BlueJ
Инструкция по применению

Версия 2.0.1

Для BlueJ Версии 2.0.x

Майкл Kölling

Mærsk Институт

Университет Южной Дании

Содержание

1 Предисловие

1.1 О BlueJ

Эта инструкция является введением в использование BlueJ - среды разработки программ на языке Java ™, созданной специально для обучения на начальном уровне. Она была разработана и реализована группой BlueJ в Университете Deakin в Мельбурне (Австралия), и в Университете Кент в Кентербери (Великобритания).

Дополнительная информация о BlueJ доступна по адресу http://www.bluej.org.

1.2 Назначение инструкции

Эта инструкция предназначена для лиц, желающих ознакомиться с возможностями среды. Она не объясняет положенные в основу конструкторские решения или проблемы, возникавшие при ее создании.

Эта инструкция не предназначена для обучения языку Java. Новичкам советуем изучить учебник начального уровня или придерживаться курса лекций по Java.

Эта инструкция не является полным справочником. Многие детали опущены - акцент сделан на ясности и краткости, а не на всестороннем описании особенностей. Более подробное описание см. в *The BlueJ Environment Reference Manual*, на сайте BlueJ (www.bluej.org).

Каждый раздел начинается с краткой информации о содержании. Пользователи, уже знакомые с частями системы, смогут решить, следует ли читать раздел целиком. Раздел 11 фактически является справочником.

1.3 Авторское право, лицензия и распространение

1.4 Обратная связь

Приветствуются и поощряются комментарии, вопросы, исправления, критика и любой другой вид обратной связи по BlueJ или этой инструкции. Пожалуйста отправьте сообщение по почте Майклу Кёллингу (mik@mip.sdu.dk).

2 Инсталляция

BlueJ распространяется в трех различных формах: для систем Windows, для MacOS, и для всех других систем. Установка весьма проста.

Предварительные требования

Для использования BlueJ на Вашем компьютере должна быть установлена J2SE v1.4 (также известная как JDK 1.4) или более новая. Вообще, рекомендуется последняя устойчивая версия Java (не бета-версия). Если у Вас нет установленного JDK, его можно загрузить с сайта Sun Microsystem (http://java.sun.com/j2se/). На MacOS X, свежая версия J2SE уже установлена – не требуется устанавливать ее самостоятельно. Если Вы нашли на страничке загрузки предложение загрузить JRE или SDK, следует загрузить SDK - JRE не достаточно.

2.1 Инсталляция на Windows

Дистрибутивный файл для систем Windows называется bluejsetup-xxx.exe, где xxx - номер версии. Например, дистрибутив BlueJ версии 2.0.0 называется *bluejsetup-200.exe*. Можно получить этот файл на диске, или загрузить его с сайта BlueJ (http://www.bluej.org). Для установки следует выполнить этот файл.

При установке Вы сможете выбрать каталог для системы BlueJ. Кроме того, Вам предложат создать ярлыки в меню «Пуск» и на рабочем столе.

После завершения установки программа bluej.exe может быть найдена в выбранном для BlueJ каталоге.

При первом запуске BlueJ будет искать систему Java (JDK). Если на компьютере имеется более одной подходящей системы Java (например, JDK 1.4.2 и JDK 1.5.0), Вы сможете выбрать, какую систему использовать. Если Java JDK не найдена, Вас попросят указать ее местонахождение (это может случиться, если JDK была установлена, а потом соответствующие записи системного реестра были удалены).

Инсталлятор BlueJ устанавливает также программу *vmselect.exe*. Используя эту программу, Вы сможете позже изменить версию Java, которую должна использовать BlueJ. Для запуска BlueJ с другой версией Java выполните *vmselect*,.

Выбор JDK сохраняется для каждой версии BlueJ. Если Вы установили различные версии BlueJ, можно использовать одну версию BlueJ с JDK 1.4.2, а другую версию BlueJ - с JDK 1.5. Изменение версии Java для BlueJ сохранит эти изменения для всех инсталляций BlueJ той же версии для конкретного пользователя.

2.2 Инсталляция на Macintosh

Пожалуйста, обратите внимание, что BlueJ выполняется только на MacOS X.

Дистрибутивный файл для MacOS называется *BlueJ-xxx.zip*, где *xxx* - номер версии. Например, дистрибутив BlueJ версии 2.0.0 называется *BlueJ-200.zip*. Можно получить этот файл на диске, или загрузить его с сайта BlueJ (http://www.bluej.org).

Обычно MacOS будет распаковывать этот файл автоматически после загрузки. В противном случае, чтобы его распаковать, дважды щелкните по нему.

После распаковки Вы обнаружите папку BlueJ-*xxx*. Переместите эту папку в вашу папку Приложений (или туда, где Вы хотели бы хранить BlueJ). Никаких дополнительных действий по установке не требуется.

2.3 Инсталляция на Linux/Unix и других системах

Общий дистрибутив для Linux/Unix и других систем - выполнимый jar-файл. Он называется *bluej-xxx.jar*, где *xxx* - номер версии. Например, дистрибутив BlueJ версии 2.0.0 называется *bluej-200.jar*. Можно получить этот файл на диске, или загрузить его с сайта BlueJ (http://www.bluej.org).

Запустите инсталлятор, выполняя следующую команду. ВНИМАНИЕ! В примере использован дистрибутивный файл *bluej-200.jar* - Вы должны использовать имя файла, который имеется у Вас (с правильным номером версии).

<*путь к J2se*>/bin/java -jar bluej-200.jar

<*путь к J2se*> - каталог, куда был установлен пакет J2SE SDK.

Появится окно, позволяющее выбрать каталог инсталляции BlueJ и версию Java, которая будет использоваться с BlueJ.

Щелкните *Install*. Когда Install закончит работу, BlueJ будет установлена.

2.4 Проблемы при установке

Если Вы столкнулись с проблемами при установке, просмотрите раздел *Frequently Asked Questions* (FAQ) (страница популярных вопросов) на сайте BlueJ (http://www.bluej.org/help/faq.html) и прочтите раздел *How To Ask For Help* (как попросить помощь) (http://www.bluej.org/help/ask-help.html).

3 Начало - редактирование /компиляция /выполнение

3.1 Запуск BlueJ

На Windows и MacOS программа *BlueJ* уже установлена. Выполните ее.

На системах Unix инсталлятор устанавливает в инсталляционном каталоге сценарий с именем *bluej*. Из графического интерфейса дважды щелкните по файлу. Из командной строки BlueJ можно запустить без параметров или с именем проекта в качестве параметра:

```
$ bluej
```

или

```
$ bluej examples/people
```

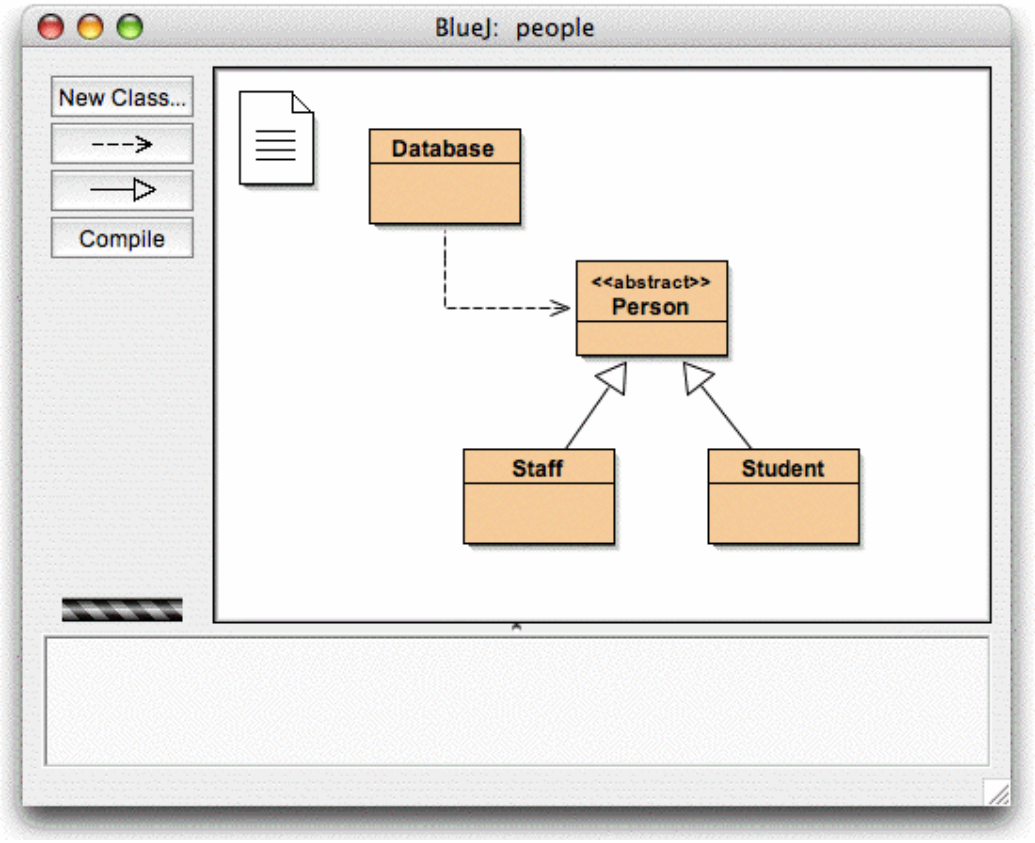

Рисунок 1: Главное окно BlueJ

3.2 Открытие проекта

Кратко: Чтобы открыть проект, выберите Open из меню Project.

Проекты BlueJ, как стандартные пакеты Java, являются каталогами, содержащими файлы проекта.

После запуска BlueJ выполните из меню Project – Open..., чтобы выбрать и открыть проект.

Некоторые примеры проектов включены в стандартный дистрибутив BlueJ и после установки находятся в каталоге *examples*.

