БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ
Кафедра веб-технологий и компьютерного моделирования

А. С. Кравчук, А. И. Кравчук, Е. В. Кремень

ЯЗЫК JAVA.
ОБЩИЕ СВЕДЕНИЯ

Учебные материалы
для студентов специальности 1-31 03 08
«Математика и информационные технологии
(по направлениям)»

МИНСК
2023

УДК 004.432.045:004.738.5Java (075.8)
ББК 32.973.2-018.1я73-1
К78

Утверждено на заседании кафедры
веб-технологий и компьютерного моделирования
8 ноября 2022 г., протокол № 3

Рецензент
кандидат физико-математических наук,
доцент *Г. А. Расолько*

Кравчук, А. С.

К78 Язык Java. Общие сведения : учеб. материалы / А. С. Кравчук, А. И. Кравчук, Е. В. Кремень. – Минск : БГУ, 2023. – 33 с.

Рассматриваются причины создания языка Java, факторы, обусловившие его формирование и унаследованные им особенности, а также то, что делает Java столь важным. Описываются методологии программирования, лежащие в основе Java, инструменты для разработки Java-приложений, обсуждаются характеристики общих механизмов безопасности реализации приложений Java SE. Сделан краткий обзор online-компиляторов для Java.

УДК 004.432.045:004.738.5Java (075.8)
ББК 32.973.2-018.1я73-1

Оглавление

Введение

Огромные перспективы и самые захватывающие возможности для разработчиков программного обеспечения лежат в обуздании силы сетей. Приложения, созданные сегодня, для каких бы целей и аудитории они ни планировались, как правило запускаются на машинах, связанных глобальной сетью компьютеров. Возрастающая важность сетей предъявляет новые требования к существующем инструментам и заставляет быстро расти список абсолютно новых типов приложений.

Пользователи хотят иметь программное обеспечение, которое работает стабильно на любой платформе и которое хорошо сочетается с другими приложениями. Пользователи хотят иметь динамические приложения, которые используют преимущества всемирной паутины, способны получить доступ к несоизмеримым и распределенным источникам информации. Пользователи хотят умные приложения, которые могут бродить по сетям вместо нас, разыскивая информацию и служа электронными агентами. Пользователи уже довольно давно знают, какое программное обеспечение хотят, но в действительности оно появилось только сравнительно недавно.

Исторически проблема заключается в том, что инструментов для создания таких приложений не хватает. Требования к скорости и портируемость являются взаимоисключающими, а безопасность в большей степени игнорируется или не понимается. В прошлом действительно мобильные языки были громоздкими, интерпретационными и медленными. Своей популярностью они скорее обязаны высокому уровню функциональности, а не портируемости. Многие языка обычно обеспечивали скорость, привязываясь к определенным платформам, так что они решали проблему портируемости только наполовину. Существовало даже несколько безопасных языков, по они в основном были ответвлениями портируемых, поэтому имели те же проблемы. Язык Java это современный язык, который работает на трех фронтах: портируемость, скорость и безопасность. Поэтому он является одним из доминирующих языков в мире программирования уже долгое время.

Замечание.

Портируемость (или часто в неудачных русских переводах зарубежных книг «портативность») – переносимость, кроссплатформенность.

Появление Java

Язык программирования Java, разработанный компанией Sun Microsystems, под руководством светил интернета Джеймса Гослинга (James Gosling) и Билла Джоя (Bill Joy) был создан как машинно-независимый язык программирования, который достаточно безопасен для сетевых применений. Язык Java решил проблемы, поднятые выше, и сыграл значительную роль в развитии интернета.

Изначально энтузиазм в отношении Java был в основном сосредоточен на его возможностях создания встраиваемых приложений для всемирной паутины.

Сегодня язык Java стал одной из главных платформ для веб-приложений и сервисов. Портируемость (переносимость) и скорость языка Java делают его платформой номер один для современных бизнес-приложений. Программное обеспечение, написанное на Java, работает также на платформах Linux с открытым кодом и находится сегодня в сердце делового и финансового мира.

