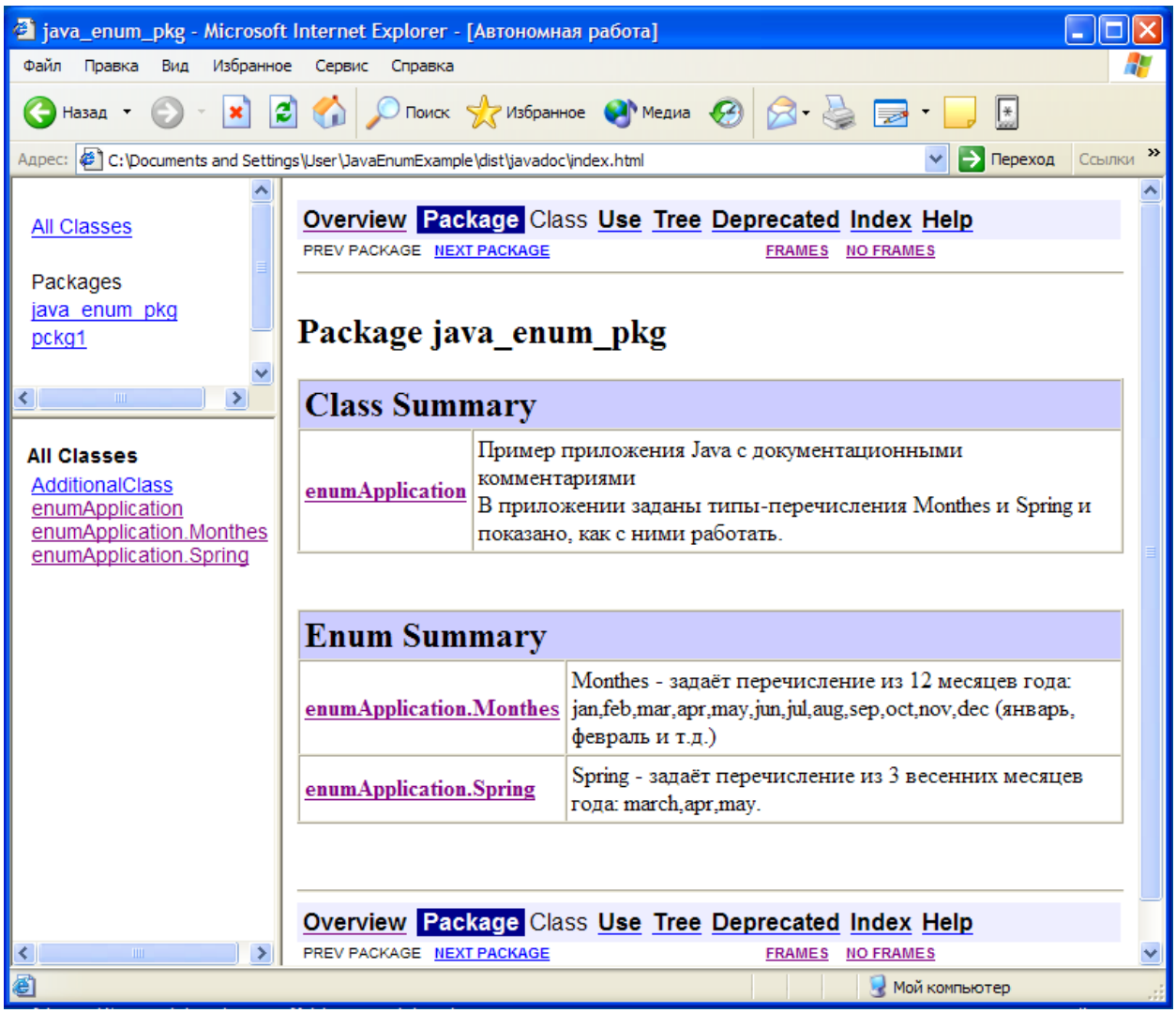
- @see (“смотри”) – применяется для создания в документе гиперссылок на другие комментарии. Можно использовать для любых конструкций (классов, методов и т.д.). Формат использования: @see ИмяКласса – для класса; @see ИмяКласса.ИмяПеречисления – для типа-перечисления, заданного в классе; @see ИмяКласса#ИмяЧлена – для метода или переменной; для интерфейса – аналогично классу. При этом имя класса или интерфейса может быть либо коротким, либо квалифицировано именем пакета.
- @version (“версия”) – информация о версии. Используется для классов и интерфейсов. Формат использования: @version Информация о версии в произвольной форме.
- @author (“автор”) - Информация об авторе. Используется для классов и интерфейсов. Формат использования: @author Информация об авторе в произвольной форме. Может включать не только имя, но и данные об авторских правах, а также об электронной почте автора, его сайте и т.д.
- @since (“начиная с”) - Информация о версии JDK, начиная с которой введён или работоспособен класс или интерфейс. Формат использования: @since Информация в произвольной форме.
- @param (сокращение от parameter -“параметр”) - информация о параметре метода. Комментарий /** @param … */ ставится в месте декларации метода в списке параметров перед соответствующим параметром. Формат использования: @param ИмяПараметра Описание.
- @return (“возвращает”) - информация о возвращаемом методом значении и его типе. Формат использования: @return Информация в произвольной форме.
- @throws (“возбуждает исключение”) - информация об исключительных ситуациях, которые могут возбуждаться методом. Формат использования: @throws ИмяКлассаИсключения Описание.
- @deprecated (“устаревшее”) - информация о том, что данный метод устарел и в последующих версиях будет ликвидирован. При попытке использования таких методов компилятор выдаёт программисту предупреждение (warning) о том, что метод устарел, хотя и компилирует проект. Формат использования: @deprecated Информация в произвольной форме.