Для этого раздела откройте проект *People* из каталога *examples*. Этот каталог находится в основном каталоге BlueJ. После открытия проекта Вы увидите окно, подобное окну, показанному на Рисунке 1. В Вашей системе окно может иметь несколько другой вид, но отличия должны быть незначительными.

3.3 Создание объектов

Кратко: Для создания объекта, выберите конструктор из меню класса.

Одна из главных особенностей BlueJ состоит в том, что можно не только выполнять законченное приложение, но можно и прямо взаимодействовать с отдельными объектами любого класса и выполнять их открытые (public) методы. Выполнение в BlueJ обычно происходит так: 1) создается объект, 2) вызывается один из его методов. Это очень полезно при разработке приложения, т.к. можно тестировать классы по мере их создания. Не требуется сначала готовить все приложение.

> *Замечание: Для выполнения статических методов класса не требуется создание объектов. Один из статических методов может быть "main", так что можно сделать то же, что обычно происходит при запуске Java-приложений – выполняется статический метод main приложения. Мы вернемся к этому позже. Прежде рассмотрим более интересные вещи, которые нельзя сделать в других средах Java.*

Прямоугольники в средней части главного окна (с метками *Database, Person, Staff* и *Student*) - иконки классов, включенных в это приложение. Можно получить контекстное меню с операциями, применимыми к классу, если щелкнуть по иконке класса правой кнопкой мышки (в Macintosh: ctrl-click[1]) (Рисунок 2). Операции в меню – это (при помощи new) конструкторы класса и ряд общих действий, предусмотренные в BlueJ для классов.

[1] Там, где указано «щелчок правой кнопкой», пользователи Macintosh должны читать ctrl-click.

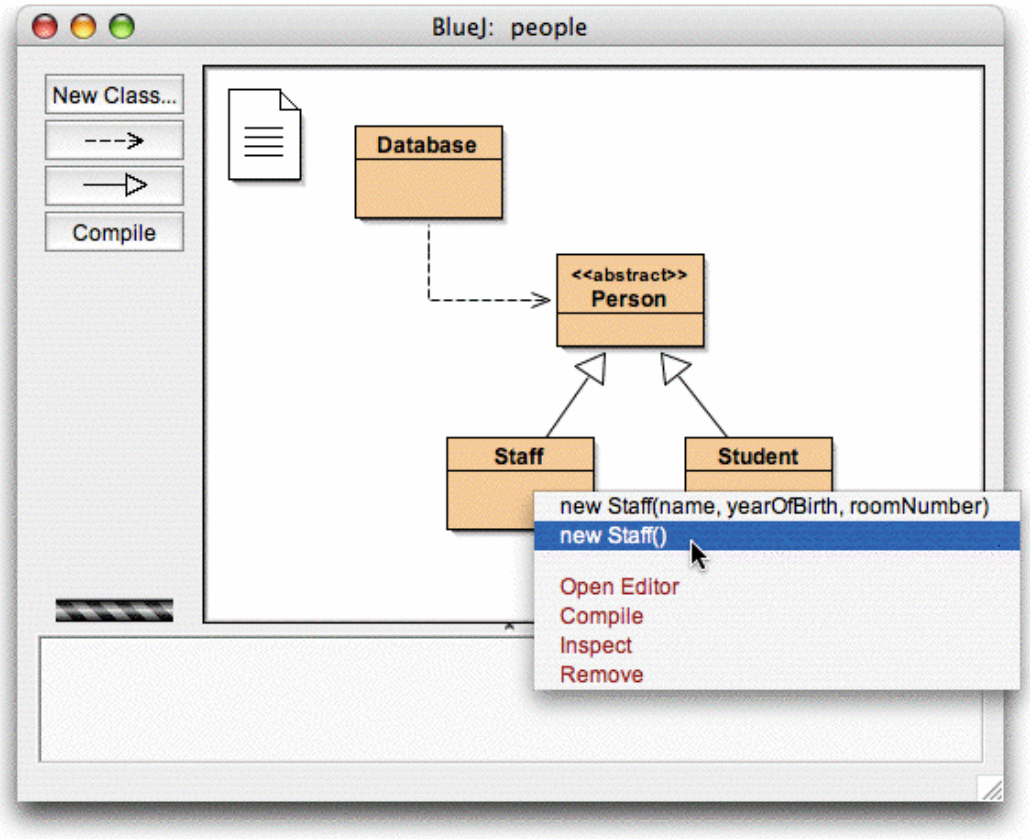

Рисунок 2: Операции для класса в контекстном меню

Мы хотим создать объект *Staff*, так что следует щелкнуть правой кнопкой мышки по пиктограмме *Staff* (тогда появится меню, показанное на Рисунке 2). Меню содержит два конструктора для создания объекта *Staff*, один конструктор - с параметрами и один – без параметров. Сначала выберем конструктор без параметров. Появится окно диалога, показанное на Рисунке 3.

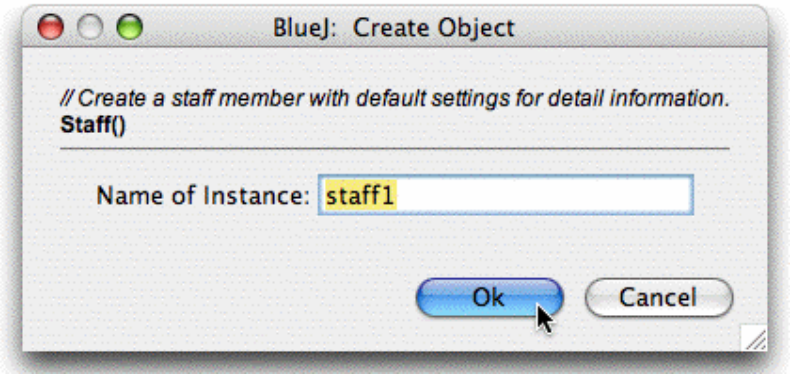

Рисунок 3: Создание объекта конструктором без параметров

Этот диалог запрашивает имя создаваемого объекта. В поле имени система BlueJ предлагает имя, которое образовано из имени класса - *staff1*. Если оно Вам подходит, просто щелкните по кнопке *OK*, и объект класса *Staff* будет создан.

Созданный объект помещается на стенд объектов (Рисунок 4). Вот и все, что требуется для создания объекта – выбрать из меню класса конструктор, выполнить его, и Вы получаете объект на стенде объектов.

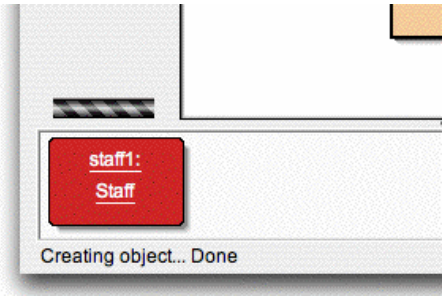

Рисунок 4: объект staff1 на стенде объектов

Вы, возможно, обратили внимание, что класс *Person* имеет пометку «abstract» (это - абстрактный класс). Если Вы попытаетесь создать объект этого класса, Вы обнаружите, что это сделать нельзя - спецификация языка Java запрещает создание объектов абстрактных классов.

3.4 Выполнение

Кратко: Чтобы выполнить метод, выберите его из контекстного меню объекта.

Если объект создан, можно выполнить его открытые (public) методы. (В Java действия, которые можно производить с объектами, называют *методами*). Щелчок правой кнопкой мышки на объекте - и появляется контекстное меню (Рисунок 5). Меню показывает методы, доступные для этого объекта и две специальных операции, выполняемые средой BlueJ (*Inspect* and *Remove* - *Инспектировать* и *Удалить*). Мы обсудим их позже. Сейчас давайте думать о методах.

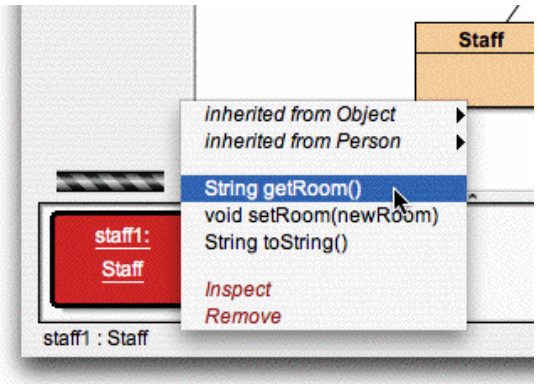

Рисунок 5: Контекстное меню объекта

Вы видите, что есть методы *setRoom* и *getRoom*, которые задают и возвращают номер комнаты для этого сотрудника. Попробуйте вызвать метод *getRoom*. Просто выберите его из меню объекта, и он будет выполнен. Результат появится в окне диалога (Рисунок 6). Мы увидим, что результат – строка «unknown room» («неизвестная комната»). Такое значение показывает, что мы еще не задавали номер комнаты для этого сотрудника.

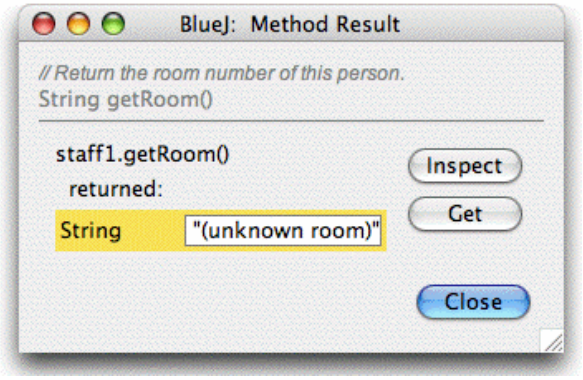

Рисунок 6: Форма с результатом выполненного метода

Методы, унаследованные от суперкласса, доступны через подменю. В верхней части контекстного меню объекта есть два подменю, одно для методов, унаследованных от класса *Object* и одно для методов, унаследованных от класса *Person* (Рисунок 5). Можно вызвать методы класса *Person* (например, *getName*), выбирая их из подменю. Попробуйте это сделать. Вы заметите, что результат также неопределенный: это результат “(unknown name)” («неизвестное имя»), потому что мы не дали нашему сотруднику имя.

