Министерство общего и профессионального образования
Российской Федерации
РОСТОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Ю.А. Кирютенко В.А. Савельев

Объектно-ориентированное программирование и язык Smalltalk.

Стандартные окна системы Smalltalk Express for Windows

Ростов-на-Дону
1999

Ю.А. КирютенкоВ.А. Савельев

Объектно-ориентированное программирование и язык Smalltalk.

Стандартные окна системы Smalltalk Express for Windows

Аннотация

Методическая разработка посвящена современному направлению в программировании — объектно-ориентированной методологии программирования и языку Smalltalk и является продолжением вышедших ранее методических разработок по объектно-ориентированному программированию и языку Smalltalk, посвященных общим концепциям и синтаксису языка, интерфейсу пользователя и среде программирования, классам Collection, Magnitude, Stream и их подклассам, протоколам поддержки всех объектов и всех классов системы, классам ядра графики системы как в Smalltalk for Dos так и в Smalltalk for Windows. В данной методической разработке описываются стандартные окна, их устройство и применение в системе Smalltalk for Windows фирмы ParkPlace/Digitalk (США).

Методическая разработка предназначена для студентов 3—5 курсов механико-математического факультета и слушателей ФПК.

Печатается в соответствии с решением кафедры математического анализа Ростовского государственного университета, протокол № 11 от 28 июня 1999 года.

Настоящие методические указания набраны в системе LaTeX с использованием кириллических шрифтов семейства «Литературная» (AOParaGraph) и математических шрифтов семейств Computer Modern и Math Symbol (American Mathematical Society).

1 Стандартные окна системы Smalltalk

Опишем те специальные окна, которые система Smalltalk предоставляет пользователю для разработки приложений, их сопровождения и отладки. От реализации к реализации видоизменяются сами окна, меняется число и расположение элементов в их меню, но основные принципы организации работы окна и операции, выполняемые по одноимённым опциям меню, во всех реализациях практически одинаковы. Поэтому, как образец, мы рассмотрим окна системы Smalltalk Express и рассмотрим следующие окна, которые обязательно есть во всех Smalltalk-системах:

Class Hierarchy Browser — окно просмотра Иерархии Классов. Показывает взаимосвязи классов внутри системы Smalltalk, позволяет редактировать любой доступный пользователю код из любого класса.

ClassBrowser — окно просмотра Класса. Подобно окну просмотра Иерархии Классов, предоставляет возможность редактировать методы, сконцентрированные в конкретном классе.

Inspector — окно инспектора. Позволяет исследовать структуру объектов и редактировать её. Эти окна полезны при отладке.

Walkback и Debugger — окно ОстановаСистемы и окно Отладчика. Предоставляют возможность просмотреть состояние программы в момент ошибки и являются средствами отладки на уровне системы Smalltalk.

MethodBrowsers — окно просмотра Методов. Предоставляет возможность просматривать и редактировать список методов.

MessageBrowsers — окно просмотра Сообщений. Удобный инструмент исследования селекторов, которые составляют метод, и определения перекрёстных ссылок для каждого селектора, что позволяет получать списки всех отправителей и разработчиков конкретного сообщения.

Все окна просмотра состоят по крайней мере из двух панелей: панели списка и текстовой панели. Выбор из списка отображает связанную с выбранным элементом списка информацию в текстовой панели. Этот текст можно изменять и в конечном счёте сохранять, тем самым изменяя систему.

1.1 Окно просмотра Иерархии Классов

Окно просмотра Иерархии Классов позволяет исследовать связи классов внутри системы Smalltalk и редактировать их содержимое.

1.1.1 Как открыть окно просмотра Иерархии Классов

Чтобы открыть окно просмотра Иерархии Классов, надо из меню File любого открытого окна системы (всегда, после старта системы, доступно окно системной информации Transcript) выбрать опцию Browse Classes (ПросмотрКлассов) или нажать Alt+B.

Окно просмотра Иерархии Классов разделено на пять панелей.

Список, отражающий текущее состояние иерархии классов — левая верхняя панель, в которой имена всех классов системы приводятся в иерархическом порядке. Класс Object (Объект) является первым в этом списке, поскольку он — базовый, самый верхний класс в иерархии. Все другие классы — подклассы класса Object и, следовательно, их имена отображаются в иерархии смещёнными вправо. Имена подклассов классов первого уровня тоже смещены вправо, и так далее.

Список переменных — средняя панель в верхней половине окна просмотра, расположенная ниже множества «радио-кнопок» instance/class (экземпляр/класс). Панель списка переменных показывает наследование переменных для выбранного класса по отношению к его суперклассам. Наследование переменных показывается в обратном порядке: сначала, на самом верху списка, переменные, определённые в выбранном классе. Суперкласс выбранного класса отображается его именем с приставкой и суффиксом в виде чёрточки, то есть в виде "- `SuperclassName` -". Затем перечисляются наследуемые переменные указанного суперкласса. Заканчивается такой список переменными, наследуемыми из класса Object. Переменные внутри каждого класса перечисляются в алфавитном порядке. Содержимое панели списка переменных зависит от выбранной «радио-кнопки»: если выбрана кнопка instance, отображаются переменные экземпляра выбранного класса, а если выбрана кнопка class — переменные класса.

Панель списка методов — панель справа от панели списка переменных. В ней отображается в соответствии с выбранной «радио-кнопкой» или список методов экземпляра или список методов класса. Если в панели списка переменных выбирается некоторая переменная, то список методов уменьшается до подмножества тех методов данного класса, которые ссылаются на выбранную переменную.

Непосредственно Smalltalk-код отображается в панели содержания — текстовой панели, которая занимает нижнюю половину окна. Когда в панели иерархии классов выбирается класс, в панели содержания отображается сообщение, которое определяет выбранный класс. Когда выбирается метод, в этой панели отображается код выбранного метода. Панель содержания обеспечивает возможности по редактированию и определению как самого класса, так и исходного текста его методов.

1.1.2 Просмотр иерархии

Чтобы выбрать класс для просмотра, надо переместить курсор в списковую панель иерархии классов, верхнюю левую панель окна, и щёлкнуть левой кнопкой мыши на имени класса. Smalltalk отобразит определение класса в панели содержания, а список переменных и методов, реализуемых классом, в панели переменных и панели списка методов, соответственно. Используя полосу прокрутки, можно просмотреть весь список классов в иерархии.

Меню Classes содержит функции, которые позволяют поддерживать иерархию, производя добавление и удаление подклассов, модифицировать панели иерархии после сделанных в иерархии изменений, открывать окно просмотра класса на выбранном классе, записывать определения класса и методов класса в файл. Это меню доступно и как меню панели, возникающее при нажатии в панели просмотра иерархии классов правой кнопки мыши. Опции (элементы) меню Classes следующие:

Add Subclass... (ДобавитьПодкласс...) — позволяет добавить подкласс к выбранному классу.

File Out... (ВывестиВФайл...) — записывает определение выбранного класса наряду со всеми его методами класса и экземпляра в файл. Файл создаётся в специальном формате файла регистрации изменений. Сначала появляется диалоговое окно, с предлагаемым системой по умолчанию именем файла, которое строиться по шаблону <укороченное имя класса>.cls. Класс будет сохранён в файле в текущем каталоге, если последний не будет изменён в процессе диалога. Подклассы выбранного класса в таком файле автоматически не сохраняются. Функция File Out... не воздействует на класс в самой системе. При выводе с помощью этого пункта меню классов из приложения со сложными взаимосвязями между классами, выведенные определения классов могут не загружаться обратно в систему. Для получения переносимой формы определения классов приложений лучше использовать специальные классы-утилиты, поставляемые как дополнения системы (каталог Extras).