Происхождение Java

Семена Java были посеяны в 1990 году основателем и главным исследователем компании Sun Microsystems Биллом Джоем. В это время компания Sun конкурировала на относительно небольшом рынке рабочих станций, в то время как компания Microsoft начинала доминировать со своими более массовыми персональными кома компьютерами с микропроцессорами Intel. Когда Sun упустила свой шанс в революции персональных компьютеров. Джой Гослинг отступил в Аспен, Колорадо для перспективных исследований. Он придерживался идеи о выполнении сложных задач с использованием простого программного обеспечения и основал компанию, названную Sun Aspen Smallworks. Среди первых членов маленькой команды программистов. собравшихся в Аспене, Джеймса Гослинга запомнили как отца языка Java. Гослинг впервые прославился вначале 80-х как автор Gosling Emacs - первой версии популярного редактора Emacs. который был написан на языке С и работал под Unix. Редактор Gosling Emacs стал популярным, но вскоре его затмила бесплатная версия — GNU Emacs, написанная первым дизайнером Emacs. К тому времени Гослинг перешел к разработке системы NeWS компании Sun, которая какое-то время соперничала с системой X Windows System для управления рабочим столом с GUI на основе Unix в 1987 году. Хотя некоторые люди придерживаются точки зрения, что система NeWS предшествовала системе X. NeWS проиграла, потому что компания Sun оставляла ее своей собственностью и не публиковала исходный кол, в то

время как первые разработчики X сформировали Консорциум X и использовали противоположный подход.

Разработка NeWS обучила Гослинга силе интеграции языка с GUI оконного типа, ориентированным на сети. Она также научила компанию Sun, что сообщество интернет-программирования, безусловно, отказывается принимать стандарты, защищенные правом собственности, какими бы хорошими они ни были. Семена лицензионной схемы языка Java и открытого кода (пусть даже и не с открытым источником) посеяны неудачей NeWS. Гослинг принес то, чему он научился, в рождающийся проект Билла Джоя в Аспене. В 1992 году работа над проектом правела к основанию корпорации FirsPerson, дочерней корпорации Sun. Ее миссией было вывести Sun в мир потребительской электроники.

Команда компании FirsPerson разрабатывала программное обеспечение для информационной бытовой техники, такой как мобильные телефоны и карманные персональные компьютеры (PDA). Целью было обеспечить переход информации из приложений в реальном времени на дешевые традиционные сети, основанные на пакетном способе передачи данных.

Ограничения в памяти и производительности диктовали компактный и эффективный код. Природа приложений также требовала, чтобы они были безопасными и ясными. Гослинг и члены его команды начали программировать на языке C++, но вскоре были сбиты с толку тем, что язык оказался слишком сложным, громоздким и небезопасным для их задач. Они решили начать с самого начала, и Гослинг начал работать над чем-то, что он называл «C++ минус минус».

С изобретением App1e Newton (первого карманного компьютера корпорации App1e) стало очевидно, что время карманных персональных компьютеров еще не пришло. Поэтому компания Sun переключила усилия FirsPerson на интерактивное телевидение (ITV). Язык программирования, выбранный для телевизионных приставок, стал практически предшественником Java - языком, названным Oak. Даже с его изяществом и возможностью предоставлять безопасную интерактивность язык Oak не смог спасти безнадежное дело ITV в то время. Покупатели его не хотели, и Sun вскоре оставила эту идею.

В то время Джой и Гослинг объединились, чтобы выбрать новую стратегию для своего инновационного языка. Шел 1993 год и вспышка интереса ко всемирной паутине предоставила им новую возможность. Язык Oak был компактным, безопасным, независимым от архитектуры и объектно-ориентированным. Как оказалось, это также являлось требованиями и к универсальному. понятному интернет-языку программирования. Sun быстро сменила фокус, и после небольших изменений язык Oak стал языком Java.

Становление

Не было бы преувеличением сказать, что язык Java разгорелся как дикий огонь. Даже до его официального выпуска язык Java использовался почти каждой крупной компанией отрасли. Лицензиатами Java были Microsoft, Intel, IBM и почти все основные производители аппаратных средств н программного обеспечения. Однако даже со всей этой поддержкой язык Java потерпел много неудач в свои первые годы.