Теперь попробуем задать номер комнаты. Мы увидим, как вызывать метод, который имеет параметры. (Вызовы *getRoom* и *getName* имели возвращаемые значения, но не имели параметров). Вызовите метод *setRoom*, выбрав его из меню. Диалог позволит задать значение параметра (Рисунок 7).

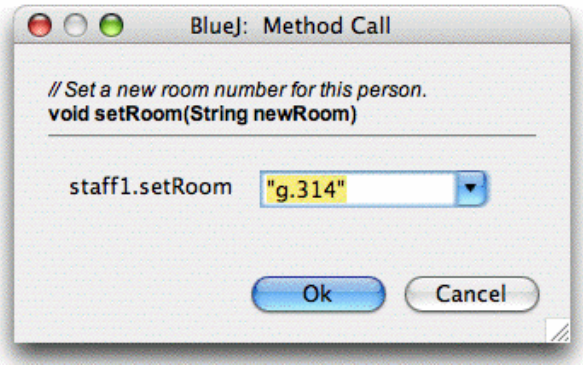

Рисунок 7: Задание параметров метода

В верхней части диалога показан интерфейс вызванного метода (включая комментарий и сигнатуру). Ниже - текстовое поле для ввода значения параметра. Сигнатура говорит нам, что методу требуется один параметр типа "String". Вве-

дите в текстовое поле номер комнаты как строку (включая кавычки) и щелкните *OK*.

Вот и все. Так как этот метод не возвращает результат, окно диалога с результатом не появится. Вызовите *getRoom* снова, чтобы проверить, что номер комнаты действительно изменился.

Потренируйтесь в создании объектов и вызове методов. Попробуйте вызвать конструктор с параметрами и вызвать другие методы, пока не почувствуете, что Вы это усвоили.

3.5 Редактирование класса

Кратко: Чтобы редактировать исходный код класса, дважды щелкните его значок класса.

Пока мы имели дело только с интерфейсом объекта. Давайте теперь заглянем внутрь**.** Вы можете увидеть реализацию класса, выбрав *Open Editor* из операций класса. (Напомним: щелчок правой кнопкой мышки по значку класса покажет контекстное меню с операциями класса.) Двойной щелчок по значку класса даст такой же результат. В этой инструкции редактор не описывается детально, но он очень прост для использования. Подробности работы с редактором будут описаны отдельно позже. Пока же, откройте реализацию класса *Staff*. Найдите текст метода *getRoom*. Он возвращает, как говорит его имя, номер комнаты сотрудника. Давайте изменим метод, разместив перед результатом слово “`room`” (так, чтобы метод возвратил, например, “room M.3.18”, а не только "M.3.18"). Мы можем сделать это, изменив строку

```
return room;
```

так:

```
return "room " + room;
```

BlueJ поддерживает полный, немодифицированный язык Java, так что нет ничего особенного в реализации Ваших классов.

3.6 Компиляция

Кратко: Чтобы компилировать класс, щелкните по кнопке Compile в редакторе. Чтобы компилировать проект, щелкните по кнопке Compile в окне проекта.

После вставки текста (до того, как сделать что-то еще), проверьте вид проекта в главном окне. Вы заметите, что значок класса *Staff* изменился: на нем появилась штриховка. Штриховка отмечает классы, которые не компилировались после изменения их текста. Вернемся в редактор.

Замечание: Вы можете спросить, почему значки класса не были заштрихованы сразу после открытия проекта. Ответ - потому что классы в проекте уже были откомпилированными. Часто проекты BlueJ поставляются нескомпилированными, так что при первом открытии проекта следует ожидать, что большинство значков будет заштриховано.

На панели инструментов в верхней части окна редактора находятся кнопки для часто выполняемых действий. Одна из кнопок - *Compile (Компилировать* - позволяет компилировать класс, не выходя из редактора. Щелкните по кнопке *Compile*. Если текст - без ошибок, то в информационной области внизу редактора должно появиться сообщение о том, что класс был откомпилирован. Если текст содержит синтаксическую ошибку, строка с ошибкой будет подсвечена, и в информационной области появится сообщение об ошибке. (В случае, если компиляция удалась с первого раза, пробуйте ввести синтаксическую ошибку - например, удалить точку с запятой - и выполнить компиляцию повторно, чтобы увидеть, как выглядит сообщение об ошибках).

После того, как класс успешно скомпилирован, закройте редактор.

Замечание: нет необходимости явно сохранять исходный текст класса после его изменения. Исходный текст автоматически сохраняется, когда это необходимо (например, когда редактор закрывается или перед компиляцией класса). Вы можете, если хотите, явно сохранить текст (пункт Save в меню Class редактора), но это действительно необходимо только в случае, если ваша система нестабильна и часто зависает, а Вы не хотите потерять проделанную работу.

В окне проекта на панели инструментов также есть кнопка *Compile*. Щелчок по ней запускает компиляцию всего проекта (Фактически, определяется, какие классы требуют компиляции, после чего эти классы компилируются в правильном порядке). Проверьте эту возможность, изменив текст не менее двух классов (так, чтобы не менее двух классов на диаграмме оказались заштрихованными) и затем щелкните по кнопке *Compile*. Если в одном из компилируемых классов будет обнаружена ошибка, откроется редактор, место ошибки будет подсвечено, и появится диагностическое сообщение.

Отметьте, что стенд объектов снова пуст. Объекты всегда удаляются со стенда после изменения текста классов.

3.7 Справка по ошибкам компилятора

Кратко: Чтобы получить справку для сообщения об ошибках компилятора, щелкните по вопросительному знаку рядом с сообщением об ошибках.

Очень часто у начинающих студентов возникают проблемы с пониманием сообщений об ошибках компиляции. Мы попытались помочь им.

Снова откройте редактор, внесите в текст ошибку, и запустите компиляцию. В информационной области появится сообщение об ошибке. Справа от сообщения

появится вопросительный знак, щелкнув по которому, можно будет получить дополнительную информацию об этом типе ошибки (Рисунок 8).

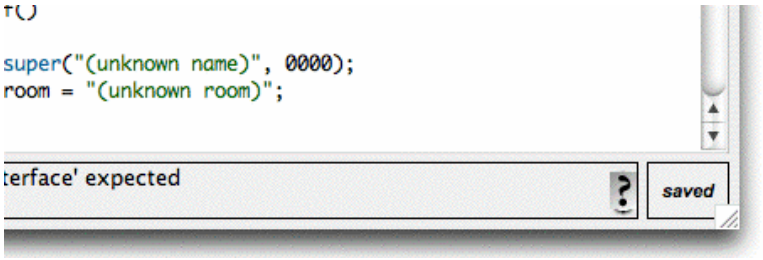

Рисунок 8: Ошибка компиляции и кнопка помощи «?»

Сейчас дополнительная информация имеется не для всех сообщений об ошибках. Некоторые пояснения еще только должны быть написаны. Но стоит использовать то, что есть - для многих ошибок пояснения уже составлены. В следующие версии BlueJ будут включены пояснения к оставшимся ошибкам.

4 Идем дальше...

В этом разделе мы познакомимся с дополнительными возможностями. Эти возможности не являются главными, но используются очень часто.

4.1 Инспекция объекта

Кратко: Инспекция объекта позволяет увидеть внутреннее состояние объекта, что является одним из простых способов отладки.

При выборе метода объекта для выполнения, Вы, вероятно, заметили операцию *Inspect*, которая доступна для объектов в дополнение к методам, определенным пользователем (Рисунок 5). Эта операция позволяет проверять состояния объектов, т.е. значения его переменных – "полей". Попробуйте создать объект с небольшим количеством задаваемых пользователем значений (например, объект *Staff* с конструктором, который принимает параметры). Затем выберите *Inspect* из контекстного меню. Появится окно диалога, в котором будут отображены поля объекта, их типы и значения (Рисунок 9).

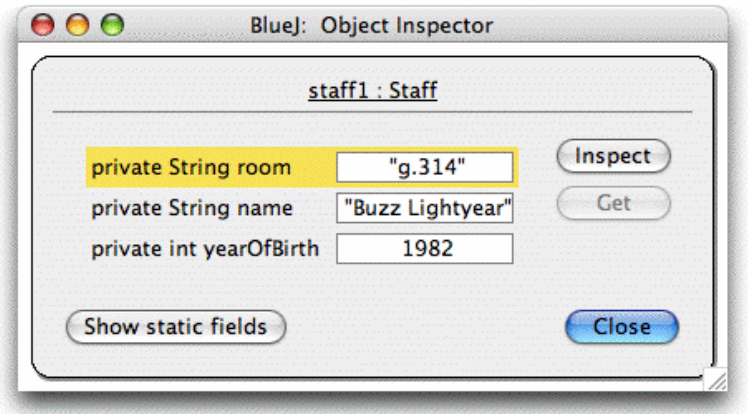

Рисунок 9: Диалог просмотра состояния объекта

Инспекция полезна для быстрой проверки правильности работы *мутаторов* – методов, которые изменяют состояние объекта. Таким образом, инспекция - это простое средство отладки.

В примере с классом *Staff* все его поля - простые типы (не объектные типы и не строки). Значение этих типов можно показать непосредственно. Вы можете сразу увидеть, сделал ли конструктор правильные присваивания.