Update (Модифицировать) — требует, чтобы окно просмотра иерархии классов повторно вычислило список классов в иерархии и отобразило его. Это надо делать всегда, если, пользуясь другими окнами просмотра, выполнялись операции удаления или добавления классов, иначе список в панели не будет отражать текущее состояние иерархии.

Browse (Просмотреть) — открывает отдельное окно просмотра на выбранном классе. Это может быть удобно, когда необходимо при просмотре одного

класса обратиться к другому.

Hide/Show (Скрыть/Показать) — переключатель, который позволяет скрывать или показывать подклассы выбранного класса. Это позволяет укорачивать список классов, скрывая те их них, которые в настоящее время не представляют интереса. Если подклассы скрыты, после имени класса в панели списка иерархии классов появляется многоточие (...). В такой ситуации в меню Classes прочитается опция Show Subclasses (ПоказатьПодклассы). Чтобы показать подклассы, можно выбрать эту функцию. То же самое последует, если просто дважды щёлкнуть на нужном классе. Если выбран класс, подклассы которого отображаются в панели, то в меню Classes прочитается опция Hide Subclasses (СкрытьПодклассы). Выберите её, или просто снова дважды щёлкните на классе, чтобы свернуть иерархию. Если класс не имеет подклассов, отображаемая блеклым шрифтом опция Hide/Show в меню Classes, в текущем состоянии не доступна.

FindClass... (НайтиКласс...) — вызывает простое диалоговое окно, позволяя ввести имя того класса, который надо отыскать в иерархии. Эта опция чрезвычайно полезна при поиске подклассов, которые содержаться в иерархии на уровнях, которые в настоящее время в панели не показаны.

Remove Class (УдалитьКласс) — удаляет выбранный класс из системы.

1.1.3 Добавление нового класса

Чтобы добавить подкласс в класс, сначала в списковой панели иерархии классов, надо выбрать класс, который будет суперклассом нового класса. Затем из меню Classes, надо выбрать опцию Add Subclass... (ДобавитьПодкласс...), которая выводит на экран диалоговое окно с именем "Add a SubClass", отображающее в поле с меткой Superclass: (Суперкласс:) имя выбранного класса, и предоставляющее поле редактирования New Class: (Новый Класс:), куда надо ввести имя нового класса.

Кроме того, в диалоговом окне находятся «радио-кнопки», которые позволяют выбрать тип создаваемого подкласса. Тип подкласса зависит от того, должны ли объекты, принадлежащие классу, содержать именованные переменные экземпляра, индексированные переменные экземпляра или массивы байт (если сказанное не совсем понятно, пусть это Вас пока не волнует). После того, как выбор сделан, сформируется новый подкласс, а список иерархии класса автоматически модифицируется.

Чтобы для класса определить переменные экземпляра, переменные класса и словари пула, сначала надо выбрать этот класс в панели списка иерархии клас-

сов. На это Smalltalk отреагирует отображением текущего определения класса в панели содержания. Определение включает в себя суперкласс класса, переменные экземпляра, переменные класса и словари пула. Определение класса изменяется, когда редактируется текст в панели содержания. Затем в меню File необходимо выбрать опцию Save (Сохранить) или нажать Alt + S. Определение класса модифицируется согласно сделанных изменений. Система автоматически перетранслирует все методы в классе и все его подклассы. Кроме того, в системный файл change.log запишется определяющее новый класс сообщение.

Напомним, что панель содержания, как всякая текстовая панель, имеет достаточно большое меню, которое вызывается при нажатии в панели правой кнопки мыши. Меню текстовой панели содержит в себе все элементы меню Edit и Smalltalk, а также опцию Save. Так что большинство операций можно делать не обращаясь к меню окна.

Изменение определения класса ведёт к немедленным последствиям, так что все будущие новые экземпляры класса будут иметь новую структуру. Надо быть очень осторожным при изменении тех классов, которые используются средой Smalltalk. Пока нет уверенности в том, что, все сделано корректно, следует определять подклассы основных классов, а не изменять структуру существующих.

1.1.4 Удаление экземпляров класса

Когда меняется определение класса или класс удаляется, а в системе все ещё остаются экземпляры этого класса, появляется сообщение об ошибке. Перед тем, как система позволит сделать и сохранить изменения, надо *удалить все экземпляры* модифицируемого класса.

Чтобы выявить все объекты, которые обращаются к объекту-приёмнику, можно послать объекту сообщение allReferences.

Для классов, связанных с окном, надо удостовериться, что в открытых окнах системы сохранена вся необходимая информация, и выполнить выражение Notifier reinitialize. При этом закроются все открытые окна и повторно инициализируется системное окно Transcript.

Если экземпляры не связаны с окном, или если некоторые из них все ещё остаются в системе,сначала сохраните образ системы, а затем пробуйте вычислить такой код:

```
MyClass allInstances do: [ :each | each become: String new].
```

Это выражение производит переназначение указателей с экземпляра удаляемого класса на объект String new.

1.1.5 Удаление класса

Чтобы удалить класс, надо сначала в панели списка иерархии выбрать тот класс, который будет удаляться. Затем из меню Classes выбрать опцию Remove Class (УдалитьКласс). Система попросит подтверждения на выполнение такой операции.

Когда класс удаляется, автоматически удаляются все методы данного класса. Smalltalk не допустит удаления класса, если он имеет подклассы или в среде есть его экземпляры. В этом случае при попытке удалить класс возникнет окно Walkback, объясняющее причину останова. Не забудьте, что Smalltalk автоматически удаляет (собирает как мусор) любой объект, на который нет ссылок со стороны других объектов системы.

Если в системе все же остаётся ссылка на удалённый класс, эта ссылка станет указывать на этот класс как на deletedClass (удаленныйКласс) и не будет удаляться сборщиком мусора. Это сделано в целях безопасности, хотя такие фиктивные ссылки и занимают место в образе.

1.1.6 Просмотр методов

Панель списка методов — верхняя правая панель окна. В классе есть два вида методов: методы экземпляра и методы класса. Список, который появляется в панели списка методов, определяется тем, какая из двух кнопок (экземпляр или класс) выбрана в панели, расположенной выше панели списка переменных.

На выделение селектора сообщения в панели списка методов, Smalltalk отреагирует тем, что разместит исходный текст метода, реализующего выбранное сообщение, в панели содержания. Всегда можно переместить курсор или прокрутить панель списка методов и выбрать другой метод.

Меню Methods содержит восемь функций:

New Method (Новый Метод) — используется для того, чтобы добавить новый метод экземпляра или класса к выбранному классу. Процедура добавления новых методов описана ниже.

Senders (Отправители) — заставляет Smalltalk искать все те методы среды, которые отправляют (вызывают) метод с выбранным селектором сообщения. Появляется окно отправителей, которое является окном просмотра Методов, и отображает каждый метод (и класс), который посылает сообщение с выбранным селектором сообщения.

Implementors (Реализаторы) — заставляет Smalltalk искать все те классы среды, которые реализуют (определяют) методы с выбранным селектором. Появляется окно реализаторов — окно просмотра Методов, оно отображает

имена всех классов, которые реализуют методы с выбранным селектором сообщения.

