

Грачёв Дмитрий Александрович

*Астраханский государственный технический университет
магистрант кафедры Автоматизированные системы
обработки информации и управления
(8512) 614-109, dagrachev@list.ru*

Лаптев Валерий Викторович

*Астраханский государственный технический университет
доцент кафедры Автоматизированные системы
обработки информации и управления, к.т.н.
(8512) 614-109, laptev@ilabsltd.com*

СЕМАНТИЧЕСКАЯ ИНТЕГРИРОВАННАЯ СРЕДА ДЛЯ ОБУЧЕНИЯ ПРОГРАММИРОВАНИЮ

SEMANTIC INTEGRATED DEVELOPMENT ENVIRONMENT FOR TRAINING IN PROGRAMMING

Аннотация. Описывается интегрированная среда Semantic IDE для обучения программированию. Редактор кода не является текстовым редактором, а оперирует конструкциями языка и объектами программы. Редактор позволяет представить программу в разных синтаксисах. Описывается работа с проектами, и показаны возможности редактора для подготовки обучающих материалов непосредственно в среде.

Ключевые слова: интегрированная среда, Semantic IDE, семантический редактор, учебный язык программирования, семантическое дерево программы, синтаксис как интерфейс.

Abstract. Integrated development environment Semantic IDE is described in this article. The editor is not a text one and operates with language constructs and program objects. The editor allows to present the program in different syntaxes. Work with projects is described, and the editor abilities for educational contents creation in the environment are shown.

Keywords: integrated development environment, Semantic IDE, semantic editor, educational programming language, program semantic tree, syntax as interface

Введение

В работе [3] сформулированы требования к учебному языку и обучающей системе по программированию. В работе [1] описан разработанный авторами в соответствии с требованиями учебный язык программирования Semantic Language и его интерпретатор. Разработка программ на учебном языке должна осуществляться в интегрированной среде программирования (Integrated Development Environment, IDE). С одной стороны, эта среда является частью обучающей системы и должна обеспечивать процесс обучения. С другой стороны, интегрированная среда должна быть похожа на современную интегрированную среду для профессиональной разработки программ. Поэтому рассмотрим некоторые свойства, присущие современным интегрированным средам.

Интегрированная среда объединяет в единую систему множество системных программ: редактор, компилятор (и/или интерпретатор), компоновщик, отладчик, профайлер, мейкер и другие. Наиболее важной частью любой интегрированной среды является редактор кода, который и обеспечивает взаимодействие пользователя-программиста со средой.

Как правило, редактор кода представляет собой обычный текстовый редактор, в котором основные операции выполняются с символами, строками и блоками текста. Однако современные промышленные среды обеспечивают некоторую поддержку языка разработки, например:

1. Осуществляется цветовая подсветка синтаксиса – ключевых слов и объектов программы; подсветка обычно настраивается в самом редакторе.
2. Обеспечивается возможность закомментировать выделенный блок строк, и отменить комментарий.
3. Предоставляется возможность сворачивать и разворачивать блоки кода: классы, модули, условные операторы и операторы цикла, определения подпрограмм, пространства имен, регионы и т.п.
4. Проверяется парность разнообразных скобок, присутствующих в коде.
5. Предоставляется набор так называемых сниппетов (snippets), обеспечивающих вставку конструкций языка по короткой комбинации клавиш или даже по одной горячей клавише; сниппеты тоже разрешается настраивать в редакторе.

В последние годы в некоторых средах (например, в Visual Studio.NET) выполняется интеллектуальный анализ кода (технология IntelliSense) уже при его создании (до компиляции) [4]. Подобный механизм обеспечивает более богатые возможности:

1. При наборе кода предоставляются интеллектуальные подсказки, позволяющие не набирать конструкцию (например, вызов метода класса стандартной библиотеки) вручную, а выбрать нужную из списка, который появляется при наборе нескольких первых символов.
2. Обеспечивается возможность получения краткой справки об объектах программы (например, информация о составе методов класса или о списке параметров метода) непосредственно в редакторе без обращения к системе помощи.
3. Обеспечивается интеллектуальная навигация по коду, например, переход от вызова метода к его определению и обратно.
4. Ошибочные конструкции помечаются в окне редактора в момент набора (без компиляции), и в отдельном окне выводятся сообщения о возможных ошибках.