В более сложных случаях, значения полей могут быть ссылками на объекты, определенные пользователем. Чтобы рассмотреть такой пример, нам потребует-

ся другой проект. Откройте проект *people2*, который также включен в стандартный дистрибутив BlueJ. Окно проекта для *people2* показано на Рисунке 10. В этом проекте в дополнение к классам, которые Вы видели ранее, имеется класс *Address*. Одно из полей в классе *Person* имеет определенный пользователем тип *Address.*

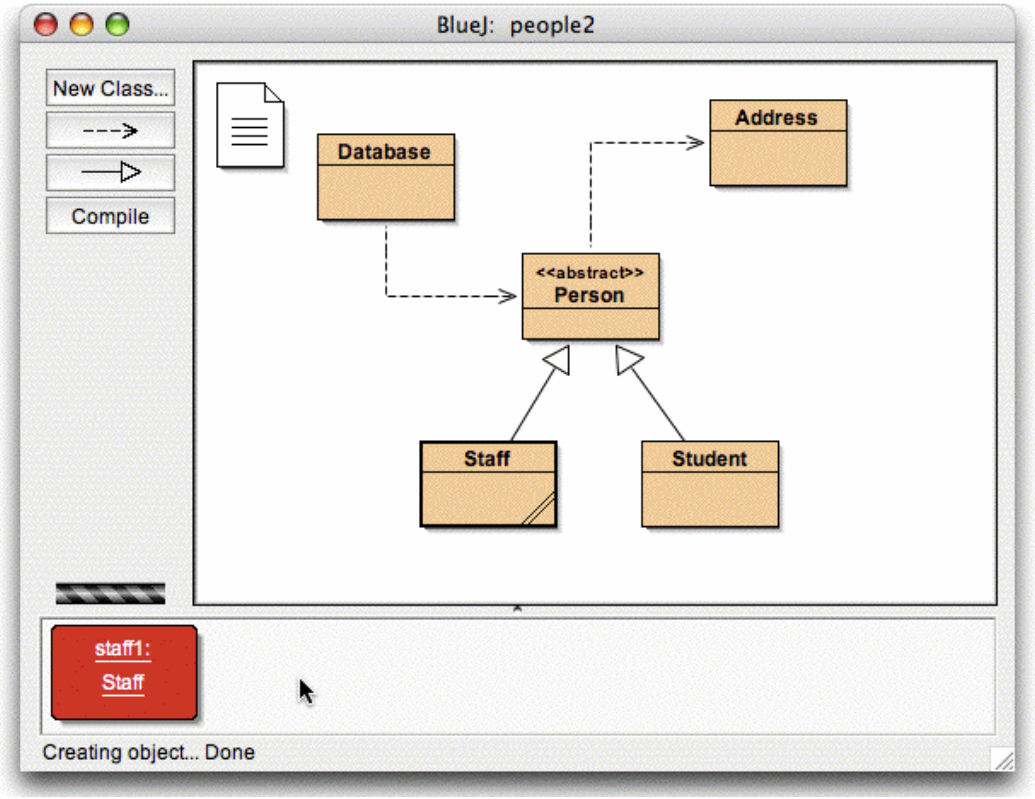

Рисунок 10: Окно проекта people2

Далее мы попробуем инспектировать объект, поля которого – объекты. Для этого создадим объект класса *Stuff* и вызовем метод *setAddress* этого объекта (Вы найдете его в подменю методов, унаследованных от класса *Person*). Введите адрес (введено “Gyldenstenvej 29”, “Odense”, “5230”)[1]. Метод *setAddress* создаст объект типа *Address* (используя конструктор этого класса), сохранит введенные значения в этом объекте, и ссылку на объект поместит в переменную *address*.

Теперь проинспектируем объект staff1. Окно диалога показано на Рисунке 11. Поля объекта staff1 теперь включают *address*. Как видно, его значение показано как стрелка, отсылающая к другому объекту. Так как *address* - сложный, определяемый пользователем объект, его значение нельзя показать непосредственно в этом окне. Чтобы проинспектировать поле *address*, выберите его, щелкнув по нему (при выборе оно изменит цвет), затем щелкните по кнопке *Inspect* в окне диалога. (Тот же результат даст двойной щелчок по полю *address* или по стрел-

[1] При создании объекта staff1 будут созданы и поля базового класса (*Person*), в частности, поле address типа *Address*.

ке). Будет открыто новое окно инспектора объектов, в котором мы увидим состояние объекта *address* типа *Address*. (Рисунок 12).

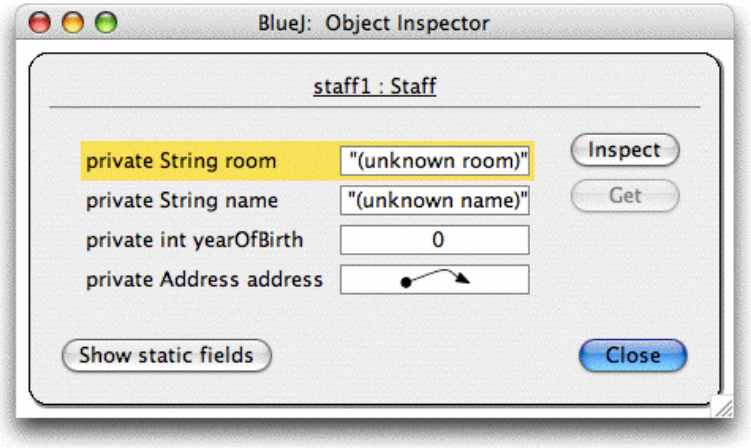

Рисунок 11: Инспекция объекта со ссылкой на объект

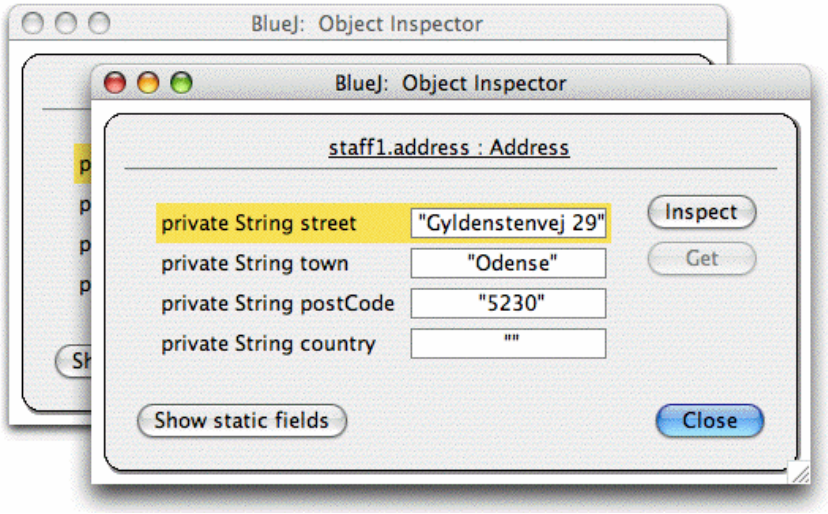

Рисунок 12: Инспекция встроенного объекта

Если бы выбранное поле имело атрибут public, тогда, вместо щелчка по Inspect, можно было бы также выбрать поле address и щелкнуть по кнопке Get. Объект, связанный с полем, был бы помещен на стенд объектов. Можете провести такое исследование, вызывая методы объектов.

4.2 Передача объектов как параметров

Кратко: Объект можно передать методу как параметр, щелкнув по пиктограмме объекта.

Объекты можно передавать как параметры методам других объектов. Давайте исследуем пример. Создадим объект класса *Database*. (Обратите внимание, что класс *Database* имеет единственный конструктор без параметров, так что создается без проблем.). Объект типа *Database* предназначен для хранения списка людей. Он имеет операции для добавления человека, для отображения всех людей, хранящихся в списке. (Конечно, называть такой объект «База данных» - преувеличение!)

Если у Вас на стенде объектов еще нет объектов типа *Staff* или *Student*, создайте один из них. В дальнейшем нам нужны будут на стенде объект типа *Database* и объект одного из типов - *Staff или Student*[1].

Теперь выполните метод *addPerson*[2] объекта типа *Database*. Сигнатура метода показывает, что ему требуется параметр – объект типа *Person.* (Напоминаем: класс *Person* - абстрактный, так что нельзя создать объекты этого класса. Но, поскольку классы *Student* и *Staff* являются подклассами класса *Person*, объекты типов *Student* и *Staff* могут быть использованы вместо объекта типа *Person*. Так что допустимо передать объект типа *Student* или *Staff* туда, где требуется объект типа *Person*). Чтобы передать объект со стенда как параметр методу, который Вы вызываете, Вы просто должны ввести его имя в поле параметра или щелкнуть по объекту на стенде. Это введет его имя в диалог вызова метода. Щелкните *OK*, и вызов сделан. Однако, так как метод *addPerson* не возвращает никакого значения, мы не увидим, что добавление произошло. Можно вызвать метод *listAll* для объекта *database1*, и, просматривая выведенные данные, убедиться, что *addPerson* действительно выполнился. Метод *listAll* выводит в окно терминала (стандартный вывод) информацию, хранящуюся в базе данных. Обратите внимание, что окно терминала открывается автоматически.

Попробуйте проделать это снова с несколькими людьми, добавляя их в "базу данных".

---

[1] На стенде – объект database1 класса Database и объект staff1 класса Staff.

[2] AddPerson – добавить человека. Метод применяется для добавления людей в базу данных.

5 Создание нового проекта

Краткий тур: создание нового проекта.

5.1 Создание каталога для проекта

Кратко: Чтобы создать проект, выберите New ... из меню Project.

Чтобы создать новый проект, выберите Project - New ... из меню. Откроется диалог для выбора файла, который позволит Вам определить имя и местоположение нового проекта. Попробуйте сделать это. Можно задать любое имя для нового проекта. Щелчок по OK - и будет создан каталог с тем именем, которое Вы определили, а главное окно покажет новый пустой проект.

5.2 Создание классов

Кратко: Чтобы создать класс, щелкните по кнопке New Class и задайте имя класса.

Классы можно создавать, щелкая по кнопке *New Class* на панели инструментов проекта. Вас должны задать имя класса - это имя должно быть правильным идентификатором Java.

Можно выбрать один из четырех типов классов: абстрактный, интерфейс, апплет или "стандартный". В зависимости от выбора для генерации текста класса будет применен один из четырех шаблонов. Впоследствии можно будет изменить текст класса (и тип), пользуясь редактором (например, добавить ключевое слово "abstract" в определение класса и превратить его в абстрактный).