Local Senders (ЛокальныеОтправители) — во всем подобна опции Senders, за исключением того, что среда поиска отправителей ограничена текущим классом и его подклассами.

Local Implementors (ЛокальныеРеализаторы) — во всем подобна опции Implementors, за исключением того, что среда поиска реализаторов ограничена текущим классом и его подклассами.

Messages (Сообщения) — заставляет Smalltalk собрать в список все селекторы методов, вызываемых в выбранном методе. Открывается окно SelectorBrowser, отображающее этот список.

File Out... (ВывестиВФайл...) — заставляет Smalltalk послать выбранный метод в файл на диске, который создаётся в формате файла регистрации изменений.

Remove (Удалить) — удаляет из класса выбранный метод. Список методов немедленно модифицируется.

Меню Methods дублируется, как меню панели, и возникает при нажатии в панели методов правой кнопки мыши.

Меню Variables работает вместе с панелью списка переменных и помогает сузить круг методов, отображаемых в панели списка методов. В классах с большим числом методов, сужение списка может быть очень полезно при быстром поиске нужного метода или методов. Меню Variables дублируется, как меню панели, и возникает при нажатии в панели переменных правой кнопки мыши.

Меню Variable содержит три элемента:

Assigned (Назначение) — вызывает панель списка методов и отображает в ней только те методы, в которых назначается переменная, выбранная в списковой панели переменных.

Used (Использование) — вызывает панель списка методов и отображает в ней только те методы, в которых вызывается переменная, выбранная в списковой панели переменных.

Both (И то и другое) — выбор по умолчанию, вызывает панель списка методов и отображает в ней только те методы, в которых или назначается или используется переменная, выбранная в списковой панели переменных.

Способ сужения круга выбираемых методов обозначается в меню Variables «галочкой» (checkmark — маркером выбора), которая появляется рядом с одним из элементов меню. Это взаимно исключающий выбор, в любой момент может быть выбранным только один элемент. Выбор элемента из списка переменных можно отменить, если нажать или на «радио-кнопку» или на имя класса в панели иерархии классов.

1.1.7 Добавление, изменение и удаление метода

Чтобы увидеть исходный текст метода, надо переместить курсор в панель списка методов и выбрать соответствующий селектор сообщения. Панель содержания — текстовая панель, используется для добавления и изменения методов. Панель списка методов используется для удаления методов из выбранного класса. Все функции из меню панели содержания ведут себя точно так же, как и во всех текстовых панелях.

Чтобы добавить новый метод, сначала надо нажать соответствующую «радио-кнопку», определяя какой метод (экземпляра или класса) будет создаваться, а затем выбрать опцию New Method из меню Methods. В панели содержания появляется доступный для редактирования шаблон. В соответствии с шаблоном следует ввести правильно сформированный текст нового метода и использовать функцию Save из меню File, вызывая компилятор и устанавливая новый метод в систему.

Чтобы изменить существующий метод, сначала надо выбрать его в панели списка методов, отредактировать исходный текст в текстовой панели, а затем выбрать функцию Save.

Если с процессе трансляции добавляемого или изменяемого метода будет обнаружена синтаксическая ошибка, в соответствующем месте текста метода появляется сообщение, объясняющее причину ошибки. Сообщение об ошибке сразу выбирается (отображается в негативе). Чтобы удалить сообщение об ошибке, достаточно нажать клавишу Backspace, а затем заново отредактировать текст и снова использовать функцию Save.

Когда метод без синтаксических ошибок успешно откомпилируется, система Smalltalk автоматически установит его в класс и все будущие вызовы этого метода будут использовать его новую версию. Исходный текст нового или изменяемого метода будет записан в файл change.log.

Можно отредактировать текст любого существующего в классе метода. Имя нового метода принимается из первой строки текста. Но, если Вы изменяли имя метода и провели операцию сохранения, то будет создан новый метод с новым именем, а первоначальный метод останется без изменений. Это чрезвычайно полезно при создании внутри класса разновидностей метода.

Чтобы удалить существующий метод из некоторого класса, надо его выбрать

в панели списка методов, и выбрать опцию Remove (Удалить) из меню Methods. Список методов немедленно модифицируется, показывая, что выбранный метод удалён из класса.

1.2 Окно просмотра Класса

Окно просмотра Класса может быть открыто одним из двух способов. Можно или посылать сообщение edit любому классу, или выбрать опцию Browse (Просмотреть) из меню Classes окна просмотра Иерархии Классов.

Переместите курсор в любую текстовую панель. Напечатайте имя класса, который хотите просмотреть, сопровождая его селектором edit, и выберите это выражение. Выберите из меню Smalltalk опцию Do it. Откроется окно просмотра класса на указанном в выражении классе. Заголовок окна просмотра Класса идентифицирует просматриваемый класс. В этом окне можно просматривать, добавлять и изменять только методы выбранного класса.

Списковая панель, определяющая просматриваемый словарь — верхняя левая панель, содержит список, позволяющий сделать всего два выбора: class или instance (класс или экземпляр). Если выбирается class, то в панели списка методов, расположенной ниже первой панели, немедленно отображается список методов класса. Если выбирается instance, то отображается список методов экземпляра. Выбирая в этой панели метод, его исходный текст можно увидеть в панели содержания — самой большой панели, расположенной справа от списковых панелей, в которой можно редактировать, перетранслировать и добавлять новые методы.

Все функции всех меню этого окна аналогичны соответствующим элементам окна просмотра Иерархии Классов. Обратите внимание, что меню Methods окна Class Browser — подмножество меню Methods окна просмотра Иерархии Классов.

1.3 Инспекторы

Окно Inspector (Инспектор) используется как инструмент для исследования и изменения структуры любого объекта системы. Чтобы открыть инспектор на объекте, надо послать сообщение inspect этому объекту. Например, чтобы открыть инспектор на объекте Display pen (пере, связанном с экраном), надо вычислите выражение Display pen inspect. Можно поступить и проще: выбрать в любой текстовой панели имя объекта, который надо просмотреть (например, Display pen) и выбрать опцию InspectIt из меню Smalltalk. Это приведёт к тем же последствиям, что и вычисление приведённого выше выражения. Так можно выбирать и просматривать объекты почти из любого контекста.

Окно Инспектора имеет две панели. Левая списковая панель — панель списка переменных экземпляра. Правая текстовая панель — панель содержания, отображающая значение выбранной переменной экземпляра. Списковая панель переменных экземпляра показывает все переменные экземпляра осматриваемого объекта (учитывая механизм наследования). Первый элемент в списке — всегда имя **self**, которое ссылается на инспектируемый объект. Затем перечисляются именованные переменные экземпляра, если они есть. Если объект имеет индексированные переменные экземпляра, то они перечисляются последними, указанием числовых индексов.

Когда из списка выбирается одна из переменных, её текущее значение отображается в панели содержания переменной экземпляра. Если выбрать **self**, то отобразится текущее значение инспектируемого объекта. Если выбрать в меню **Inspect** опцию **Inspect** (или сделать двойной щелчок на переменной экземпляра), новое окно Инспектора открывается на выбранной переменной экземпляра.

Панель содержания переменной экземпляра — текстовая панель. Чтобы вызвать нужные функции, можно использовать соответствующие опции из меню **File**, **Edit**, **Smalltalk** или воспользоваться опциями из меню текстовой панели. Панель можно использовать и для вычисления любого желаемого выражения. Есть две очень важные особенности этой панели:

- Любое выражение, которое вычисляется, компилируется в окружении, определяемом осматриваемым объектом. Это означает, что в выражениях можно использовать имена всех переменных экземпляра.