В наиболее развитых средах обеспечивается поддержка некоторых простых операций рефакторинга [2,4].

Результаты этих операций отражаются в окне редактора. Подобные возможности позволяют значительно снизить количество ошибок при наборе кода, и существенно повышают производительность программиста. Тем не менее, подавляющее большинство операций, которые программист выполняет в редакторе, – это обычные операции с текстом. Применение текстовых операций к коду программы приводит к тому, что регулярно нарушается синтаксис языковых конструкций. Это нередко приводит к возникновению мелких синтаксических ошибок, и в итоге – к непроизводительным затратам времени.

Как правило, редактор интегрированной среды сохраняет код программы в текстовом виде. Текстовое представление кода является входным для компилятора,

что приводит к необходимости иметь в компиляторе фазы лексического и синтаксического анализа. С одной стороны, это замедляет процесс разработки программы. С другой стороны, ошибки компиляции затрудняют процесс обучения, так как отвлекают обучаемого от главной цели – от решения задачи.

Кроме того, текст программы можно открыть, просмотреть и изменить вне интегрированной среды. При профессиональной разработке это бывает полезно, но при обучении провоцирует обучаемых на нечестные способы выполнения заданий по программированию.

Концепции разработки

Б. Страуструп в своей книге [6] отмечал, что главным препятствием на пути развития языка C++ являются символьно-ориентированные инструменты (в частности, текстовый редактор кода). Наиболее перспективный и интересный подход – отказаться от традиционного текстового представления и реализовать инструментарий на основе семантических понятий языка программирования. В этом случае синтаксис языка представляет собой интерфейс между языком программирования и пользователем (программистом). И как всякий интерфейс, его можно заменять, не изменяя базовой семантики языка.

В соответствии этим подходом и с учетом анализа свойств современных сред авторами был сформулирован ряд концепций реализации обучающей интегрированной среды, которая является одной из подсистем автоматизированной обучающей среды по программированию:

- среда должна поддерживать работу и с одномодульными, и с многомодульными программами;
- среда должна обеспечивать простой и независимый от платформы механизм накопления программных компонент; в частности, с помощью этого механизма должны быть разработаны и включены в среду модули стандартной библиотеки;
- среда должна позволять набор кода программы и в русской, и в английской лексике;
- среда должна обеспечивать возможность переключения лексики и синтаксиса языка программирования, причем это переключение не должно приводить к повторному анализу кода программы;
- среда должна обеспечивать механизм добавления нового синтаксиса;
- изменение ключевых слов в коде должно быть невозможно;
- ошибки должны определяться в момент набора программы;
- редактор кода должен оперировать конструкциями языка программирования и объектами программы;
- редактор должен обеспечивать при необходимости традиционные операции редактирования текста;
- ввод-вывод данных должен осуществляться в рамках среды без выхода в операционную систему.
- среда должна обеспечивать разработку системы помощи непосредственно в редакторе без привлечения сторонних средств.

В настоящее время практически все эти концепции реализованы в рамках интегрированной среды Semantic IDE.

Среда Semantic IDE и семантический редактор

Внешний вид среды Semantic IDE показан на рис. 1. Центральное окно – это окно редактора кода. В нем пользователь набирает код программ и пишет текст до-

кументов. Для каждого документа создается отдельная вкладка. Над центральным окном расположены главное меню и лента выбранного пункта главного меню.

Справа расположено окно проектов. В нем отображается текущий проект, с которым работает пользователь в данный момент. Внизу – окно сообщений об ошибках, и окно консоли. Сообщения об ошибках появляются при редактировании кода – для этого не требуется запускать программу на трансляцию. При выполнении программы среда переключается в окно консоли, в котором программист задает входные данные и в которое выводятся результаты работы программы. При работе в среде не открывается никаких дополнительных окон от операционной системы.

Программисту часто приходится выполнять поиск различных имен в проекте. Например, требуется найти все вызовы конкретной функции, или от вызова функции перейти к определению, чтобы уточнить список параметров и их типы. Для выполнения подобных операций предназначена отдельная вкладка окна консоли, куда выводятся результаты поиска.

Среда Semantic IDE предназначена в первую очередь для обучения начинающих программистов. Поэтому среда должна отслеживать все действия пользователя, собирать и обрабатывать статистику по операциям, совершаемым пользователем. Вывод действий пользователя осуществляется во вкладке Команды кодового окна редактора.