После создания класс представляется значком на диаграмме проекта. Если класс - не стандартный, его тип (interface, abstract, или applet) указан на значке класса. Когда Вы впервые открываете новый класс для редактирования, Вы должны заметить, что уже создан «скелет» класса – синтаксически правильный код. Класс может быть откомпилирован, но, конечно, он не делает ничего, связанного с Вашей задачей. Попробуйте создать несколько классов и откомпилировать их.

5.3 Создание зависимостей

Кратко: Чтобы создать зависимость (изображаемую стрелкой), щелкните по кнопке стрелки и перетащите стрелку в диаграмму проекта, или просто напишите исходный текст в редакторе.

Диаграмма классов показывает зависимости между классами в форме стрелок. Отношения наследования (extends -"расширяет", или implements -"реализует") показывают как стрелки со сплошной линией; отношения "использует" показывают как стрелки со штриховой линией.

Можно добавить зависимости графически (добавляя стрелки к диаграмме) или непосредственно в исходном тексте. Если стрелка добавляется графически, текст автоматически модифицируется; если зависимость добавляется в тексте, модифицируется диаграмма.

Чтобы добавить стрелку графически, щелкните по кнопке нужной стрелки (сплошная стрелка для "extends" или "implements", штриховая стрелка - для "использует"), щелкните по классу, из которого стрелка должна выходить, затем щелкните по классу, в который стрелка должна входить.

Добавление стрелки наследования вставляет служебные слова "extends" или "implements" в исходный текст класса (в зависимости от того, является ли адресат классом или интерфейсом).

Добавление стрелки "использует" не изменяет текст немедленно (кроме использования класса из другого пакета. В этом случае генерируется инструкция "import", но таких примеров мы еще не видели). Если стрелка "использует" на диаграмме указывает на класс, который фактически не используется, генерируется предупреждение о том, что отношение "использует" было объявлено, но класс не используется.

Легко добавить стрелки, просто изменив в редакторе текст класса. После сохранения класса диаграмма будет модифицирована. (Не забудьте: закрытие редактора автоматически сохраняет текст.)

5.4 Удаление элементов

Кратко: Чтобы удалить класс или стрелку, выберите действие Remove из контекстного меню.

Чтобы удалить класс с диаграммы, выберите класс и затем выберите *Remove* из меню *Edit*. Можно также выбрать *Remove* из контекстного меню класса. Оба варианта работают и для стрелок: можно сначала выбрать стрелку и затем - *Remove* из меню, или можно использовать контекстное меню стрелки.

6 Использование панели кода

Панель кода BlueJ позволяет выполнить простую и быструю оценку произвольных фрагментов кода Java (выражений и операторов). Таким образом, панель кода может использоваться, чтобы исследовать особенности семантики Java, иллюстрировать работу операторов и экспериментировать с синтаксисом Java.

6.1 Открытие панели кода

Кратко: Для работы с панелью кода, выберите Show Code Pad из меню View.

Панель кода по умолчанию не видна. Чтобы показать ее, используйте пункт *Show Code Pad* из меню *View*. Главное окно будет теперь включать панель кода, расположенную в нижней части справа, рядом со стендом объектов (Рисунок 13). Горизонтальные и вертикальные границы панели кода и стенда объектов можно перемещать, если требуется изменить их размеры.

Панель кода позволяет вводить выражения или операторы. При нажатии *Enter*, каждая введенная строка будет выполнена, и Вы увидите результат.

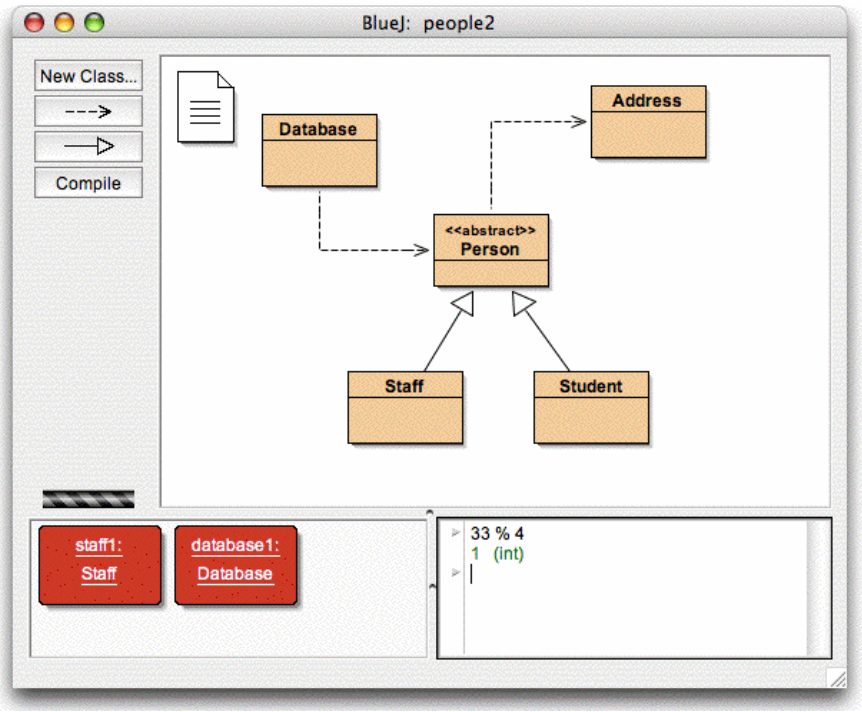

Рисунок 13: Главное окно с панелью кода

6.2 Вычисление простых выражений

Кратко: Чтобы вычислить выражение, введите его в панель кода и нажмите Enter.

Панель кода позволяет оценить значения простых выражений. Введите, например:

```
4 + 45
"Hello" .length ()
Math.max (33, 4)
(int) 33.7
javax.swing. JOptionPane.showInputDialog (null, "Name")
```

В выражениях можно использовать стандартные элементы и объекты Java, а также классы из текущего проекта. Панель кода отобразит значение результата с указанием его типа (в скобках). Если выражение неправильно, будет выведено сообщение об ошибках,.

Можете также использовать объекты, находящиеся на стенде. Попробуйте поместите объект класса *Student* на стенд, использовав контекстное меню класса. Назовите объект *student1*.

В панели кода введите оператор

```
student1.getName ()
```

Так же можно обратиться ко всем доступным методам классов Вашего проекта.

6.3 Перемещение объектов с панели кода на стенд

Кратко: Для перемещения объекта с панели кода на стенд, перетащите маленькую пиктограмму объекта.

Результаты некоторых выражений - объекты, а не простые значения. В этом случае результат показывается как *<ссылка на объект>* с указанием типа объекта. В строке результата слева выводится маленькая пиктограмма объекта (Рисунок 14).

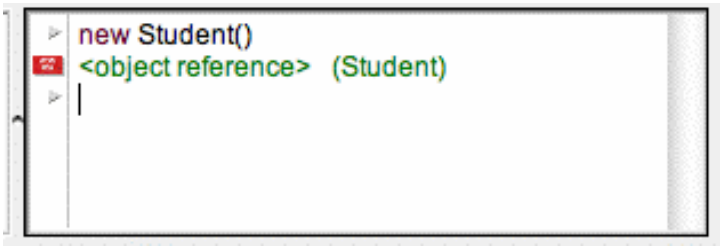

Рисунок 14: Объект как результат выражения, введенного в панели кода

Если результат - строка, значение строки будет отображено, но Вы увидите также маленький значок объекта (так как строки - объекты).

Вот примеры выражений, при помощи которых можно создать объекты

```
new Student()
"marmelade".substring(3,8)
new java.util.Random()
"hello" + "world"
```

Маленький значок объекта можно теперь использовать для продолжения работы с объектом-результатом. Можно указать на значок и перетащить его на стенд (Рисунок 15). При перетаскивании от Вас потребуется задать имя объекта. Когда объект будет на стенде, его методы будут доступны или через контекстное меню или при наборе оператора вызова метода в панели кода.

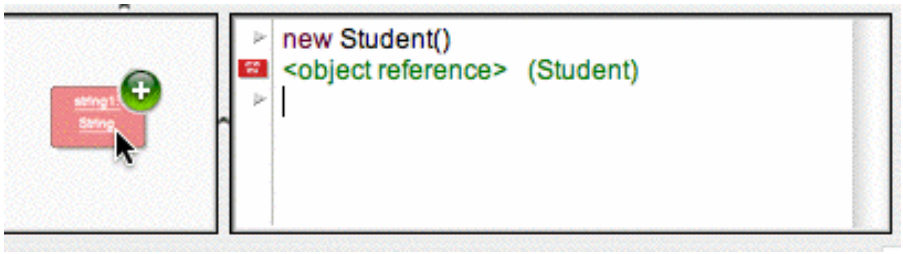

Рисунок 15: Перемещение объекта на стенд

6.4 Инспекция объектов

Кратко: Чтобы инспектировать объекты, полученные как результат в панели кода, дважды щелкните по маленькому значку объекта.

Если требуется инспектировать объект, который является результатом выражения, набранного в панели кода, не обязательно помещать объект на стенд. Можно дважды щелкнуть по значку объекта, чтобы открыть инспектор объектов.

6.5 Выполнение операторов

Кратко: Операторы, набранные в панели кода, выполняются.

Можно использовать панель кода для выполнения операторов, которые не возвращают значений. Попробуйте, например:

```
System.out.println("Gurkensalat");
System.out.println(new java.util.Random().nextInt(10));
```

В конце оператора обязательно должна быть точка с запятой.

6.6 Многострочные операторы и последовательности операторов

Кратко: shift-Enter в конце строки является признаком продолжения оператора (цепочки операторов) в следующую строку.