- Если выбирается опция **Save**, все содержимое текстовой панели компилируется и вычисляется, а результат заменяет текущее значение выбранной переменной экземпляра. Если выбирается опция **Restore** (Восстановить) из меню **File**, то **Smalltalk** отобразит в текстовой панели текущее значение выбранной переменной экземпляра.

При инспектировании словарей, возникают некоторые особенности. Напомним, что словарь — объект, который содержит в качестве своих элементов ассоциативные пары, связывающие ключи со значениями. Точно так же как и обычные инспекторы, инспекторы словарей имеет те же две панели, однако, в панели списка переменных экземпляра перечисляются ключи словаря, а не имена переменных экземпляра и индексы. Когда в этой панели выбирается ключ, связанное с ним значение отображается в панели содержания. Чтобы увидеть все это, откройте инспектор на словаре **ColorConstants**, выполняя выражение **ColorConstants inspect**.

Обратите также внимание на то, что в списковой панели инспекторов словарей отсутствует **self**, а к строке меню окна добавляется меню **Dictionary** (Словарь). Меню **Dictionary** имеет три опции:

Add (Добавить) — позволяет добавить в словарь новый элемент. Система вызывает диалоговое окно (Подсказчик), спрашивая о новом ключе. Пока новый ключ не выбран, связанное с ним значение равно nil, но его можно изменить в панели содержания, а затем сохранить.

Remove (Удалить) — позволяет удалить из словаря ассоциативную пару с выбранным ключом.

Inspect (Осмотреть) — создаёт окно Инспектора на выбранном ключе.

1.4 Окно просмотра Методов

Окно просмотра Методов (Method Browser) позволяет просматривать и редактировать список методов. Есть четыре основанных на меню способа открыть окно просмотра Методов. Выбор опции Senders в меню любого окна, которое предлагает эту опцию (например, меню Methods окна просмотра Иерархии Классов), открывает окно просмотра Методов на списке всех методов системы, которые посылают выбранный селектор сообщения. Выбор опции Implementors в меню любого окна, которое предлагает эту опцию, откроет окно просмотра Методов на списке всех методов, которые реализуют выбранный селектор сообщения. Аналогично, но только на классе и его подклассах работают опции Local Senders и Local Implementors.

Окно просмотра Методов часто упоминается как окно отправителей или как окно реализаторов, в зависимости от отображаемой окном информации. Окно просмотра Методов имеет два панели: панель списка методов в верхней части и текстовую панель в нижней части. Панель списка методов отображает список методов, идентифицированных селектором сообщения и классом. Когда в списке выбирается метод, его исходный текст отображается в текстовой панели.

Меню Methods в строке меню окна отправителей или разработчиков имеет шесть опций, последние пять из которых повторяют опции из аналогичного меню окна просмотра Иерархии Классов. Но есть и новая опция:

Remove from List (Удалить из Списка) — обеспечивает удобный способ отбросить записи из панели списка методов, которые в данный момент не нужны; удаление записи не удаляет метод из класса.

Текстовая панель позволяет просматривать и редактировать исходный текст выбранного метода. При редактировании методов в этой панели, можно, как и в других окна, использовать соответствующие опции из меню File, Edit и Smalltalk. Когда отредактированный текст сохраняется, метод перетранслируется.

1.5 Окно просмотра Сообщений (Селекторов)

Окно просмотра Сообщений (Message (Selector) Browser) имеет три панели, позволяющие полностью идентифицировать сообщение для каждого селектора в методе, на котором было открыто окно просмотра.

Левая верхняя списковая панель отображает селекторы всех сообщений, представленных в методе, отображаемом в нижней текстовой панели окна просмотра. Кроме того, окно имеет ещё одну списковую панель (правую верхнюю), благодаря которой можно просмотреть список методов (отправителей или реализаторов) для селектора, выбранного в панели селекторов.

Окно просмотра Сообщений часто упоминается просто как окно Сообщений, так как первичная списковая панель отображает найденные в методе селекторы, которые часто называют сообщениями.

Если в панели списка селекторов выбрать селектор, то в тексте метода, отображаемом в нижней текстовой панели, выберется (выделится) полное сообщение, связанное с этим селектором. Это может оказаться чрезвычайно полезно при попытке идентифицировать сложные сообщения, в котором параметры размещаются вместе с компонентами селектора.

Если для любого выделенного в левой верхней панели сообщения найти его отправителей или реализаторов, то соответствующий список появиться в правой верхней панели, а выделение любой строки в ней приведёт к отображению в нижней текстовой панели текста выбранного метода.

Все расположенные здесь меню нам уже знакомы по другим окнам и имеет только знакомые нам элементы.

2 Окна, предназначенные для отладки

Даже если Вы «большой профессионал», вряд ли с первого раза сможете написать не содержащий ошибок Smalltalk-код. К счастью, интерактивный характер среды разработки системы Smalltalk позволяет достаточно просто организовать поиск и исправление ошибок.

Теоретически, основными инструментами для отладки являются окна уведомлений, окна-инспекторы и отладчик. Об окнах-инспекторах мы уже рассказывали. Остановимся на двух окнах, непосредственно предназначенных для отладки. Первым в случае обнаружения ошибки появляются на экране *окно уведомлений* с именем Walkback (ОстановСистемы), из которого открывается второе окно — Debugger (Отладчик). Окно Walkback появляется автоматически при вычислении Smalltalk-кода. Если для выяснения причины ошибки, информации, содержащейся в этом окне не достаточно, следует открыть окно Debugger, выбирая опцию Debug (Отладить) из меню Walkback окна Walkback или нажимая кнопку Debug,

размещённую в текстовой панели окна **Walkback**. Окно **Debugger** предоставляет больше информации о случившемся. Эти важные окна легко различимы, они определены с разным цветом фона панелей: красным для окна **Walkback** и жёлтым для окна **Debugger**.

Следует отметить, что в **Smalltalk**-коде встречаются три существенно разных вида ошибок. Первый вид ошибок состоит в том, что в **Smalltalk**-выражение включается синтаксически неправильный код. Компилятор (вызываемый всякий раз, когда вызывается команда **Save**), не позволит добавить в систему синтаксически неправильный код. Только знание синтаксиса языка вполне достаточно для устранения ошибок подобного вида, поэтому мы здесь не будем их рассматривать.

Ошибки второго вида возникают во время выполнения **Smalltalk**-кода. Другими словами, система находит нечто, что она не может или не желает выполнять. Именно в такой ситуации возникает окно уведомлений **Walkback**, сообщая, в чем состоит проблема. Именно этот вид ошибок мы и будем далее рассматривать.

Третий вид ошибок составляют такие ошибки, которые возникают при правильном коде, но неправильном проекте. Ваша программа прекрасно выполняется, но не делает того, что она по-вашему должны была бы делать. Это может произойти, например, потому, что Вы не поняли, как работает некоторый характерный механизм в существующем классе, или потому, что неправилен сам проект или реализация одного из ваших классов. Ниже мы увидим, как можно проследить за выполнением вашего кода и кода системы, и это позволяет отыскивать некоторые ошибки и такого рода.