Признаком окончания команды документации является начало новой команды или окончание комментария.

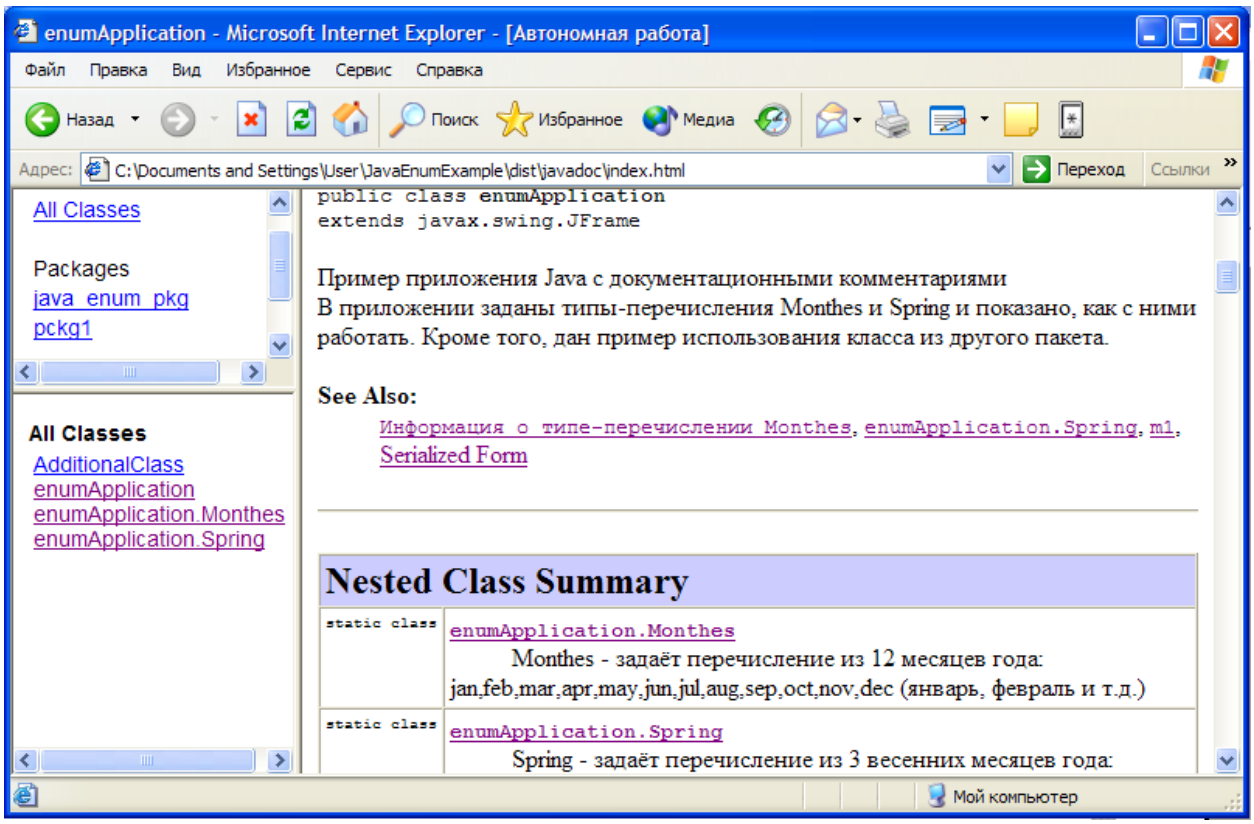
Пример документации, созданной для пакета, из которого взят приведённый выше фрагмент кода:



Головная страница файлов документации



Страница описания элементов пакета java_enum_pkg



Страница описания класса enumApplication

Обратите внимание, что в краткой сводке (summary) приводятся только начальные строки соответствующей информации по элементам пакета или класса. Полную информацию можно прочитать после перехода по гиперссылке в описании соответствующего элемента. Поэтому важно, чтобы первые 2-3 строки информации содержали самые важные сведения.

2.14. Основные компоненты пакетов swing и awt

Пока мы научились работать только с формами, кнопками и всплывающими меню. Перечислим ещё ряд полезных компонентов.

Во-первых, следует остановиться на том, что в палитре компонентов NetBeans предлагается три категории компонентов: из библиотеки Swing (пакет swing), библиотеки AWT (пакет awt), и категория Beans. В Sun Java Studio Enterprise имеется ещё одна категория – Layouts, “менеджеры размещения”, - компоненты, отвечающие за способ расположения и выравнивания компонентов на форме.

Библиотека Swing является основной для большинства современных графических приложений Java. В ней предлагаются следующие компоненты (перечислены в том порядке, в каком они возникают в палитре компонентов):

№	Компонент	Назначение компонента
1	JLabel	“Метка” – вывод однострочного неформатированного текста
2	JButton	“Кнопка” – кнопка с текстом и/или с картинкой
3	JToggleButton	“Защёлкивающаяся кнопка” – кнопка с фиксацией. Может быть одной из нескольких таких кнопок в группе, в этом случае нажатие одной кнопки вызывает отпускание другой. Работа группы обеспечивается компонентом ButtonGroup, который должен быть перетащен на форму, а затем назначен свойству buttonGroup.

4	JCheckBox	“Чекбокс” - пункт выбора с независимой фиксацией.
5	JRadioButton	“Радиокнопка” - пункт выбора с зависимой фиксацией, должен быть одним из нескольких в группе. Работа группы обеспечивается компонентом ButtonGroup.
6	ButtonGroup	Обеспечивает работу групп компонентов JToggleButton или JradioButton.
7	JComboBox	“Комбобокс” – выпадающий список.
8	JList	Прокручивающийся список.
9	JTextField	“Текстовое поле” – однострочный пункт ввода и редактирования текста.
10	JTextArea	“Текстовая область” – многострочный пункт ввода и редактирования текста.
11	JPanel	“Панель” – группирующий компонент, позволяющий располагать на себе другие компоненты. Передвижение панели перемещает вместе с ней все расположенные на ней компоненты. По умолчанию свойство layout (“размещение”) установлено как FlowLayout – “в виде потока”. Для простых задач вместо этого компонента лучше использовать JLayeredPane.
12	JTabbedPane	“Панель с закладками” – каждый положенный на неё компонент показывается в отдельной закладке. Чтобы разместить на одной закладке несколько компонентов, сначала положите на панель с закладками обычную панель. Для того, чтобы создать последующие закладки, выделите панель с закладками, вызовите правой кнопкой мыши всплывающее меню, пункт Add From Palette (“добавить из палитры”), и добавьте ещё одну панель или другой компонент.
13	JScrollBar	Независимая полоса прокрутки. Используется редко – для программно управляемой прокрутки содержимого компонентов, для которых отсутствуют встроенные полосы прокрутки.
14	JScrollPane	“Панель с полосами прокрутки”
15	JMenuBar	“Меню формы” – предназначено для расположения в нём компонентов типа JMenu (заголовков меню).
16	JPopupMenu	“Всплывающее меню” - предназначено для расположения в нём компонентов типа JMenuItem (пунктов меню).
17	JSlider	"Ползунок". Используется для плавной регулировки числовых величин, а также связанных с ними программно регулируемых изменений.
18	JProgressBar	“Прогрессбар” – полоса показа доли выполнения задачи. Показывает уровень, отражающий долю выполнения задачи
19	JSplitPane	“Панель с разделителем” – панель, состоящая из двух частей, между которыми имеется линия разделителя, которую можно перетаскивать мышью, меняя взаимный размер частей.
20	JFormattedTextField	“Поле ввода форматированного текста”
21	JPasswordField	“Поле ввода пароля” – вводимый текст отображается звёздочками.
22	JSpinner	“Спиннер” - поле ввода числа с кнопками увеличения/ уменьшения.
23	JSeparator	“Сепаратор” – разделительная линия. Используется в