Серия нарушений договоров и антитрестовских судебных процессов между компаниями Sun и Microsoft по поводу распространения Java и его использования в браузере Internet Explorer препятствовала применению его в самой распространенной в мире операционной системе для настольного компьютера - Windows. Конфликт Microsoft и Java также стал одной из причин большого федерального дела о серьезных нарушениях правил конкуренции в деятельности компаний, в результате чего свидетельские показания на суде выявили согласованные усилия гиганта программного обеспечения с целью подорвать Java, демонстрируя недоработки в их версии языка. В то же время Microsoft представила свои собственный язык C# (C-шарп), основанный на Java, как часть своей платформы .NET.

Java продолжает распространяться на самых разнообразных платформах. Когда читатель начнет знакомиться с архитектурой Java, то он увидит, что много интересного определяется автономной средой виртуальной машины, на которой работают приложения Java. Язык Java очень тщательно разработан. Она может быть применен как в программном обеспечении для существующих компьютерных платформ, так и для использования в аппаратном обеспечении. Java в аппаратном обеспечении используется в некоторых чиповых картах и других встроенных системах. Применение программного обеспечения Java доступно для всех современных компьютерных платформ, включая портативные компьютерные устройства. Сегодня ответвление платформы Java является основой операционной системы Android от Google, на которой работают миллиарды телефонов н других мобильных устройств.

В 2010-м корпорация Oracle купила Sun Microsystems и стала владельцем языка Java. Начало было непростым: Oracle судилась с Google из-за использования языка Java в Android и проиграла. Однако язык Java продолжает активно развиваться и в настоящее время новые версии Java выходят каждые 6 месяцев. В прошлом циклы выпуска Java были намного длиннее, до 3-5 лет.

Виртуальная машина

Язык Java является одновременно и компилируемым н интерпретируемым языком. Исходный код Java превращен в простые бинарные инструкции, что больше похоже на машинный код микропроцессора. В отличие от C или C++, где исполняемый файл компилируется до машинных команд, код Java компилируется в универсальный формат - инструкции дли виртуальной машины.

Скомпилированный байт-код Java выполняется интерпретатором времени выполнения. Система поддержки выполнения выполняет все обычные действия процессора аппаратного обеспечения, но делает это в безопасной виртуальной среде. Она выполняет набор стековых инструкций и управляет памятью как операционная система. Она создает простые типы данных и управляет ими, загружает и вызывает вновь блоки кода. Что самое главное, она делает это в соответствии с четко определенной открытой спецификацией, которую может применить любой, кто хочет создать виртуальную машину Java. Вместе виртуальная машина и определение языка представляют полную спецификацию. Нет свойств базового языка Java. которые бы остались неопределенными или зависимыми от внедрения. Например, язык Java специфицирует размеры и математические свойства всех своих простых типов данных, а не оставляет их на усмотрение реализации платформы.

Интерпретатор Java довольно легковесный и компактный, он может быть реализован в любой форме, подходящей для определенной операционной системы. Интерпретатор может работать как отдельное приложение или может быть внедрен в другое программное обеспечение, такое как веб-браузер. Все сказанное означает, что код Java полностью портируем (переносим). Один и тот же байт-код приложения Java может работать на любой платформе, что обеспечивает среду выполнения Java (Рисунок 1). Вам не нужно производить альтернативные версии вашего приложения для разных операционных систем и не нужно раздавать исходный код конечным пользователям.

Исторически интерпретаторы считались медленными, но Java не является традиционным интерпретированным языком. В дополнение к компиляции исходного кода в портируемый (переносимый) байт-код Java была тщательно разработана таким образом, что программная реализация среды выполнения может далее оптимизировать свою производительность, компилируя байт-код в оригинальный машинный код «на лету». Это называется компиляция «точно в срок», или динамическая компиляция. С динамической компиляцией код Java может выполняться достаточно быстро и поддерживать свою переносимость (портируемость) и безопасность.

Существует только одна серьезная проблема, от которой страдает производительность компилируемого Java-кода во время выполнения - это проверка границ массива ради безопасности виртуальной машины. Все остальное может быть оптимизировано в исполняемый код так же, как при статической компиляции языка. Кроме этого, язык Java содержит больше структурной информации, чем многие другие языки, предусматривая больше типов оптимизации. Также помните, что эти оптимизации могут осуществляться во время выполнения, принимая во внимание реальное поведение и характеристики приложений.