Конечно, в реальной практике есть и четвёртый вид ошибок, которые предопределены поставляемой библиотекой классов. Хотя и крайне редко, но такие ошибки встречаются, особенно в новых версиях. Если такие ошибки вдруг возникнут, и Вы поняли описанные здесь основные идеи технологии поиска ошибок, они помогут отыскать даже их.

2.1 Окно **Walkback**

Окно Walkback вызывается и появляется на экране тогда, когда происходит хотя бы одно из следующих событий:

- некоторому объекту послано сообщение **halt** (при этом говорят, что установлена контрольная точка); например, **self halt**;
- нажаты клавиши Ctrl + Break;
- во время выполнения **Smalltalk**-кода виртуальной машиной обнаружена ошибка ОВ(например, переполнение стека);

- некоторому объекту послано сообщение **error:**, которое использует в качестве параметра строку, описывающую ошибку; например, **self error: 'key not found'**.

Появившееся окно уже содержит некоторую полезную информацию. Можно начинать поиск причины его возникновения. Но, сначала, небольшое «лирическое» отступление. Отладка в системе **Smalltalk** имеет много общего с отладкой в других языках программирования. Везде применяются одни и те же «общие правила». Чтобы о них не забыть, сначала напомним их.

Правило первое: *внимательно прочитайте сообщение об ошибке.*

Любая система может сообщить только непосредственную и поверхностную причину возникшей проблемы, а не её первопричину. И все же! Стоит посмотреть на то, что говорится в сообщении об ошибке. Очень соблазнительно как можно скорее начать её поиск в исполняемом коде! Но помните, что система пытается сообщать Вам *нечто*.

После того, как вы прочитали сообщение об ошибке, не торопитесь. По крайней мере, постарайтесь понять то, что сообщает система. Подумайте, в чем могла бы состоять причина. Отладка — одна из тех вещей, где весьма важно постараться сразу же определить причину. Если быть не очень внимательным, можно ходить вокруг да около, пробуя разные, не относящиеся к сути способы решения, и ни на шаг не приближаясь к самой проблеме.

Правило второе: *никогда ничего не предполагайте заранее.*

Только то, что вы «абсолютно» уверены , что «вот эта переменная» в конкретный момент времени имеет конкретное значение, или что управление было передано «именно в эту часть» исполняемого кода, ещё не означает, что все обязательно было именно так. Проверьте, чтобы убедиться. Как и многое другое, **Smalltalk**-среда позволяет сделает это достаточно просто.

Вернёмся к отладке в системе **Smalltalk**, и попытаемся вычислить в окне **Workspace** выражение **'Привет' at: 8**, в котором происходит обращение к восьмой литере строки **'Привет'**. Поскольку таковой в строке нет, это приведёт к появлению окна **Walkback**, с заголовком: '**8 is outside of collection...**' ('8 вне пределов набора...').

Заголовок окна **Walkback** пытается описать происшедшую ошибку. Поэтому в качестве заголовка окна уведомления Вы всегда будете получать "имя" возникшей ситуации: "Message not understood" ("Сообщение, не понято"), "Division by zero" ("Деление на нуль"), и т.д.. Кроме того в текстовой панели Вы увидите срез стека вызовов, который точно показывает, что происходило. Помните, что система **Smalltalk** не делает никаких различий между «системным кодом» и «вашим

кодом». Для неё они совершенно одинаковы. Это означает, что срез стека вызовов может состоять из смеси вашего кода и кода из библиотеки классов. Так что, не удивляйтесь, если не узнаете весь отображаемый код.

Итак, заголовок прочитан и все внимание — на текстовую панель окна! Она показывает все посланные, но незавершённые сообщения. Каждая строка в текстовой панели представляет одно посланное сообщение, при этом первым стоит сообщение, посланное самым последним. В каждой такой строке, сначала написано имя класса объекта, принявшего сообщение, а затем, после '>>', селектор самого сообщения. Если используемый по сообщению метод определён в суперклассе класса приёмника, имя класса, в котором определён метод, отображается в круглых скобках. Иногда строка будет иметь вид:

```
[ ] in ClassName >> methodName.
```

Это означает, что ошибка произошла во время вычисления блока, расположенного в методе **methodName** из класса **ClassName**.

В приведённом простом примере Вы встретились со срезом стека, в котором правильный кусок кода системы приводит к старту вашего кода, содержащего ошибку. Так случается, когда Вы в рабочем окне пользуетесь командой **do it** или **show it**.

Но часто встречается и такая ситуация, когда ваш код вызывает некоторый метод из библиотеки классов, а тот, в свою очередь, вызывает множество других методов из библиотеки классов, прежде, чем в одном из системных методов не возникнет исключительная ситуация. Это не означает, что нашлась ошибка в самой системе! Это, как правило, означает, что в вашем коде было сделано нечто такое, что привело к возникновению некоторой проблемы, которая не обнаруживала себя до тех пор, пока система не попыталась выполнить запрещённую в её собственном коде операцию (подобной доступу к ключу, которого нет в словаре).

Вы можете просматривать срез стека снизу вверх, следуя тем путём, которым система выполняла код. Если наверху стека находится код системы, переместитесь ниже (обратно по времени), чтобы найти место, где начинается ваш код. Но, как уже было сказано, ваш код мог сгенерировать ошибку намного раньше (или в стеке вызовов или вне него), и в счастливом неведении передать «плохой объект», который в конечном счёте (много позже) и вызывает ошибку. Если метод, который фактически приводит к ошибке, уже *возвратил объект*, его нет в срезе стека вызовов. В этом случае, чтобы найти ошибку, надо в некотором месте прервать исполнение вашего кода, а затем проследить за его дальнейшим выполнением «вручную». Как это сделать мы коротко рассмотрим в следующем разделе, когда будем изучать второе отладочное окно.

Информация, содержащаяся в окне **Walkback** рассмотрена, и теперь предстоит сделать одно из трёх:

- определить, в чем проблема, опираясь только на информацию, содержащейся в окне **Walkback**, сразу закрыть его и перейти к решению возникшей проблемы.

- продолжить выполнение кода из точки прерывания, если окно **Walkback** возникло либо в результате прерывания по комбинации клавиш Control + Break, либо потому, что было послано сообщение **halt**; в этих случаях с программой все в порядке, так что можно выбрать опцию **Resume** (Продолжить) из меню **Walkback**, после чего окно **Walkback** закроется и выполнение кода продолжится.

- прийти к выводу, что нужна более подробная информация о происшедшем, и воспользоваться окном **Debugger** для чего или выбрать опцию **Debug** (Отладить) из меню **Walkback**, или нажать кнопку **Debug**; окно **Walkback** закроется, а окно **Debuger** появится на экране.

2.2 Окно Debugger

Окно **Debugger** (Отладчик) расширяет возможности окна **Walkback** по просмотру возникших проблем и, что очень важно, позволяет управлять процессом выполнения кода. Окно имеет четыре панели и пять кнопок.

Верхняя левая списковая панель служит двум целям: представлению процесса останова и перечислению контрольных точек. Если нажата «радио-кнопка» **Walkback**, расположенная выше этой панели, верхняя левая списковая панель содержит тот же список, что и окно **Walkback**, вызвавшее данное окно **Debugger**. Когда в этой панели выбирается строка, другие панели окна отображают связанную с выделенной строкой отладочную информацию.

Если нажата «радио-кнопка» **Breakpoints** (КонтрольныеТочки), верхняя левая списковая панель состоит из строк, содержащих имя класса и селектор метода, для всех методов с контрольными точками. Когда в этой панели выбирается строка, панель, расположенная ниже, отображает исходный текст выбранного метода.