		декоративных целях для разделения рабочих областей формы и других группирующих компонентов.
24	JTextPane	“Текстовая панель”. По умолчанию автоматически переносит текст на новую строку. А не располагает в одну строку с показом горизонтального скроллера, как это делает JTextArea.
25	JEditorPane	“Панель текстового редактора”
26	JTree	“Дерево” – показывает дерево, в котором каждая ветвь может быть с иконками и текстом, а узлы разворачиваются и сворачиваются.
27	JTable	“Таблица” – показ текстовой таблицы. Имеет возможность заполнения значениями по умолчанию на этапе проектирования.
28	JToolBar	“Тулбар” – панель инструментов. Обычно на нём размещают кнопки JToggleButton, для которых назначены иконки.
29	JInternalFrame	“Дочернее окно” – окно многооконного приложения. Его можно перемещать в пределах родительского окна – главного окна приложения. В настоящее время такой стиль приложений практически не используется.
30	JLayeredPane	“Панель с абсолютным позиционированием элементов”
31	JDesktopPane	“Панель – рабочий стол”. Ещё один тип панели с абсолютным позиционированием элементов.

32	JOptionPane	“Диалоговая панель” – предназначена для показа диалоговых форм. В отличие от большинства других компонентов работа идёт с помощью методов класса. Имеются вызовы диалогов:
		• С сообщением:
		`javax.swing.JOptionPane.showMessageDialog(null, "Кнопку нажали");`
		`JOptionPane.showMessageDialog(null,"Привет!", "Заголовок сообщения", JOptionPane.INFORMATION_MESSAGE);`
		• С подтверждением:
		`int option=javax.swing.JOptionPane.showConfirmDialog( null,"Продолжить?");`
		Проверка, какую кнопку нажали или диалог закрыли без осуществления выбора, осуществляется сравнением с константами javax.swing.JOptionPane.NO_OPTION, CANCEL_OPTION, CLOSED_OPTION, OK_OPTION, YES_OPTION
		• С предложением ввести значение:
		`String input=javax.swing.JOptionPane.showInputDialog( null,"Введите значение:");`
		- при отказе от ввода возвращается null.
		Первый параметр – имя формы, в которой показывается диалог. Если он null – используется форма по умолчанию (главная форма приложения).
		Существуют варианты указанных выше методов, позволяющие при вызове задавать дополнительные параметры диалога (заголовок, надписи на кнопках и др.).

33	JColorChooser	“Диалог выбора цвета” – предназначен для выбора пользователем цвета.
34	JFileChooser	“Диалог выбора файла” – предназначен для выбора пользователем файлов. Перед использованием требуется

		положить его на какую-нибудь диалоговую форму (JDialog, JFrame) или какой-нибудь группирующий компонент формы.
35	JFrame	“Экранная форма”. Показывается вызовом вида jFrame1.setVisible(true);
36	JDialog	“Диалоговая форма”. Показывается вызовом вида jDialog1.setVisible(true);

Очень часто в приложении требуется вывести служебную информацию. В старых версиях Java для этого служил вызов System.out.println(“Текст сообщения”). В учебных проектах и при выводе отладочной информации этот метод до сих пор удобен. Но предоставлять таким образом информацию конечному пользователю представляется анахронизмом. Для выдачи пользователю информационного сообщения лучше использовать вызов

```
JOptionPane.showMessageDialog(null,"Привет!","Заголовок сообщения",
                                    JOptionPane.INFORMATION_MESSAGE);
```

Если требуется вывести предупреждение об ошибке, последний параметр должен иметь значение `JOptionPane.ERROR_MESSAGE`, другое предупреждение - `JOptionPane.WARNING_MESSAGE` , вопрос - `JOptionPane.QUESTION_MESSAGE`. Наконец, если не требуется сопровождать вопрос иконкой на диалоговой панели, параметр должен быть `JOptionPane.PLAIN_MESSAGE`.