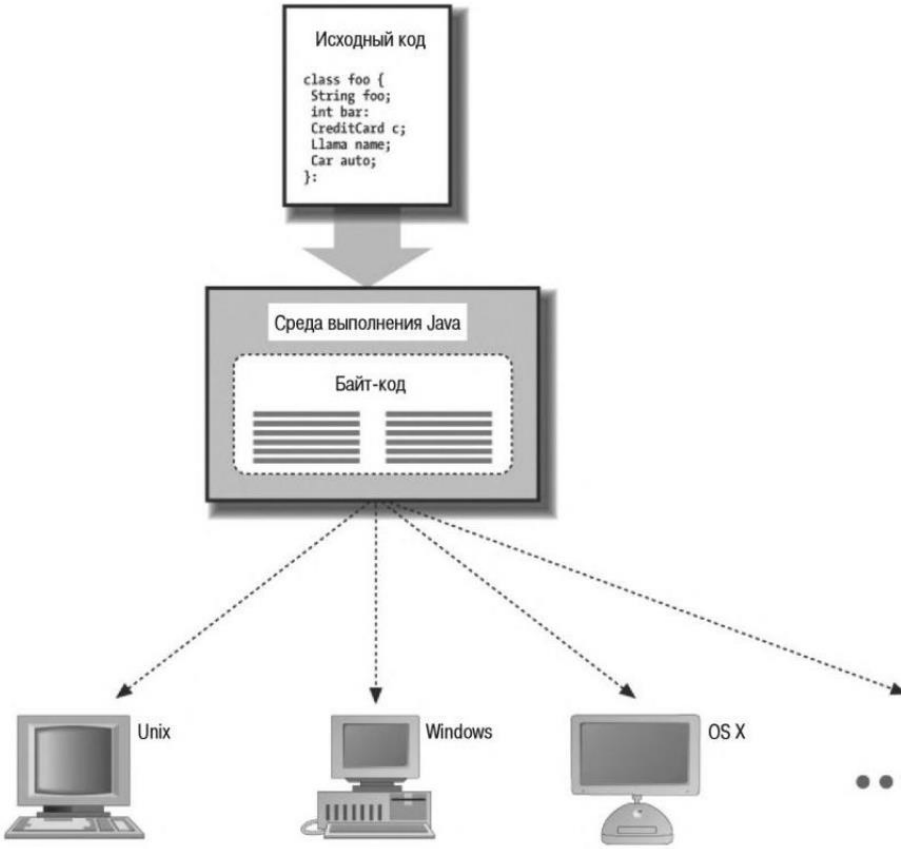

Рисунок 1 – Среда выполнения Java

Проблема традиционной динамической компиляции - это то, что оптимизация кода занимает время. Итак, динамический компилятор может давать достойные результаты, но может страдать от значительного времени ожидания при запуске приложения. В основном это не имеет большого значения для долго работающих приложений серверной стороны. но является серьезной проблемой для программного обеспечения клиентской стороны и приложений, которые работают на меньших устройствах с ограниченными возможностями. Чтобы решить этот вопрос, технология компилирования Java, называемая HotSpot, использует прием, именуемый адаптивной оптимизацией. Если посмотреть, на что в действительности расходуется время работы программ, окажется, что они тратят почти все

время, выполняя относительно небольшую часть кода снова и снова. Участок кода, который повторно выполняется, может быть только малым фрагментом всей программы, но его поведение определяет общую производительность программы. Кроме этого адаптивная оптимизация при динамической перекомпиляции кода Java позволяет, например, использовать преимущества различных типов трансляции (интерпретации или компиляции), а также некоторые другие методики оптимизации.

Чтобы получить преимущество от этого факта, технология HotSpot начинает как обычный интерпретатор байт-кода Java, но с одной разницей: она измеряет (профилирует) код во время его выполнения, чтобы увидеть, какие части кода выполняются многократно. Когда она знает, какие части кода являются критическими для производительности, технология HotSpot компилирует эти части в оптимальный собственный машинный код. Поскольку она компилирует только маленькую часть программы в машинный код, то она имеет возможность использовать время, необходимое для оптимизации этих частей. Остальная часть программы может вообще не нуждаться в компиляции, а только в интерпретации, что сохраняет память и время. В действительности виртуальная машина Java может работать в двух режимах: клиентском и серверном, что определяет выбор, делать ли акцент на быстром времени старта и сохранении памяти или на скорости выполнения приложения.