Панель в нижней части окна отображает исходный текст выбранного метода. Если выбрана «радио-кнопка» **Walkback**, в исходном тексте выбранного метода (**Walkback**-метода) выделяется (отображается в негативной форме) то сообщение, которое было послано, но выполнение которого не завершилось. Эта панель служит текстовым редактором, с помощью которого можно работать с кодом метода точно также, как в окне просмотра Иерархии Классов. Например, можно модифицировать метод, и выбрать опцию **Save** из меню **File**, сохраняя его в классе. Но будьте внимательны! Если в это же время на том же методе открыто другое

окно просмотра, и впоследствии Вы возвращаетесь и его используете, Вы снова возвращаете ошибку в ваш код! Это происходит потому, что окна просмотра автоматически себя не модифицируют, когда код, который они отражают, изменяется в другом месте.

Если выбранный Walkback-метод модифицируется, все строки верхней левой списковой панели, расположенные выше самого нижнего местонахождения выбранного из списка метода, отбрасываются, поскольку изменился метод из-за которого они попали в этот список.

Два панели наверху справа служат окном-инспектором для приёмника сообщения, аргументов и временных переменных выбранного метода. Панель имен переменных, левая из двух панелей окна-инспектора, содержит self, представляя приёмник, имена всех аргументов и временных переменных. Самая правая текстовая панель, отображает значение выбранной из левой панели переменной и позволяет редактировать её. Двойной щелчок на элементе в панели имен переменных откроет окно инспектора на выбранном объекте. То же можно сделать, выделяя имя переменной и выбирая единственную опцию Inspect из меню Inspect. Например, можно просмотреть переменные экземпляра приёмника сообщения, дважды щёлкая на self.

При редактировании кода и в панели значения переменных, и в панели, отображающей текст выбранного метода, конечно же можно использовать все опции из меню File, Edit и Smalltalk. Когда выбирается опция Save из меню File, значение выбранной переменной или текст метода изменяются.

Но иногда в отладчике нельзя увидеть фактическую причину проблемы, поскольку, как мы сказали, реальная причина находится в некотором методе, который был вызван, отработал и уже возвратил объект. Вот в этом случае Вы можете в вашем коде установить контрольные точки, а затем проследить за последующим выполнением кода, пытаясь отыскать ошибку. Для этого, надо только вставить в нужном месте вашей программы контрольную точку в виде выражения self halt. Это сообщение реализовано в классе Object, так что каждый объект системы понимает его, и делает одно и то же — открывает окно уведомлений Walkback.

Когда останов произошёл, Вы можете принять решение и о продолжении вычислений, но, более вероятно, раз уж здесь Вы поставили контрольную точку, Вы откроете окно отладчика. Ведь в нем Вы можете просматривать значения переменных, и посылать сообщения. А самое главное – Вы можете использовать расположенная ниже строки меню окна Debugger кнопки Hop (Сдвинуться), Skip (Прыгнуть) и Jump (Перейти), исследуя выполнение кода в пошаговом режиме. Их функции также доступны через одноимённые опции из меню Go. Работают кнопки следующим образом:

Hop — выполняет одно из двух: или посылает одно сообщение языка Smalltalk или производит одно назначение;

Skip — выполняет немного больше, чем Hop: выполняет все до посылки следующего сообщения или выполнения назначения в текущем методе или до следующей контрольной точки; обратите внимание, что Skip может выполнять отдельные выражения и в методах низшего уровня.

Jump — выполняет больше чем Skip: до следующей контрольной точки или до конца отлаживаемого выражения.

Однако, иногда такой способ отладки может оказаться достаточно утомительным, особенно если надо только узнать вызывается ли конкретный метод или узнать каково значение некоторой переменной. Для такой простой ситуации, система Smalltalk позволяет произвести печать в системном окне Transcript с помощью выражений, которые можно вставлять на время отладки в отлаживаемый код. Чтобы напечатать нужную информационную строку в окне Transcript, надо использовать выражения, подобные следующему:

```
Transcript show: 'Here is a string'; cr.
```

Обратите внимание на то, что таким образом в окно Transcript можно посылать только строки. Если надо посмотреть на другие объекты, им сначала надо послать сообщение printString. Например:

```
Transcript show: 'anArray is ', anArray printString; cr.
```

Можно придумать и другие способы контроля за происходящих во время выполнения кода событиями. Так что для отладки иногда можно обойтись и без отладчика. Но вернёмся к окну Debugger. Меню окна Debugger содержит следующие функции:

Resume (Продолжить) — как и в окне Walkback, позволяет продолжить выполнение кода после сообщения halt или прерывания Ctrl+Break, окно Debugger исчезает, и выполнение продолжится от следующей точки прерывания; окно не позволить продолжить вычисления, если был изменён метод из Walkback-списка.

Restart (Перезапустить) — если был выбран Walkback-метод, то окно Debugger исчезает, и выполнение кода повторится, начиная с выбранного метода, посредством посылки соответствующего сообщения, с возможно изменёнными Walkback-данными.

Senders (Отправители) — как в окне просмотра Иерархии Классов, инициализирует окно просмотра Методов, которое будет содержать все методы системы Smalltalk, которые посылают сообщение, соответствующее выбранному сообщению.

Implementors (Реализаторы) — как в окне просмотра Иерархии Классов, инициализирует окно просмотра Методов, которое будет содержать все те методы системы, которые реализуют метод с тем же самым селектором, что и выбранный.

Add Breakpoint (Добавить контрольную точку) — инициализирует диалоговое окно, которое запросит у пользователя имя класса и селектор метода, который необходимо добавить к списку контрольных точек.

Remove Breakpoint (Удалить контрольную точку) — инициализирует выбор контрольной точки, которая будет удалена пользователем из списка контрольных точек.

More Levels (БольшеУровней) — инициализирует, если возможно, дополнительные строки, которые будут включены в списковую панель Walkback.

3 Пример использования отладчика

3.1 Определение нового класса

Построим класс, специально допуская в определении ошибки, чтобы получить некоторый опыт использования отладчика Debugger.[1]

Первое, что надо сделать — создать новый класс, назовём его WordIndex, который позволит создать базу данных документов, опираясь на те слова, которые они содержат. Документы — текстовые ASCII-файлы, рассматриваются как ряд слов, содержащих алфавитно-цифровые литеры, разделяемые рядом не алфавитно-цифровых литер. Запрос к базе данных состоит из массива строк (экземпляров класса String), при этом каждая строка представляет слово. В ответ на запрос экземпляр класса WordIndex должен возвращать набор имен файлов, тексты которых содержат все указанные в запросе слова. Например, таким способом можно определить по характеристикам, хранящимся в компьютере отдела кадров, тех служащих, характеристики которых содержат слова «объектно-ориентированное программирование».

Каждый экземпляр класса WordIndex будет иметь переменные экземпляра documents и words:

documents (документы) — множество строк, каждая из которых представляет путь к файлу, содержащему тот документ, слова которого будут введены в экземпляр класса.

[1] Этот пример взят нами из учебника (каталог tutorial), поставляемого вместе с системой SmalltalkExpress.

words (слова) — словарь, каждый ключ которого содержит строку, представляющую слово, а каждое значение которого является множеством, содержащим пути ко всем тем документам из базы данных, которые содержат слово-ключ.