Библиотека компонентов AWT (Abstract Window Toolkit - Абстрактный Инструментарий графического Окна) является устаревшей по сравнению с библиотекой Swing, хотя сам пакет awt до сих пор является основой графики Java. В библиотеке AWT имеются практически те же компоненты, что и в Swing, но в меньшем количестве и в более примитивном варианте - с худшим дизайном и меньшей функциональностью.

Единственный компонент AWT, у которого нет прямого аналога в Swing – компонент типа Canvas – “холст для рисования”. Он обеспечивал вывод графических примитивов. Например, следующим образом:

```
java.awt.Graphics g=canvas1.getGraphics();
g.drawLine(10,10,100,100);
```

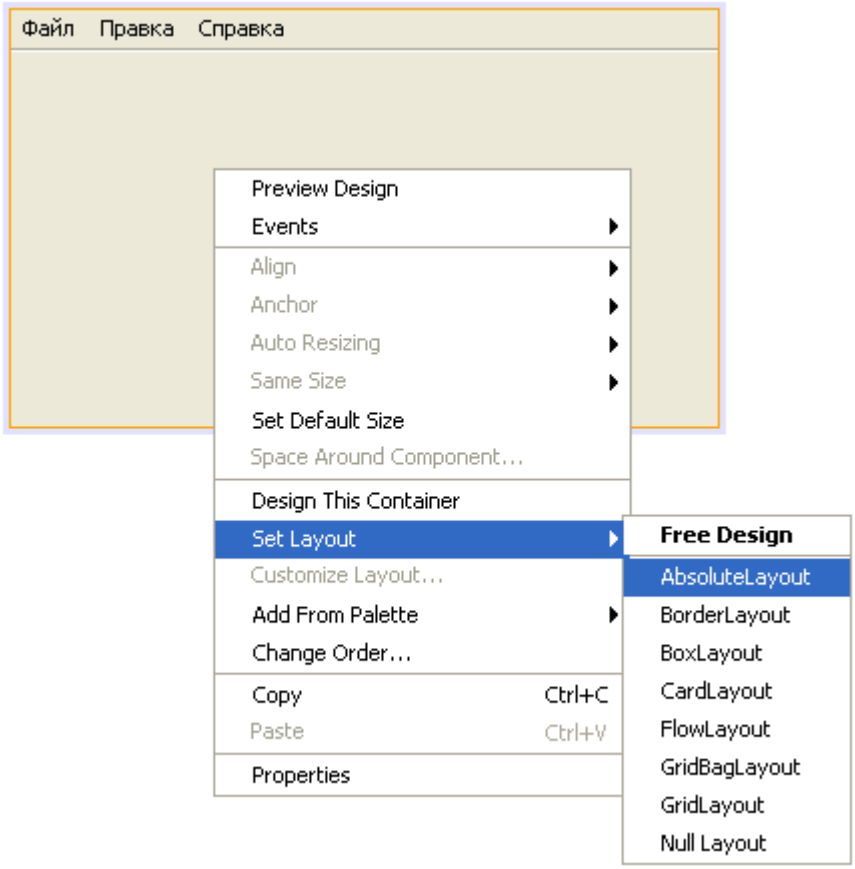
В Swing для этих целей можно рисовать по любому компоненту, например, по панели, или даже по кнопке:

```
java.awt.Graphics g=jPanel1.getGraphics();
g.drawLine(10,10,100,100);
g=jButton3.getGraphics();
g.drawLine(10,10,100,100);
```