Java в сравнении с другими языками

В своем выборе свойств язык Java опирается на многолетний опыт программирования на других языках. Следует уделить немного времени сравнению Java на высоком уровне с некоторыми другими языками. Эго будет полезно как для тех, у кого имеется опыт программирования на других языках, так и для новичков, которым необходимо хорошо разобраться в теме. При изложении данного материала никто не рассчитывает на то, что читатель знает еще какой-либо язык программирования.

Как уже отмечалось как минимум три опорные идеи необходимы для поддержки универсального языка программирования сегодня: портируемость (переносимость), скорость и безопасность.

Вы. возможно, слышали, что язык Java во многом похож на язык C или C++, но в действительности это только внешнее сходство. Когда читатель впервые взглянет на Java-код, то увидит, что базовый синтаксис похож на C или C++. Но на этом сходство и заканчивается. Язык Java ни в коем случае не является прямым потомком C или следующим поколением C++. Если вы сравните свойства этих языков, увидите, что у Java в действительности больше общего с динамическими языками высшего уровня, такими как Smalltalk и Lisp. В действительности реализация Java

настолько далека от реализации C++, насколько это только можно себе представить.

Следует отметить, что такой современный язык как C# в целом это проект-ответ Microsoft на проект Java, имеющий много преимуществ при поверхностном рассмотрении. При общем дизайне и подходе при реализации платформы (например, использование виртуальной машины, байт-код, и др.) эти два языка, по существу, не отличаются в показателях скорости или характеристиках безопасности. C# теоретически является таким же портируемым (переносимым), как Java, но сегодня он поддерживается на гораздо меньшем количестве платформ. Как и Java, C# много позаимствовал из синтаксиса C, но в действительности является близким родственником динамических языков. Многие разработчики Java считают, что понять C# достаточно легко, и наоборот. Большая часть времени, которое тратится при переходе с одного языка на другой, уходит на изучение стандартной библиотеки.

Язык Java берет многое от синтаксиса языка C и C++, так что читатель увидит краткие языковые конструкции, включая изобилие фигурных скобок и точек с запятой. Язык Java подписывается под философией C, заключающейся в том, что хороший язык должен быть компактным, другими словами, он должен быть достаточно легковесным и регулярным, только так Java-программист сможет сразу запомнить все его возможности. Следует отметить, что, как и в случае с расширяемостью библиотек языка C, пакеты с классами языка Java могут добавляться в центральные компоненты языка для расширения его словаря.

Язык C успешен, поскольку предоставляет программную среду со значительным содержанием свойств, с высокой производительностью. Язык Java также старается балансировать между функциональностью, скоростью и портируемостью (переносимостью), но совсем другим образом. Язык Java также решает проблемы безопасности, которые язык C не решает (хотя в современных системах многие из этих проблем сейчас решаются операционной системой и аппаратным обеспечением).

В самом начале, до внедрения динамической компиляции и адаптивной оптимизации, язык Java был медленнее, чем статически компилируемые языки, и злые языки постоянно повторяли, что он никогда не достигнет достойного уровня. Но в настоящее время в связи с приложенными усилиями эта критика в общем сошла на нет. Движок видеоигры с открытым кодом Quake2, созданной ID Software, был переписан Java. Если язык Java является достаточно быстрым для игры-шутера от первого лица, то он определенно достаточно быстр для бизнес-приложений.

Замечание.

Язык сценариев (или скриптовый) – это язык программирования, который разработан для записи последовательностей операций («сценариев»), выполняемых пользователем на своем компьютере. Раньше назывался языком пакетной обработки. В зависимости от быстродействия различают языки сценариев предварительно компилируемые (например, широко используемый для создания и продвижения сайтов Perl) и динамического разбора (command.com, sh). Первые транслируют программу в байт-код, который затем исполняют. Языки динамического разбора считывают инструкции из файла программы минимально необходимыми блоками, которые исполняют, не читая, дальнейший код.