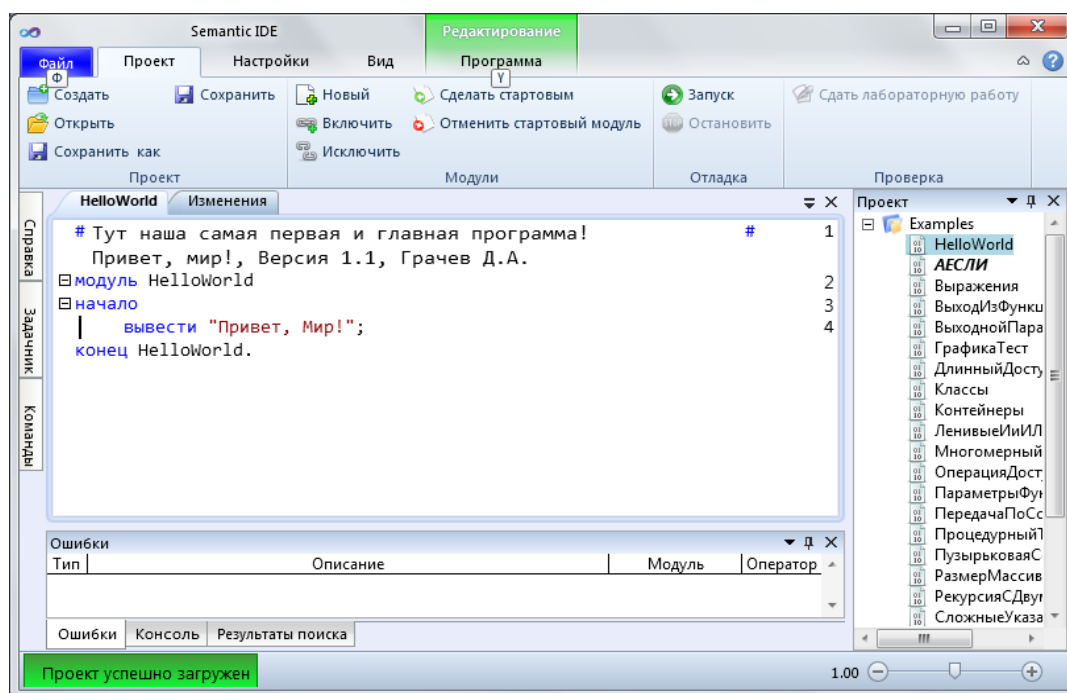


Рис. 1. Внешний вид Semantic IDE

Все окна, кроме центрального окна редактора, являются плавающими и их можно перемещать и закреплять как вкладки в центральном окне.

Проекты. Работа в интегрированной среде начинается с создания проекта. Проект – это набор файлов, созданных в редакторе кода и сохраненных на диск. При

создании проекта на диске создается папка, в которую записывается файл проекта. В этой же папке хранятся все остальные файлы, включенные в проект. Проект создается даже в том случае, если включает единственный файл.

Любой файл, созданный в редакторе кода, может содержать программный модуль на учебном языке. Но в общем случае это не обязательно – файл может содержать только неисполняемую информацию. Таким образом, в редакторе кода можно создавать как программные, так и информационные проекты. Примером информационных проектов в Semantic IDE являются проект справочной системы и проект Задачник (см. рис.1 – вкладки слева), содержащий набор лабораторных работ по программированию.

Программные проекты могут быть исполняемыми и неисполняемыми. Примером неисполняемого проекта в среде является проект Framework, в котором собраны модули стандартной библиотеки. Если же программный проект требуется выполнять, то один из модулей проекта назначается стартовым: выполнение программы начнется с секции инициализации этого модуля. Остальные модули загружаются и связываются по мере необходимости. Примером исполняемого проекта, предоставляемого в составе Semantic IDE, является проект Example, в который включено множество примеров программ на учебном языке (см. рис. 1).