Итак, откройте окно просмотра Иерархии Классов и добавьте в систему новый класс, определение которого имеет следующий вид:

```
Object subclass: #WordIndex
    instanceVariableNames: 'documents words '
    classVariableNames: ' '
    poolDictionaries: ' '
```

Определите в классе WordIndex следующие шесть методов экземпляра.

```
addDocument: pathName
  "Добавить все слова из документа, описанного
   строкой pathName к словарю слов."
  | word wordStream |
  (documents includes: pathName)
          ifTrue: [self removeDocument: pathName].
  wordStream := File pathName: pathName.
  documents add: pathName.
  [(word := wordStream nextWord) == nil]
        whileFalse: [self addWord: word asLowerCase to: pathName].
  wordStream close

addWord: wordString for: pathName
  "Добавить wordString к словарю слов для
   документа, описанного именем пути pathName."
   (words at: wordString) add: pathName

initialize
  "Инициализировать новый пустой экземпляр класса WordIndex."
  documents := Set new.
  words := Dictionary new

locateDocuments: queryWords
  "Возвратить множество, содержащее имена путей тех документов,
   которые содержат все слова из массива queryWords. "
  | answer bag |
  bag := Bag new.
```

```
  answer := Set new.
  queryWords do: [:word |
      bag addAll: (documents at: word
                          ifAbsent: [ #() ]) ].
  bag asSet do: [:document |
      queryWords size = (bag occurrencesOf: document)
              ifTrue: [answer add: document]].
  ↑ answer asSortedCollection asArray

removeDocument: pathName
  "Удалите строку pathName, описывающую
   документ, из словаря words."
  words do: [ :docs | docs remove: pathName].
  self removeUnusedWords

removeUnusedWords
  "Удалите из словаря words все те слова,
   которые имеют пустой набор документов."
  | newWords |
  newWords := Dictionary new.
  words associationsDo: [ :anAssoc | anAssoc value isEmpty
                          ifFalse: [newWords add: anAssoc]].
  words := newWords
```

Чтобы добавить определение класса WordIndex и его методы в образ системы Smalltalk, можно воспользоваться готовым файлом, который поставляется вместе с системой, вычисляя выражение:

```
(File pathName: 'tutorial\wrdindx8.st') fileIn; close
```

После этого надо выбрать опцию Update (*Модифицировать*) из меню Classes окна просмотра Иерархии Классов для того, чтобы имя нового класса появилось в иерархии.

3.2 Отладка класса WordIndex

Давайте рассмотрим класс WordIndex в терминах сообщений, которые создают экземпляр класса WordIndex и делают запросы. Напомним, что преднамеренно приведённые тексты методов содержат некоторые ошибки.

Итак, предположим, что класс работает так, как надо и посмотрим, что он делает. Чтобы воспользоваться экземпляром класса, его сначала надо создать, для этого следует вычислить выражение:

```
MyIndex := WordIndex new initialize
```

Объект MyIndex создаётся как новая глобальная переменная, которая ссылается на новый экземпляр класса WordIndex. Метод initialize инициализирует все переменные экземпляра WordIndex; то есть переменная documents теперь содержат пустое множество, а переменная words содержат пустой словарь. При вычислении этого выражения окна Walkback не возникло.

Далее, добавим файлы `chapter.5` и `chapter.6`, поставляемые с системой, как документы экземпляра MyIndex. Это делает метод addDocument:, который создаёт файловый поток, чтобы просмотреть документ, неоднократно посылая сообщение nextWord файловому потоку, чтобы получить каждое слово, а затем использует метод addWord:for: для ввода каждой пары слово→документ в словарь words. Значит, чтобы добавить слова из файлов `chapter.5`, `chapter.6` в экземпляр, надо вычислить следующие два выражения:

```
MyIndex addDocument: 'tutorial\chapter.5'.
MyIndex addDocument: 'tutorial\chapter.6'.
```

Не тут то было! Возникает окно Walkback, значит есть какая-то ошибка. В данном случае метка окна Walkback информирует нас о том, что сообщение addWord:to: не было понято ('addWord:to:' not understood), в то время как верхняя строка в текстовой панели показывает WordIndex как класс, объект которого не понял сообщение. Как отмечалось ранее, можно сделать одно из трёх действий:

- определить, в чем проблема, опираясь только на информацию, содержащейся в окне Walkback.
- продолжать выполнение, если окно Walkback возникло или в результате прерывания по комбинации клавиш Control + Break, или потому, что было послано сообщение halt.
- получить более подробную информацию, из окна Debugger.

В этом случае, чтобы установить причину ошибки, достаточно информации и в окне Walkback. Посмотрите на код в классе WordIndex, используя окно просмотра Иерархии Классов. Там определён метод с именем addWord:for:, а в методе addDocument: послано сообщение addWord:to:, которое и не было понято. Использовано неправильное сообщение!

Чтобы исправить ошибку, закройте окно Walkback и, пользуясь окном просмотра Иерархии Классов, исправьте метод addDocument:, вставляя addWord:for: вместо addWord:to:.

Пробуем снова добавить файлы обучающей программы в Index, используя те же выражения. Все равно не получается! На этот раз возникает новое окно Walkback, заголовок которого сообщает: Key is missing (Ключ отсутствует). Так как

причина ошибки не столь очевидна, получим более подробную информацию, открывая окно Debugger, для чего нажмём на кнопку Debug, расположенную в текстовой панели окна Walkback.

Верхняя левая панель (списковая панель) окна Debugger повторяет информацию из окна Walkback и её можно использовать для того, чтобы выбирать в ней Walkback-строки, получая в других панелях окна связанную со строкой информацию.

Чтобы понять причину ошибки, сначала выберем в левой верхней панели строку WordIndex>>addDocument:. В нижней текстовой панели появиться текст метода addDocument: класса WordIndex, с выбранным текстом

```
self addWord: word asLowerCase for: pathName
```

Это означает, что данное сообщение было послано, но его вычисление не было завершено. Просматривая значения всех переменных из средней верхней панели убеждаемся в том, что здесь все в порядке. Теперь выберем в левой верхней панели строку WordIndex>>addWord:for:, в нижней текстовой панели появиться текст метода addWord:for: класса WordIndex с выбранным текстом words at: wordString. Со значениями переменных здесь тоже все в порядке.

Поднимемся в левой верхней панели ещё на одну строчку и выберем строку Dictionary>>at:. Текстовая панель в нижней части окна отобразит исходный текст метода at: из класса Dictionary:

```
at: aKey
  "Возвращает значение из пары key/value приёмника сообщения,
   ключ которой равняется aKey. Если такой ключ не найден,
   сообщает об ошибке."
  | answer |
  ↑ (answer := self lookUpKey: aKey) == nil
     ifTrue: [self errorAbsentKey]
     ifFalse: [answer value]
```

с выделенным текстом self errorAbsentKey, который является текущим выражением, вычисление которого в этом методе не было завершено к моменту останова.

Метод at: возвращает значение той пары key/value из словаря-приёмника, ключ которой равняется аргументу aKey. Если ключ отсутствует, вызывается метод errorAbse приводящий к инициализации окна Walkback.

В средней панели вверху найдём self, аргумент aKey и временную переменную answer. Выбрав self, видим в панели вверху справа, что его значение — пустой словарь. Теперь выберем аргумент aKey, его значение — строка 'tutorial', первое слово в файле. Таким образом, мы пытались производить поиск в пустом словаре self, с первым словом из файла в качестве ключа.