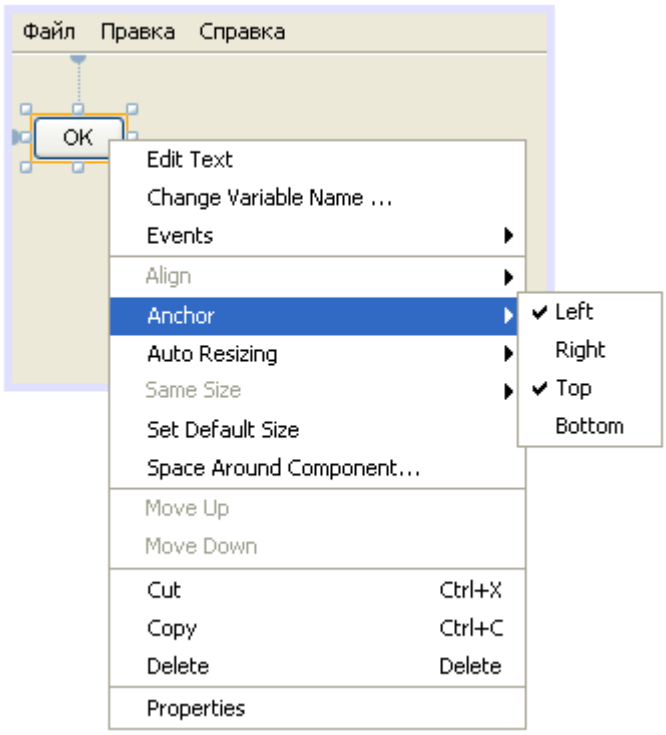
Ещё одна категория, на которой следует остановиться, это компоненты Layout – менеджеры размещения. Разработчики Java предложили оригинальную, но очень неоднозначную идею организации расположения компонентов на форме. Вместо того, чтобы явно указывать позиции компонентов на этапе проектирования или работы программы, и использовать якоря (anchors) для привязки краёв компонентов к краям группирующего компонента, как это делается в других языках программирования, предлагается использовать тот или иной менеджер размещения. При изменении размера формы взаимное расположение компонентов будет меняться в зависимости от типа менеджера. Например, “обычное” размещение с фиксированным положением достигается с помощью размещения на форме менеджера AbsoluteLayout. В NetBeans это делается через пункт Set Layout всплывающего меню, как показано на рисунке. По умолчанию действует режим Free Design - “свободный дизайн”. Если установить менеджер размещения AbsoluteLayout, в редакторе свойств компонентов оказываются доступны свойства x и y – координаты компонентов.

Использовать якоря всё же можно, но с ограниченными возможностями и только в

менеджере размещения Free Design – в других менеджерах они не работают. Для использования якоря следует щёлкнуть с помощью правой клавиши мыши по компоненту, расположенному на форме (например, кнопке), и в появившемся меню выбрать пункт Anchors. Якорь привязывает компонент к соответствующей стороне формы.



Выбор менеджера размещения



Установка привязки к краям формы – якорей

Left – привязка к левому краю формы, Right – к правому, Top- к верхнему, Bottom – к нижнему. По умолчанию менеджер сам выбирает варианты привязки, показывая их пунктирными линиями.

2.15. Технологии Java и .Net

Язык Java был создан в 1995 году как платформо-независимый язык прикладного программирования. Он очень быстро приобрёл широкую популярность, и заметно потеснил языки C и C++ в области разработки прикладного программного обеспечения. В результате стали говорить о технологии Java и о платформе Java, подчёркивая, что это больше, чем просто язык программирования. В 1998 году появилась компонентная модель Java Beans, и ряд сред разработки приложений Java стал успешно конкурировать со средами, обеспечивающими визуальное проектирование пользовательского интерфейса – Microsoft Visual BASIC и Borland Delphi. Казалось, что язык Java завоевал лидирующие позиции в области прикладного программирования.

Но в 2000 году Microsoft была предложена новая технология, названная .Net, в большой степени вобравшая в себя основные черты технологии Java: динамическую объектную модель, повышенную безопасность приложений (в том числе обеспечиваемую использованием ссылок и сборщика мусора), использование виртуальной машины и платформо-независимого байтового кода. Но технология .Net имела ряд новых черт.

Во-первых, вместо одного языка программирования в .Net стало возможно использование произвольного числа языков программирования. От них требовалось только, чтобы они удовлетворяли спецификации, позволяющей скомпилированным классам работать под управлением виртуальной машины, называемой в .Net Common Language Environment или Common Language Runtime – общей исполняющей средой поддержки языков программирования. Базовым языком программирования стал созданный одновременно с .Net язык C# - фактически, явившийся усовершенствованным вариантом языка Java, но несовместимый с ним как по ряду синтаксических конструкций, так и по скомпилированному коду.

Во-вторых, если Java рассматривался всеми в качестве языка программирования, то технология .Net фактически создавалась как платформо-независимая часть операционной системы MS Windows®. Поэтому важной частью .Net стал набор базовых классов .Net Framework, обеспечивающий поддержку прикладного программирования в большинстве практически важных областей.