Для хранения проектов на диске был разработан специальный xml-формат. Например, текущая версия проекта Задачник выглядит следующим образом:

```
<?xml version="1.0" encoding="utf-16"?>
<Project Name="Задачник" StartupModule="#notset">
  <Files>
    <Module Path="Лабораторная_1.sl" />
    <Module Path="Лабораторная_2.sl" />
    ...
    <Module Path="Лабораторная_10.sl" />
    <Module Path="Лабораторная_11.sl" />
  </Files>
</Project>
```

Файл проекта сохраняется в папке проекта в виде xml-файла с расширением «prj», а файлы, созданные в редакторе и включенные в проект, имеют расширение «sl». В данном случае отсутствует стартовый модуль, поскольку проект не является исполняемой программой.

Редактирование кода. Редактор кода – это не традиционный текстовый редактор. В Semantic IDE реализован семантический редактор. Как было описано в работе [3], семантический редактор оперирует не символами, а операторами учебного языка и объектами программы. Поэтому, во-первых, большинство элементов оператора сразу вставляются в код в правильном виде, и во-вторых, полностью исчезают ошибки набора ключевых слов. Редактор разрешает символьный ввод только в строго определенных позициях оператора. Например, в операторе объявления переменной разрешено вводить посимвольно только имя переменной.

Операторы учебного языка добавляются в код программы с помощью сниппетов и контекстного подсказчика. Каждый оператор учебного языка начинается ключевым словом, поэтому сниппет – это первые две-три буквы ключевого слова оператора. При наборе букв сниппета в кодовом окне редактора подсказчик выводит список операторов, и подсветка устанавливается на конкретном операторе.

Контекстный подсказчик работает и при наборе составного имени при вводе селектора-точки. Если перед точкой было набрано имя модуля, то в списке выводится список всех открытых объектов этого модуля. Если перед точкой было набрано имя объекта определяемого типа, то подсказчик выводит имена всех открытых полей и методов, определенных в этом типе.

Редактор следит за действиями программиста и сообщает об ошибках в момент набора программы. Пока ввод оператора не завершен, в окне ошибок «вывешены» все сообщения об ошибках, которые возникают по мере ввода составных частей оператора. Например, при ошибке в написании имени переменной во время ввода оператора мгновенно появляется сообщение о том, что данная переменная не была определена. Отметим, что в кодовом окне нумеруются операторы программы, а не строки текста.

Аналогично выполняется удаление – удаляется вся конструкция целиком. Если же оператор не удаляется, то разрешается замена элементов оператора. При замене тоже работает контекстный подсказчик. Например, в операторе объявления переменной можно заменить тип переменной, выбрав его из списка предложенных типов. Точно так же предлагается список видимых в данной точке имен переменных, если программисту потребовалось заменить имя переменной.

Заметим, что программа при вводе/удалении операторов всегда является синтаксически правильной – синтаксические ошибки отсутствуют. Даже при наборе арифметических или логических выражений всегда вставляется (и удаляется) синтаксически правильная часть выражения. Все возникающие ошибки являются исключительно лексическими (например, неверно набранная числовая константа) или семантическими (например, в операторе присваивания требуется недопустимое преобразование типа). Таким образом, набор кода программы требует минимального количества действий от программиста, и при этом существенно сокращается количество ошибок.

Поскольку традиционные текстовые операции с кодом отсутствуют, редактор обеспечивает набор высокоуровневых семантических операций модификации, которые не нарушают синтаксической корректности кода. Примером подобной операции может служить вставка объемлющего оператора цикла, преобразующая выделенную последовательность операторов в тело этого цикла. Как обычно, доступные операции можно выбирать из списка, предоставляемого контекстным подсказчиком.

Оператор-комментарий. Одним из операторов, размещаемых в коде, является оператор-комментарий, обозначаемый в окне редактора символом «решетка» (#). Оператор-комментарий может быть вставлен в код в любом месте, где допускается вставка оператора языка программирования, а также перед кодом и после него. Однако в семантическом редакторе оператор-комментарий трактуется более широко, чем обычные комментарии в программах: разрешается создавать произвольную последовательность операторов-комментариев вообще без программного кода.

В пределах комментария разрешается посимвольный ввод любого текста, и выполняются традиционные текстовые операции с символами и строками. Разрешены и обычные операции с буфером обмена – это позволяет вставлять фрагменты материалов из других программ (например, из MS Word). В операторе-комментарии можно создать таблицы, разрешается вставлять рисунки и видеоролики. Обеспечивается возможность связывания комментариев с помощью гиперссылок, в том числе и с другими файлами проекта.