Можете ввести последовательности операторов или операторы, занимающие несколько строк, используя *shift-Enter* в конце промежуточной строки. Нажатие *shift-enter* переместит курсор в начало следующей строки, но не приведет к выполнению оператора. В конце последней строки нажмите просто Enter. Попробуйте, например, *для* цикла:

```
for (int i=0; i<5; i++) {
  System.out.println("number: " + i);
}
```

6.7 Работа с переменными

Кратко: В одном многострочном операторе можно использовать локальные переменные. Имена объектов со стенда могут использоваться как поля объекта «Стенд».

Переменные - поля объектов и локальные переменные - могут использоваться в панели кода с ограничениями.

Можно объявить локальные переменные в панели кода, но это имеет смысл только для одного многострочного оператора, так как эти переменные исчезают после нажатия Enter и выполнения оператора. Например: можно ввести следующий фрагмент как многострочный оператор, и он будет работать как ожидается:

```
int sum;
sum = 0;
for (int i=0; i<100; i++) {
   sum += i;
}
System.out.println("The sum is: " + sum);
```

Вводя ту же самую последовательность отдельными строками (т.е. нажимая в конце каждой строки Enter), мы не получим результат, так как локальная переменная sum перестает существовать после нажатия Enter и выполнения оператора.

Вы можете думать о вводимом в панель кода тексте как о тексте в пределах тела метода. Любые операторы, допустимые для записи в теле метода, допустимы также в панели кода. Однако части кода, разделенные нажатиями Enter, отно-

сятся к *разным* методам, и Вы не можете обращаться из одной части к переменным, объявленным в другой части[1].

Вы должны думать об объектах на стенде как о полях специального объекта «стенд». Нельзя определить никаких новых полей этого объекта из панели кода, также как нельзя определить новые поля обычного объекта в методе объекта. Однако можно обратиться к «полям» объекта «стенд» (объектам проекта) и выполнять их методы.

Можно создать новое «поле» на стенде, переместив объект с панели кода на стенд.

6.8 История команд

Кратко: Стрелка "вверх" и стрелка "вниз" перемещают по набранным и выполненным в панели кода строкам.

Панель кода хранит историю набранных строк. Используя клавиши «вверх» и «вниз», можно выбрать предыдущие строки, которые могут быть отредактированы и снова выполнены.

.

[1] Указанное ограничение относится к переменным простых типов, а не к объектам.

7 Отладка

Этот раздел вводит самые важные аспекты функциональных возможностей отладки в BlueJ. Беседуя с преподавателями программирования, мы часто слышали, что было бы хорошо знакомить студентов с использованием отладчика на первом году обучения, но для этого совершенно нет времени. Студенты сражаются с редактором, компилятором, выполнением, а на знакомство с таким важным инструментом времени не остаётся.

Именно поэтому мы решили упростить отладчик, насколько это возможно. Преследовалась цель создать отладчик, правила работы с которым можно было бы объяснить за 15 минут, после чего студенты смогли бы пользоваться им самостоятельно. Давайте посмотрим, насколько это удалось.

Прежде всего, мы ограничились тремя функциями традиционных отладчиков:

- установка точек прерывания
- пошаговое выполнение кода
- контроль значений переменных

Каждая из этих задач очень проста. Выполним их по порядку.

Начнем с открытия проекта *debugdemo*, включенного в каталог *examples* дистрибутива. Этот проект содержит несколько классов, позволяющих продемонстрировать возможности отладчика – для другого в них смысла мало.

7.1 Установка точек прерывания

Кратко: Чтобы установить точку прерывания в строке, щелкните в области номеров строк и точек прерывания редактора - слева от текста строки.

Установка точки прерывания позволяет прерывать выполнение кода в некотором месте. Когда выполнение прервано, можно исследовать состояние объектов. Это помогает понять, что происходит в Вашем коде.

В редакторе, слева от текста, имеется область точек прерывания и номеров строк (Рисунок 16). Вы можете установить точку прерывания, щелкнув по ней. Маленький значок «Стоп» отметит точку прерывания. Попробуйте это сделать. Откройте класс *Demo*, найдите метод *loop*, и установите точку прерывания где-нибудь в цикле *for*. Значок точки прерывания должен появиться в выбранной строке. После установки точки прерывания редактор можно закрыть.

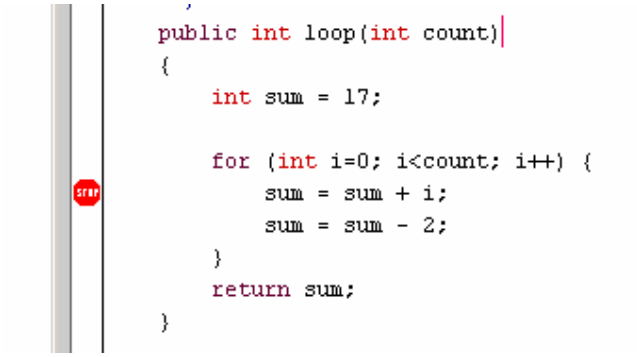

Рисунок 16: Установленная точка прерывания

Когда выполнение программы дойдет до точки прерывания, программа будет приостановлена. Давайте попробуем.

Создайте объект класса *Demo*, и вызовите метод *loop* с параметром, скажем, 10. Как только точка прерывания будет достигнута, откроется окно редактора с указанием текущей строки программы, и появится окно отладчика (Рисунок 17).

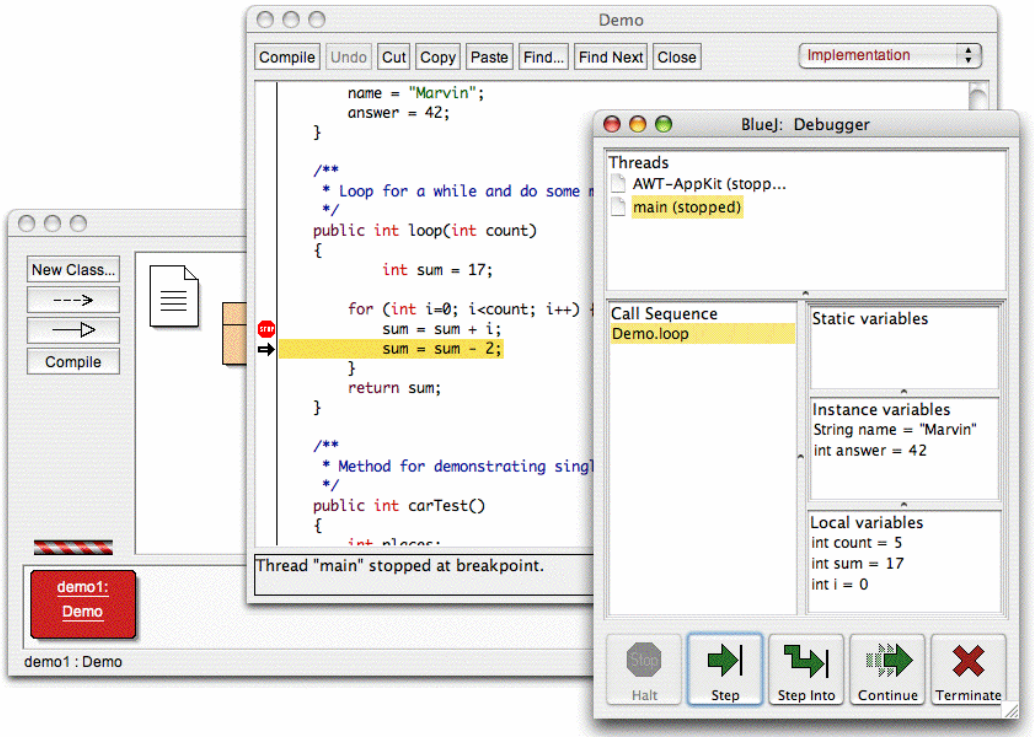

Рисунок 17: Окно отладчика после однократного нажатия Step

Подсветка в редакторе показывает строку, которая будет выполнена следующей. (Выполнение прервано *до* выполнения этой строки.)

7.2 Продвижение по коду

Кратко: Для пошагового выполнения кода используйте кнопки Step и Step into.

Теперь, когда выполнение программы прервано (значит, метод действительно начал выполняться, и точка прерывания достигнута), мы можем пройти код по шагам, наблюдая за изменениями. Для этого несколько раз нажмите кнопку *Step* в окне отладчика. Вы должны видеть, как изменяется положение подсвеченной строки в тексте программы (подсвечивается та строка, которая должна быть выполнена). Каждый раз, когда Вы щелкаете по кнопке *Step*, выполняется одна строка кода, и выполнение снова приостанавливается. Обратите внимание, что значения переменных, отображаемые отладчиком, тоже изменяются (например, значение *sum*), так что можно выполнять программу шаг за шагом и наблюдать за тем, что получается. Если этот процесс утомил Вас, можно щелкнуть по значку точки прерывания, чтобы удалить ее, а затем щелкнуть по кнопке *Continue* в отладчике, и выполнение программы будет продолжено без прерываний (в этой точке).

Попробуем все это еще раз с другим методом. Установите точку прерывания в методе *carTest()* класса *Demo*, в строке

```
places = myCar.seats();
```

Вызовите метод *carTest()*. Когда точка прерывания достигнута, следующей должна быть выполнена строка, содержащая вызов метода *seats()* класса *Car*. Если щелкнуть по *Step,* строка будет выполнена целиком. Давайте теперь попробуем *Step into*. Если нажать *Step into*, Вы попадете внутрь метода и сможете выполнить его строка за строкой по шагам (а не все вместе за один шаг). В нашем случае, Вы попадете в метод *seats()* класса *Car*. Вы можете теперь весело прощелкать этот метод до конца, после чего возвратитесь в метод *carTest()*. Обратите внимание, как изменяется окно отладчика.

Кнопки *Step* и *Step Into* действуют одинаково, если в строке отсутствуют вызовы методов.

7.3 Контроль значений переменных

Кратко: Контроль переменных прост – их значения автоматически отображаются в отладчике.

Когда ведется отладка, важно иметь возможность контролировать состояние программных объектов (значения локальных переменных и значения полей объекта).