В .Net основой программирования, как и в Java, служат классы. Исходный код класса, написанный на любом из языков .Net (то есть удовлетворяющий спецификации Common Language Environment), компилируется в платформо-независимый код. Этот код уже не имеет специфики языка программирования, на котором был написан, работает под управлением исполняющей среды .Net и может использоваться любыми другими классами .Net. Причём скомпилированный код класса может использоваться не только для вызовов, но и для наследования и обеспечения полиморфизма. Такие классы называются *компонентами*. Важно то, что для использования каким-либо приложением необходимого класса .Net Framework как при разработке приложения, так и при его запуске на компьютере пользователя нет необходимости загружать класс через Интернет или устанавливать на компьютере каким-либо другим образом. Достаточно того, чтобы был установлен свободно распространяемый пакет компонентов .Net Framework, что делается в версиях MS Windows® начиная с Windows® XP непосредственно во время установки операционной системы. Причём набор компонентов в пакете стандартизован, что обеспечивает гарантированное нахождение нужного компонента и его работоспособность. Именно эти особенности обеспечивают преимущество оболочки операционной системы по сравнению с отдельными программами и пакетами.

То, что технология .Net была сделана открытой и стандартизирована ISO (International Standard Organization – Международная Организация по Стандартам) в 2004 году, безусловно, сделало её привлекательной для многих разработчиков. Появление проекта Mono с реализацией .Net под Linux, а также ряда других проектов по реализации .Net под разными операционными системами позволило этой технологии выйти за пределы семейства операционных систем одной фирмы и сделало позиции Java неконкурентоспособными в данной области. Действительно, язык программирования – это одно, а оболочка операционной системы с набором большого числа компонентов – совсем другое. Тем более, что язык Java стал одним из возможных языков .Net, то есть вошёл в эту технологию как составная часть.

Как это ни удивительно, в рамках технологии Java удалось найти достойный ответ на вызов со стороны .Net . Им стала платформа NetBeans и идеология Open Source (“Открытый исходный код”).

NetBeans – это технология компонентного программирования, созданная на основе модели Java Beans, о которой речь пойдёт позже. Помимо набора компонентов в состав платформы NetBeans входит свободно распространяемая среда разработки NetBeans, позволяющая создавать различные типы программ на языке Java, в том числе – с использованием компонентов NetBeans, а также создавать такие компоненты.

Движение Open Source, набирающее популярность в последние годы, стремится к всеобщей открытости программного кода. При этом следует отличать два варианта открытости. Первый из них (freeware – свободно распространяемое программное обеспечение) подразумевает свободное распространение и использование программ и их исходных кодов, как правило, с единственным ограничением – сохранением открытости и свободы распространения программ и исходных кодов программных продуктов, использующих этот исходный код. Второй требует открытости исходного кода для изучения и, при необходимости, исправления ошибок, но не означает передачи авторских прав на какое-либо другое использование этого исходного кода.

Среда и компоненты NetBeans распространяются на основе соглашения Sun open licence (“Открытая лицензия Sun”). Эта лицензия позволяет свободно использовать среду и

компоненты NetBeans для создания программного обеспечения, как свободно распространяемого, так и коммерческого, но требует, чтобы исходные коды создаваемых программ были открыты для просмотра.

Мультиплатформенность и наличие большого количества свободно распространяемых компонентов NetBeans в сочетании с качественной бесплатной средой разработки и очень широким использованием языка Java даёт возможность надеяться, что NetBeans сможет стать унифицированной оболочкой различных операционных систем. Но для серьёзного соперничества с .Net потребуется наличие стандартизированных пакетов компонентов, одной библиотеки Swing мало. Более того, необходимо, чтобы все эти пакеты входили в поставку JDK. Наличие разрозненных нестандартизированных пакетов не даст преимуществ перед конкурирующей технологией .Net.

Одним из решений этой проблемы стало расширение базового набора пакетов и классов в составе JDK. Даже самый старый пакет java в новых версиях JDK усовершенствуется, не говоря уж о более новом пакете javax. Кроме того, в поставке NetBeans Enterprise Pack имеется большое число дополнительных библиотечных пакетов.