Таким образом, возможности представления информации практически не уступают возможностям подготовки документа в MS Word в rtf-формате. Все это поз-

воляет преподавателю непосредственно в редакторе готовить обучающие материалы. Проект справочной системы и проект Задачник подготовлены именно таким образом. Кроме того, теоретический материал можно проиллюстрировать кодом программы-примера, которую можно запустить на выполнение. Это позволяет продемонстрировать образцы хорошего стиля программирования и способствует более быстрому и прочному усвоению изучаемой темы.

Синтаксис как интерфейс

Редактор кода в соответствии с действиями программиста строит внутреннее представление программы. Один модуль – это дерево с вершинами сложной структуры, которое мы назвали семантическим. Каждая вершина (узел) дерева представляет отдельный оператор программы, в котором сохраняется полная информация о семантике оператора. Например, для оператора присваивания хранится информация о вычисляемом выражении, и о переменной, которой присваивается значение выражения.

В каждом узле имеется ссылка на следующий узел-оператор. Таким образом, последовательность операторов представляет собой последовательный список узлов семантического дерева. В узлах, соответствующих блочным операторам учебного языка [1], имеется еще одна ссылка – на первый вложенный оператор тела.

Внутреннее представление программы в виде семантического дерева является входным для интерпретатора. Поскольку все ошибки выявляются при создании кода в редакторе, на вход интерпретатора поступает правильная программа. Интерпретатор не выполняет ни лексического, ни синтаксического анализа – это существенным образом упростило реализацию, и повысило быстродействие.

С другой стороны, представление программы в виде семантического дерева позволяет реализовать идею Б.Страуструпа о том, что синтаксис языка программирования является только интерфейсом [6], который, вообще говоря, можно менять, выбирая наиболее удобный. В Semantic IDE помимо синтаксиса Semantic Language реализованы Си-подобный и Pascal-подобный синтаксисы. Си-подобный синтаксис разработан на основе языка Java [7], а за образец Pascal-подобного принят синтаксис языка Component Pascal [5], реализованного в системе BlackBox Component Builder. Кроме того, каждое из представлений может быть показано либо в русской лексике, либо в английской.

На рис. 2 показано представление программы вычисления факториала на языке Semantic Language в английской и русской лексике.

```
module Факториал
start
  variable-integer i := 1;
  variable-real current := 1;
  constant integer N = 15;
  while i < N repeat
    let current := current * i;
    let i := i + 1;
    output '\n';
    output current;
  end of while;
end Факториал.
```

```
модуль Факториал
начало
  переменная-целое i := 1;
  переменная-вещ current := 1;
  константа целое N = 15;
  пока i < N повторять
    присвоить current := current * i;
    присвоить i := i + 1;
    вывести '\n';
    вывести current;
  конец цикла;
конец Факториал.
```

Рис 2. Представление программы на языке Semantic Language

Та же программа в Си-подобном и Pascal-подобном виде показана на рис. 3.

<pre> package Факториал; function Main() { var int i = 1; var double current = 1; const int N = 15; while (i < N) { let current = current * i; let i = i + 1; Console.Write("\n"); Console.Write(current); } // конец Main } // конец Факториал </pre>	<pre> MODULE Факториал; BEGIN VAR i: INTEGER := 1; VAR current: REAL := 1; CONST N: INTEGER = 15; WHILE i < N DO LET current := current * i; LET i := i + 1; Log.Out("\n"); Log.Out(current); END END Факториал. </pre>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Рис 3. Представление программы на языке Semantic Language
а) Си-подобный синтаксис б) Pascal-подобный синтаксис

В кодах программы синим цветом подсвечены ключевые слова соответствующего языка, а зеленым – слова, не входящие в язык. Эти слова выводятся в кодое окно только для пояснения кода и не представлены в семантическом дереве (их можно отключить).

Каждый синтаксис определяется собственной грамматикой представления. Грамматика не используется для синтаксического анализа программы и нужна только для определения внешнего представления программы в окне редактора кода. В грамматике каждому оператору соответствует правило представления, задаваемое в формате РБНФ. Например, оператор присваивания в грамматике представления Semantic Language, задан таким правилом:

<Assign> ::= "присвоить" <expression> ":" <expression> ";"

Нетерминал в левой части правила – это семантическая единица языка (оператор), которая в семантическом дереве представлена соответствующим узлом. В правой части правила – синтаксис представления этого оператора в окне редактора. Последовательности символов, заключенные в кавычки, выводятся буквально в окно редактора – это ключевые слова языка, которые в коде показаны синим цветом. Нетерминалы, представленные в правой части правил, представляют собой параметры данной семантической единицы. В данном случае узел имеет два параметра: слева параметр-выражение, представляющий объект, которому присваивается значение правого параметра-выражения.