Сделать это просто – бóльшую часть Вы уже видели. Вам не нужны специальные команды для контроля переменных; статические переменные, переменные текущего объекта и локальные переменные текущего метода отображаются и обновляются автоматически.

Вы можете выбрать методы в цепочке вызовов, чтобы просмотреть переменные других объектов и методов, активных в настоящий момент. Испытайте снова, например, точку прерывания в методе *carTest( )*. В левой части окна Вы увидите панель Call sequence - последовательность вызовов методов. В настоящий момент там видны

```
Car.seats
Demo.carTest
```

Это последовательность показывает, что метод `Car.seats` вызван методом `Demo.carTest`. Можно выбрать `Demo.carTest` в этом списке и посмотреть исходный код и текущие значения переменных этого метода.

Строкой выше точки прерывания расположен оператор `Car myCar = new Car(…)`. Отметьте, что значение локальной переменной `myCar` показано как ссылка на объект (<*object reference*>). Все значения объектных типов (кроме String) показывают таким способом. Можно посмотреть значение этой переменной (объекта), дважды щелкнув по ней. После щелчков откроется окно инспектора объектов, идентичное описанному ранее (раздел 4.1). В действительности нет никакого различия между инспекцией объектов в отладчике и инспекцией на стенде.

7.4 Halt - остановить и Terminate -закончить

Кратко: Кнопки Halt (Остановить) *и* Terminate (Закончить) *используются, чтобы приостановить выполнение на время или прекратить выполнение.*

Иногда программа долго выполняется, и Вам интересно, все ли в порядке. Возможно, в программе - бесконечный цикл, возможно – просто длинные и сложные вычисления. Пользуясь отладчиком, Вы можете проверить, в чем дело. Вызовите метод *longloop ( )* класса *Demo*. Его выполнение длится довольно долго.

Предположим, Вы хотите узнать, что происходит. Откройте окно отладчика, если оно еще не открыто.

Теперь щелкните по кнопке *Halt*. Выполнение прервется так же, как если бы была достигнута точки прерывания. Вы можете теперь выполнить несколько шагов, наблюдая за значениями переменных, и убедиться, что все нормально - только для завершения требуется немного больше времени. Можно несколько раз нажать *Continue* и *Halt*, чтобы увидеть, как быстро увеличивается переменная i. Если Вы не хотите продолжить (например, Вы обнаружили, что действительно находитесь в бесконечном цикле), следует нажать *Terminate* и закончить выполнение метода. *Terminate* не стоит нажимать слишком часто - останавливая виртуальную машину, можно оставить объекты в неопределенном состоянии, так что желательно использовать *Terminate* только в аварийных ситуациях.

8 Создание автономных приложений

Кратко: Чтобы создать автономное приложение, выберите Project - Create Jar File...

BlueJ может создать выполнимые jar-файлы. Такие jar-файлы могут быть выполнены в некоторых системах при двойном щелчке по файлу (например, в Windows и MacOS X), или набором команды `java -jar <имя файла>.jar` (в командной строке Unix или DOS).

Потренируемся на проекте *hello*. Откройте его (он находится в каталоге *examples*). Убедитесь, что проект откомпилирован. Выберите *Create Jar File...* из меню *Project*.

Откроется окно диалога, в котором Вы сможете определить главный (main) класс (Рисунок 18). В этом классе должен быть определен *метод main* (с сигнатурой `public static void main(String[] args)`).

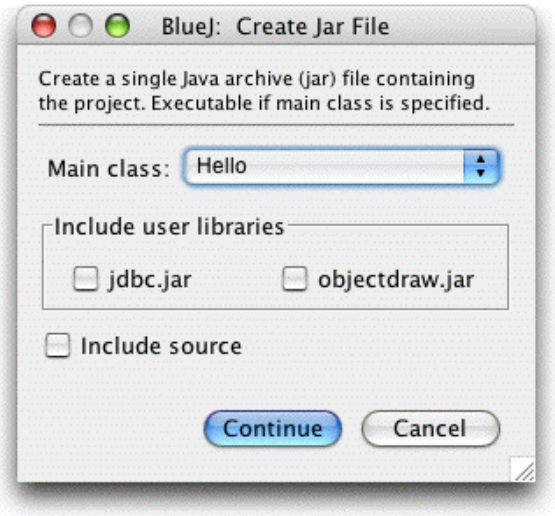

Рисунок 18: Диалог "Создать Jar- Файл"

В нашем примере выбор главного класса прост: есть только один класс. Выберите *Hello* из контекстного меню. Если у Вас – другой проект, выберите класс, содержащий метод `main`, который Вы хотите выполнить.

Обычно исходные тексты не включают в исполняемые файлы. Но Вы можете включить, если собираетесь передавать не только исполняемый файл, но и исходные тексты. (Вы можете использовать формат jar, чтобы переслать ваш проект по электронной почте в виде одного файла.)

Если Вы настроили BlueJ на использование пользовательских библиотек (через установку *Preferences/Libraries*, или используя каталог *lib/userlib*), Вы увидите в

середине окна область с заголовком *Include user libraries (Включить пользовательские библиотеки)*. (Если Вы не используете никаких библиотек, эта область будет отсутствовать). Следует отметить каждую библиотеку, которую использует Ваш текущий проект.

Щелкните *Continue*. После этого Вы увидите диалог выбора имени jar-файла. Введите `hello` и щелкните по *Create*.

Если не требуется включение библиотек, сразу будет создан файл *hello.jar*. Если требуется включать библиотеки, будет создан каталог с именем hello, и в этот каталог будет помещен файл *hello.jar*. Кроме того, каталог будет содержать все необходимые библиотеки. Ваш jar-файл будет искать нужные ему библиотеки в том каталоге, где он находится, поэтому при перемещении файлов позаботьтесь о том, чтобы они были вместе.

В приложении, использующем графический интерфейс пользователя (GUI), для запуска Вы можете дважды щелкнуть по jar-файлу. В нашем примере используется текстовый ввод - вывод, так что мы должны запустить этот файл из командной строки. Давайте попробуем выполнить jar-файл.

Откройте окно DOS или терминал. Перейдите в каталог, в котором Вы сохранили ваш jar-файл (Вы должны увидеть файл *hello.jar*). В предположении, что система Java у Вас правильно установлена, введите команду

```
java -jar hello.jar
```

и приложение будет выполнено.

9 Создание апплетов

9.1 Выполнение апплета

Кратко: Чтобы выполнить апплет, выберите Run Applet из контекстного меню апплета.

BlueJ позволяет создавать и выполнять апплеты так же, как и приложения. Мы включили апплеты в каталог примеров дистрибутива. Давайте попробуем выполнить апплет. Откройте проект *appletdemo* из каталога *examples*.

Вы увидите, что этот проект содержит только один класс с именем *CaseConverte*r. Значок класса отмечен (<<applet>>) как апплет. После компиляции, выберите *Run Applet* из контекстного меню класса.

Появится окно диалога для выбора некоторых параметров (Рисунок 19).

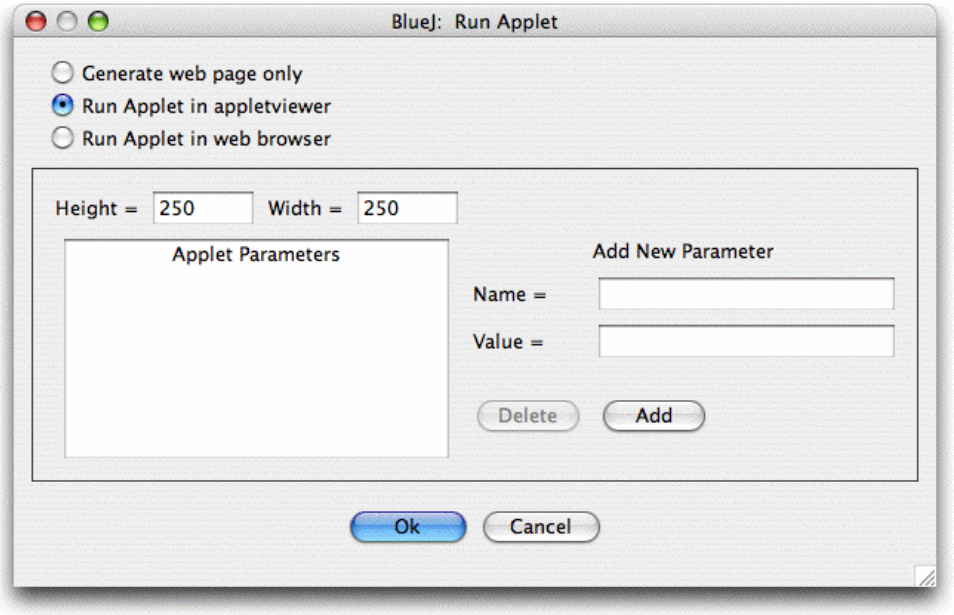

Рисунок 19: Окно диалога "Run Applet" («Выполнить апплет»)

Можно выбрать выполнение апплета в браузере или в appletviewer (в программе просмотра апплетов) или только генерировать web-страницу, не выполняя апплет. Примите установки по умолчанию и щелкните *OK*. Через несколько мгновений appletviewer отобразит наш апплет, позволяющий изменить регистр введенного текста.

Appletviewer устанавливается автоматически вместе с J2SE SDK (при установке Java), так что его версия соответствует версии компилятора Java. Appletviewer вообще причиняет меньше проблем, чем браузеры. Ваш web-браузер может выполнять другую версию Java (в зависимости от версии браузера), и это может создать проблемы. Однако с большинством современных браузеров все должно работать хорошо.

В системах Microsoft Windows и MacOS, BlueJ использует браузер, заданный по умолчанию. В системах Unix браузер задается в параметрах настройки BlueJ.

9.2 Создание апплета

Кратко: Чтобы создать апплет, щелкните по кнопке New Class *и выберите* Applet *как тип класса.*

Мы узнали, как выполнить готовый апплет, а теперь создадим свой апплет.