Краткие итоги по главе 2

- ✓ Три базовых принципа объектно-ориентированного программирования: *инкапсуляция*, *наследование*, *полиморфизм*.
- ✓ *Класс* – это описание того, как устроен *объект*. И *поля данных*, и *методы* задаются в классах. Но при выполнении программы *поля данных* хранятся в объектах, а *методы* – в классах. Методы задают *поведение* объекта, а поля данных - *состояние* объекта.
- ✓ Переменные, описываемые в классах, называются *глобальными*. Они задают поля данных объектов. Переменные, описываемые в методах, называются *локальными*. Они являются вспомогательными и существуют только во время вызова метода.
- ✓ Переменные ссылочного типа содержат адреса данных, а не сами данные. Поэтому присваивания для таких переменных меняют адреса, но не данные. Все объектные типы являются ссылочными. Потеря ссылки на объект приводит к сборке мусора.
- ✓ Объект создаётся с помощью вызова *конструктора* - специальной подпрограммы-функции, задаваемой в классе.
- ✓ Методы делятся на *методы объектов* и *методы классов*. Метод объекта можно вызывать только из объекта соответствующего типа. А метод класса может работать и при отсутствии объекта, и вызываться из класса.
- ✓ При декларации класса можно указывать, что он общедоступен, с помощью *модификатора доступа* public. В этом случае возможен доступ к данному классу из других пакетов. В файле .java можно располагать только один общедоступный класс и произвольное число классов с другим уровнем видимости. Если модификатор public отсутствует, то доступ к классу разрешён только из классов, находящихся с ним в одном пакете. Про такие файлы говорят, что у них *пакетный* вариант доступа.
- ✓ Важнейшими пакетами являются java и javax, а также вложенные в них пакеты. Информацию о содержащихся в них элементах можно получить в среде разработки, набрав java. или javax. И прочитав появившуюся подсказку.
- ✓ Все классы и объекты приложения вызываются и управляются из метода main, который имеет сигнатуру `public static void main(String[] args)`. Он является методом класса, и поэтому для его работы нет необходимости в создании объекта, являющегося экземпляром класса.
- ✓ Визуальное проектирование приложения с графическим интерфейсом пользователя (GUI) происходит в режиме Design. Как правило, основой для построения такого интерфейса служат компоненты Swing.
- ✓ Документирование исходного кода в Java осуществляется с помощью специальных документационных комментариев /** Текст комментария в формате HTML */ . Также имеется ряд команд документации, начинающихся с символа @ .Утилита javadoc позволяет по документационным комментариям создавать систему HTML-страниц с документацией о пакетах и классах.
- ✓ Для выдачи пользователю информационного сообщения следует использовать вызов `JOptionPane.showMessageDialog(null,"Привет!","Заголовок сообщения", JOptionPane.INFORMATION_MESSAGE).`

Типичные ошибки:

- Очень часто встречающаяся ошибка: путают классы, объекты и объектные переменные. Но это совершенно разные сущности. Класс – это тип, то есть описание того, как устроена ячейка памяти, в которой будут располагаться поля данных объекта, и какие методы можно вызывать. Объект – это содержимое ячейки памяти

объектного типа. А в переменной объектного типа содержится адрес объекта.

- Ошибочно считают, что методы хранятся в объектах.
- Пытаются обратиться к объекту с помощью ссылочной переменной, которой назначено значение null.
- Непреднамеренная потеря связи с объектом. Обычно возникает при изменении ссылки на объект внутри подпрограммы.
- При работе с радиокнопками JRadioButton или кнопками JToggleButton забывают назначить им группу ButtonGroup.

Задания

- Создать новый проект NetBeans как Java Application. Импортировать классы пакета swing. В методе main вызвать диалоговую панель `JOptionPane` с каким-либо сообщением. Например, “Привет всем!”.
- На основе проекта с имеющимся кодом – заготовки приложения с графическим интерфейсом, создать приложение с кнопкой “OK”, закрывающей приложение, и кнопкой “Нажми меня!”, вызывающей при нажатии диалоговую панель с сообщением “Меня нажали”.
- Переименовать пункты меню приложения , переведя их на русский язык.
- Добавить в класс приложения общедоступное числовое поле x, инициализированное неким значением, и поле y того же типа, но без инициализации. Добавить кнопку, вызывающую диалоговую панель с сообщением о значениях полей x и y.
- Создать документационные комментарии для поля x и методов – обработчиков событий нажатия на кнопки. Вызвать генерацию документации для проекта, просмотреть в ней созданные комментарии.