Возвратимся к строке WordIndex>>addWord:for:, выберем параметр wordString. Как и раньше, это строка 'tutorial'. Таким образом, мы обращались к словарю words с ключом, и не проверили, присутствует ли данный ключ в словаре! Исправим метод addWord:for: в нижней текстовой панели окна Debugger так, чтобы он выглядел следующим образом:

```
addWord: wordString for: pathName
  "Добавить wordString к словарю слов для
   документа, описанного именем пути pathName."
   (words includesKey: wordString)
         ifFalse: [words at: wordString put: Set new].
   (words at: wordString) add: pathname
```

Сохраним новую редакцию метода и посмотрим, что произошло в окне Debugger. Строки в списковой панели из окна Walkback, расположенные выше строки WordIndex>>addWord:for:, исчезли, поскольку был изменён метод, с которым они были связаны. А строка Word Index>>addWord:for: все ещё выделена. Выберем, наконец, опцию Restart из меню Debugger, выполнение возобновится, посылая выбранное сообщение снова. Поскольку ошибка исправлена, окно Debugger исчезнет, метод, наконец-то, выполнится, наполняя экземпляр класса WordIndex содержанием.

Чтобы послать запрос объекту MyIndex, надо воспользоваться сообщением locateDocuments: Каждый запрос должен возвращать массив строк, содержащий пути к тем документам, которые содержат все слова, содержащиеся в запросе.

Метод locateDocuments: сложнее остальных методов в классе. Он использует экземпляр класса Bag, чтобы в нем накапливать пути для всех файлов, которые содержат каждое слово из запроса. (Не забывайте, что экземпляры класса Bag, в отличие от множеств, могут содержать дубликаты объектов.) Затем, экземпляр класса Bag исследуется с целью отыскать в нем все те документы, которые повторяются столько раз, сколько слов в запросе; это именно те документы, которые содержат все слова.

Итак, пробуем сделать запрос, вычисляя выражение:

```
MyIndex locateDocuments: #('show' 'class')
```

Снова возникает окно Walkback, информируя о том, что сообщение at:ifAbsent:, посланное экземпляру класса Set, не было им понято. Откроем окно Debugger, выберем в левой верхней панели строку Set(Object)>>doesNotUnderstand:, а затем выберем self из списка временных переменных, его значение:

```
Set('tutorial\chapter.6' 'tutorial\chapter.5')
```

Теперь выберем строку [ ] WordIndex>>locateDocuments: (третью строку сверху). Просмотрим на исходный текст метода, в котором внутри блока выделено незавершённое сообщение at:ifAbsent:, и исследуем значения временных переменных. В сообщении в качестве приёмника используется переменная экземпляра documents — экземпляр класса Set. Значение, распечатанное выше, подтверждает это, поскольку содержит пути к документам. Следовательно, или мы послали неправильное сообщение объекту documents, или documents — неправильный приёмник сообщения. Но код

```
bag addAll: (documents at: word ifAbsent: [#()])
```

пытается добавить в переменную bag все документы, которые включают строку, содержащуюся в переменной word. Но такая информация содержится в переменной words! Значит, неправильным является приёмник сообщения: надо использовать словарь words:

```
bag addAll: (words at: word ifAbsent: [#()])
```

Используя текстовую панель окно Debugger, изменим метод locateDocuments:, сохраним изменённый метод, и запустим его снова. Работает!

3.3 Как работают Hop, Skip и Jump

Теперь, когда мы отладили класс WordIndex, давайте посмотрим, как можно использовать отладчик Debugger для того, чтобы понять как функционирует приложение, непосредственно наблюдения за тем, как посылаются сообщения. Откроем окно Walkback, а из него окно Debugger, вычисляя вместе следующие выражения:

```
self halt.
MyIndex locateDocuments: #('each' 'talk').
```

Нажатие кнопок Hop, Skip и Jump, как ранее отмечалось, вызывает ограниченное выполнение кода.

Пробуйте дважды нажать кнопку Hop и понаблюдайте за изменениями окна Debugger. Первое нажатие выделит выражение MyIndex locateDocuments: #('each' 'talk'). После второго, это выражение начнёт пошагово выполняться, и выделенным окажется первое выражение метода locateDocuments:, то есть Bag new. Нажмите Hop снова. Вычисление следующего шага перейдёт в метод new класса Bag и будет продолжено со следующим утверждением, которое окажется высвеченным после сделанного шага. Обращаясь с панелям переменных, можно исследовать состояние объектов после каждого шага Hop.

Теперь нажмите несколько раз кнопку Skip. Обратите внимание, что выделяемый код является частью того же самого метода до тех пор, пока выполнение метода не закончится. Это позволяет сконцентрировать внимание на пошаговом выполнении одного метода и проигнорировать все сообщения низшего уровня.

Jump выполняет наиболее «длинные» шаги из всех трёх пошаговых кнопок. Если не устанавливались контрольные точки, Jump вычисляет все до конца отлаживаемого выражения. Следовательно, нажимая после нескольких нажатий Hop и Skip, один раз Jump, мы завершим вычисление выражения

```
MyIndex locateDocuments: #('each' 'talk').
```

Список литературы

[1] Goldberg A., Robson D., Smalltalk-80. *The language*. — Addison-Wesley Publishing Company, 1988.

[2] Levis S., *The Art and Science of Smalltalk*. — Prentice Hall, 1996.

[3] *Смолток. Объектно-ориентированная система программирования. Руководство пользователя, часть 1, 2, 3* — М.: Ин-т проблем информатики РАН, 1995

[4] Иванов А., Кремер Ю., *Язык* Smalltalk*: концепция объектно-ориентированного программирования* // КомпьютерПресс — 1992, № 4 — С. 21–31

[5] Фути К., Судзуки Н., *Языки программирования и схемотехника СБИС*: пер. с япон. — М.: «Мир», 1988

[6] Кирютенко Ю.А., Савельев В.А., *Объектно-ориентированное программирование и язык* Smalltalk. *Общие концепции и синтаксис*. — Ростов-на-Дону: УПЛ РГУ, 1995

[7] Кирютенко Ю.А., Савельев В.А., *Объектно-ориентированное программирование и язык* Smalltalk. *Класс* Collection *и его подклассы*. Часть 1, 2. — Ростов-на-Дону: УПЛ РГУ, 1996

[8] Кирютенко Ю.А., Савельев В.А., *Объектно-ориентированное программирование и язык* Smalltalk. *Класс* Magnitude *и его подклассы*. — Ростов-на-Дону: УПЛ РГУ, 1997

[9] Кирютенко Ю.А., Савельев В.А., *Объектно-ориентированное программирование и язык* Smalltalk. *Протокол поддержки всех объектов системы*. — Ростов-на-Дону: УПЛ РГУ, 1997

[10] Кирютенко Ю.А., Савельев В.А., *Объектно-ориентированное программирование и язык* Smalltalk. *Протокол поддержки классов*. — Ростов-на-Дону: УПЛ РГУ, 1997

[11] Кирютенко Ю.А., Савельев В.А., *Объектно-ориентированное программирование и язык* Smalltalk. *Ядро графики: классы* Point, Rectangle, Form. — Ростов-на-Дону: УПЛ РГУ, 1997

[12] Кирютенко Ю.А., Савельев В.А., *Объектно-ориентированное программирование и язык* Smalltalk. *Графика в* Smalltalk/*V for Windows*. — Ростов-на-Дону: УПЛ РГУ, 1998

Содержание