То же правило в грамматике Pascal-подобного представления выглядит так:

<Assign> ::= "%ПРИСВОИТЬ" <expression> ":" <expression> ";"

В грамматике Си-подобного представления это правило выглядит следующим образом:

<Assign> ::= "%присвоить" <expression> "=" <expression> ";"

Знак процента «%» в начале слова означает, что это слово является исключением в данном синтаксисе. В данном случае это правило говорит о том, что в Си-подобных и Pascal-подобных языках отсутствует ключевое слово «присвоить» (английское слово «let») в операторе присваивания. Слово-исключение показано в коде зеленым цветом.

Заключение

Описанная в данной статье интегрированная среда Semantic IDE является основой обучающей системы, требования к которой изложены в [3]. Реализация Semantic IDE выполняется на языке C# в среде Visual Studio 2012. Полностью реализована вся процедурная составляющая учебного языка, и большая часть объектно-ориентированной. Разработана начальная версия системной библиотеки.

В настоящее время среда апробируется в учебном процессе на кафедре АСОИУ Астраханского государственного технического университета для выполнения лабораторных работ по дисциплине «Основы алгоритмизации». Среда используется и на факультативных занятиях по программированию в Астраханском колледже вычислительной техники. Небольшой пока опыт использования показывает существенное сокращение времени при создании кода программы, и практически полное исчезновение ошибок набора. По отзывам преподавателей представление программы в русской лексике облегчает усвоение базовых понятий студентам, не изучавшим основы программирования в школе, и способствует усвоению профессиональной терминологии. С другой стороны, возможность переключить программу в английскую лексику облегчает студентам переход на профессиональные англоязычные языки программирования.

В настоящий момент развитие среды продолжается: реализуется механизм наследования в объектно-ориентированной части языка Semantic Language, разрабатываются модули стандартной библиотеки, развивается система справочной информации, дополняется проект Задачник. Разработан и включен в среду сокращенный Python-подобный синтаксис представления программы. Разрабатываются грамматики для представления программы в синтаксисе C# и Си.

Работа поддержана грантами «У.М.Н.И.К.» в 2011 и 2012 годах, и грантом «Старт» в 2013 году. Работающую версию среды можно загрузить с сайта sem_tech.net.

Литература

1. Грачёв А.Д., Лаптев В.В. Разработка учебного языка программирования и интерпретатора для обучающей среды // Объектные системы-2012: материалы VI Международной научно-практической конференции (Ростов-на-Дону, 10-12 мая 2012г.) / Под общ. ред. П.П. Олейника. – Ростов-на-Дону: ШИ ЮРГТУ (НПИ), 2012. – 110 с., с. 92-101.
2. Давыдов С.В., Ефимов А.А. IntelliJ IDEA. Профессиональное программирование на Java. – СПб.: БХВ-Петербург, 2005. – 800 с.
3. Лаптев В.В. Требования к современной обучающей среде по программированию // Объектные системы-2010 (Зимняя сессия): материалы II Международной научно-практической конференции. Россия, Ростов-на-Дону, 10-12 ноября 2010 г. / Под общ. Ред. П.П. Олейника. – Ростов-на-Дону, 2010. – 134 с., с. 104-110.
4. Пауэрс, Л. Microsoft Visual Studio 2008 / Л. Пауэрс, М. Снелл: Пер. с англ. – СПб.: БХВ-Петербург, 2009. – 1200 с.
5. Потопахин В.В. Современное программирование с нуля! – М.: ДМК Пресс, 2010. – 240 с.
6. Страуструп Б. Дизайн и эволюция C++. – М.: ДМК Пресс; СПб.: Питер, 2006. – 448 с.
7. Хорстман К.С., Корнелл Г. Java 2. Библиотека профессионала, том 1. Основы. – М.: ООО «И.Д. Вильямс», 2011. – 816 с.