Создайте новый класс, выбрав *Applet* как тип класса (Тип можно выбрать в диалоге *New Class*). Скомпилируйте, а затем выполните апплет. Вот именно! Не слишком плохо, не так ли?

Апплет (как и другие классы) создан по имеющемуся шаблону класса, в котором содержится некоторый правильный код. Для апплетов этот код выводит две строки текста. Теперь можно открыть редактор и отредактировать апплет, вставив Ваш собственный код.

Вы увидите, что код апплета содержит все общие необходимые методы с комментариями, объясняющими их назначение. Пример кода - в методе *paint*.

9.3 Тестирование апплета

Иногда полезно создать объект апплета на стенде (как для обычного класса). Это легко сделать, т.к. в контекстном меню апплета есть конструктор. Когда объект апплета на стенде, нельзя выполнить апплет целиком, но можно вызвать его методы. Это удобно, когда требуется проверить отдельные методы, которые Вы, возможно, написали как часть реализации апплета.

Если Вы устанавливаете точки прерывания в апплете, они не будут работать, когда апплет выполняется в appletviewer или web-браузере. Причина в том, что и appletviewer, и web-браузеры используют их собственные виртуальные машины, которые ничего не знают о точках прерывания BlueJ.

Если Вам необходимо использовать точки прерывания и выполнение апплета в пошаговом режиме, Вам потребуется класс *AppletWindow*, который создал Michael Trigoboff. Класс формирует фрейм, который позволяет выполнять апплет непосредственно в BlueJ, так что можно будет пользоваться обычными методами отладки. Этот класс и пример можно найти в разделе *Resources* сайта BlueJ.

10 Другие операции

10.1 Открытие чужих пакетов в BlueJ

Кратко: Пакеты, созданные не в BlueJ можно открыть так: Project: Open Non BlueJ... .

BlueJ позволяет открывать пакеты, которые были созданы вне BlueJ – следует в меню выбрать Project – Open Non BlueJ ... . Выберите каталог, который содержит исходные файлы Java, затем щелкните по кнопке Open in BlueJ. Система запросит подтверждение того, что Вы хотите открыть каталог.

10.2 Добавление существующих классов к вашему проекту

Кратко: Классы могут быть добавлены в проект по команде Add Class from File...

Часто в Вашем проекте требуется использовать класс, текст которого Вы получили от кого-то. Например, преподаватель может дать студентам класс Java, который необходимо использовать в проекте. Такой класс легко добавить к проекту, выбрав из меню Edit - Add Class from File... , что позволит Вам указать для импорта исходный файл класса Java (с именем, заканчивающимся на *.java*).

После импортирования класса в проект, в каталоге текущего проекта создается копия файла (и класса). Эффект в точности такой же, как если бы Вы сами создали этот класс, написав его исходный текст.

Альтернатива рассмотренному действию – добавить нужный файл (и класс) в каталог проекта вручную. При следующем открытии проекта добавленный класс появится на диаграмме проекта.

10.3 Вызов метода main и других статических методов

Кратко: Статические методы можно вызвать из контекстного меню класса.

Откройте проект *hello* из каталога *examples*. Единственный класс в проекте (класс *Hello*) определяет стандартный главный метод – метод `main`.

Щелкните правой кнопкой мышки на классе, и Вы увидите, что меню класса включает не только конструктор, но также и статический метод *main*. Можно вызвать метод main непосредственно из этого меню[1] (без первоначального создания объекта).

Все статические методы можно вызвать подобным образом. Стандартный главный метод main допускает в качестве параметра массив строк – элементов типа String. Можно передать массив строк, используя стандарт синтаксиса Java для массивов-констант. Например, Вы могли бы передать методу main массив

{"one", "two", "three"}

(включая фигурные скобки). Попробуйте!

Замечание: В стандарте Java массивы-константы не могут использоваться как фактические параметры при вызове метода. Их можно применять только в качестве инициализаторов. В BlueJ, чтобы допустить интерактивный вызов стандартных методов main, мы разрешили передавать массивы-константы как параметры.

10.4 Создание документации

Кратко: Чтобы создать документацию по проекту, из меню Tools *следует выбрать* Project Documentation.

Можно создать документацию к Вашему проекту в стандартном для *javadoc* формате, не выходя из BlueJ - просто выберите в меню Tools - Project Documentation. Для всех классов проекта из исходного кода будет сгенерирована документация, и Вы увидите ее в открывшемся браузере.

Можно создать и просмотреть документацию по отдельному классу непосредственно в редакторе BlueJ. Для этого откройте редактор и воспользуйтесь всплывающим меню в правой верхней части окна редактора. Измените выбор с *Implementation* (реализация) на *Interface*, и Вы увидите в редакторе документацию по классу в стиле javadoc.

10.5 Работа с библиотеками

Кратко: Программный интерфейс стандартных классов Java можно увидеть, выбрав в меню Help - Java Class Libraries.

Бывает, что при написании программ Вам необходимо обратиться к стандартным библиотекам Java. Вы можете открыть web-браузер для просмотра доку-

[1] Статические методы класса принадлежат *классу*, а не объекту. Если создать объект, то в его контекстном меню не будет метода main.(Прим. перводчика)

ментации на программные интерфейсы JDK, выбрав из меню Help - Java Standard Classes (если у Вас есть подключение к Интернет).

Документация к JDK может также быть установлена и может использоваться локально (без подключения к сети). Подробности объясняются в разделе справки на сайте BlueJ.

10.6 Создание объектов из библиотечных классов

Кратко: Чтобы создать объекты библиотечных классов, выберите в меню Tools – Use Library Class.

BlueJ также предоставляет возможность создания объектов из классов, которые не являются частью вашего проекта, но определены в библиотеке. Вы можете, например, создать объекты класса String или класса ArrayList. Это весьма полезно для быстрых экспериментов с библиотечными объектами.

Библиотечный объект можно создать, выбирая в меню Tools – Use Library Class…. В появившемся окне диалога есть всплывающее меню, которое поможет Вам ввести полностью квалифицированное имя класса, вида *java.lang.String*. (заметьте, Вы должны задать полностью квалифицированное имя, включая имя пакета, содержащего класс.)

С полем ввода связано всплывающее меню, показывающее недавно использованные классы. Если имя класса введено, нажатие *Enter* отобразит список всех конструкторов и статических методов этого класса. Теперь может быть вызван любой конструктор или статический метод из этого списка.

При вызове происходит то же, что и при вызове любого другого конструктора или метода – появляется окно диалога, в котором Вас попросят задать фактические параметры, и, если не было ошибок, на стенде появится объект, или Вы увидите результат выполнения метода.

11 Краткий справочник

Начало

1. Чтобы открыть проект, выберите Open из меню Project.
2. Чтобы создать объект, выберите конструктор из контекстного меню класса.
3. Чтобы выполнить метод, выберите его из контекстного меню объекта.
4. Чтобы редактировать исходный код класса, дважды щелкните его значок класса.
5. Чтобы компилировать класс, щелкните по кнопке Compile в редакторе. Чтобы компилировать проект, щелкните по кнопке Compile в окне проекта.
6. Чтобы получить справку для сообщения об ошибках компилятора, щелкните по вопросительному знаку рядом с сообщением об ошибках.

Идем дальше...

7. Инспекция объекта позволяет увидеть внутреннее состояние объекта, что является одним из простых способов отладки.
8. Объект можно передать методу как параметр, щелкнув по пиктограмме объекта.

Создание нового проекта

9. Чтобы создать проект, выберите New ... из меню Project.
10. Чтобы создать класс, щелкните по кнопке New Class и задайте имя класса.
11. Чтобы создавать зависимость (изображаемую стрелкой), щелкните по кнопке стрелки и перетащите стрелку в диаграмму проекта, или только напишите исходный текст в редакторе.
12. Чтобы удалить класс или стрелку, выберите действие Remove из контекстного меню.

Использование панели кода

13. Для работы с панелью кода, выберите Show Code Pad из меню View.
14. Чтобы вычислить выражение, введите его в панель кода и нажмите Enter.
15. Для перемещения объекта с панели кода на стенд, перетащите маленькую пиктограмму объекта.
16. Чтобы инспектировать объекты, полученные как результат в панели кода, дважды щелкните по маленькому значку объекта.
17. Операторы, набранные в панели кода, выполняются.
18. shift-Enter в конце строки является признаком продолжения оператора (цепочки операторов) в следующую строку.
19. В одном многострочном операторе можно использовать локальные переменные. Имена объектов со стенда могут использоваться как поля объекта «Стенд».
20. Стрелка "вверх" и стрелка "вниз" перемещают по набранным и выполненным в панели кода строкам.

Отладка

21. Чтобы установить точку прерывания в строке, щелкните в области номеров строк и точек прерывания редактора - слева от текста строки.
22. Для пошагового выполнения кода используйте кнопки Step и Step into.
23. Контроль переменных прост – их значения автоматически отображаются в отладчике.
24. Кнопки Halt (Остановить) и Terminate (Закончить) используются, чтобы приостановить выполнение на время или прекратить выполнение.

Создание автономных приложений

25. Чтобы создать автономное приложение, выберите *Project - Create Jar File....*

Создание апплета

26. Чтобы выполнить апплет, выберите Run Applet из контекстного меню апплета.
27. Чтобы создать апплет, щелкните по кнопке *New Class* и выберите *Applet* как тип класса.

Другие операции

28. Не-BlueJ пакеты можно открыть так: *Project: Open Non BlueJ....*
29. Классы могут быть добавлены в проект по команде *Add Class from File...*
30. Статические методы можно вызвать из контекстного меню класса.
31. Чтобы создать документацию по проекту, из меню *Tools* следует выбрать *Project Documentation*.
32. Программный интерфейс стандартных классов Java можно увидеть, выбрав в меню *Help - Java Class Libraries.*
33. Чтобы создать объекты библиотечных классов, выберите в меню *Tools – Use Library Class*.