Языки сценариев, такие как Perl, Python, и Ruby, очень популярны. Нет причин, чтобы язык сценариев не подходил для безопасных сетевых приложений. Но многие скриптовые языки плохо подходят для крупномасштабного программирования. Привлекательным в скриптовых языках является то, что они динамические и являются мощным инструментом для быстрой разработки. Некоторые языки сценариев, такие как Perl, также предоставляют мощные инструменты для задач по обработке текста, которые многие универсальные языки находят громоздкими. Скриптовые языки также портируемы, хотя и на уровне исходного кода.

JavaScript не стоит путать с Java, так как это объектно-ориентированный скриптовый язык, изначально разработанный Netscape для веб-браузеров. Он является «постоянным жителем» веб-обозревателя для динамических, интерактивных приложений. Название JavaScript происходит от его интеграции с Java и некоторых общих черт, но сходство в действительности здесь и заканчивается. Хотя приложения на JavaScript вне браузера есть, он так по-настоящему и не прижился.

Проблема языков сценариев заключается в том. что структура программ и типы данных в них довольно бессистемны. Большинство скриптовых языков (за колеблющимся исключением Python и последних версий Perl) не являются объектно-ориентированными. У них также упрощенные системы типов, но в основном они не предусматривают сложных пользовательских классов и функций. Эти характеристики делают их менее подходящими для создания больших модульных приложений. Скорость еще одна проблема скриптовых языков; высокоуровневая, зачастую с интерпретацией источника, природа этих языков делает их довольно медленными.

Защитники отдельных скриптовых языков не согласятся с некоторыми из этих обобщений и, без сомнения, будут правы в ряде случаев. Скриптовые языки улучшились за последние годы особенно язык

JavaScript, на улучшение производительности которого затрачено большое количество ресурсов. Но фундаментальная уступка неоспорима: скриптовые языки появились как свободная, менее структурированная альтернатива системным языкам программирования и в основном неидеальны для больших или сложных проектов по различным причинам, по крайней мере сегодня.

Java предлагает некоторые существенные преимущества скриптового языка: он в высшей степени динамичен вместе с добавленными плюсами языков низшего уровня. У Java мощный интерфейс Regular Expression API, который соперничает с языком Perl в работе с текстом н средствах языка, ускоряющих программирование с помощью библиотек, списков переменных аргументов, статического импорта методов и других синтаксических «плюшек», которые делают его более сжатым.

Замечание.

Инкрементная модель — это метод, в котором проект проектируется, реализуется и тестируется инкрементно (то есть каждый раз с небольшими добавлениями) до самого окончания разработки. Это включает в себя как разработку, так и дальнейшую поддержку продукта. Он считается законченным в то время, когда удовлетворяет всем требованиям. Модель объединяет элементы каскадной модели с прототипированием.

Инкрементная разработка с объектно-ориентированными компонентами в сочетании с простотой Java делает разработку приложений быстрой, а их изменение легким. Исследования показали, что разработка на языке Java осуществляется быстрее чем на C или C++. Java также идет с большой базой стандартных классов ядра для общих задач, таких как разработка GUI н управление сетевыми связями. Но вместе с этими свойствами язык Java имеет масштабируемость и преимущества разработки ПО, присущие больше статическим языкам. Он предоставляет безопасную базу для создания высокоуровневых структур (и даже других языков).

Замечание.

В данном случае речь идет о масштабируемости языка программирования. Это значит, что масштабируемость — это показатель того, на сколько можно изменить возможности программы, добавив не стандартный, а написанный разработчиком модуль, метод или функцию, не потеряв при этом в производительности.

Масштабируемость языка может также трактоваться как способность написанных на нем web-приложений при той же функциональности с небольшой потерей эффективности обработать значительно возросшее количество одновременных интернет-запросов.

Кроме того, в зависимости от контекста под масштабируемостью языка программирования может пониматься возможность одновременной работы многих людей над одним большим программным продуктом, путем разбиения проекта на более мелкие подзадачи для каждого участника.

Как уже сказано, по конструкции Java похож на Smalltalk и Lisp. Однако эти языки использовались больше, как аппараты для научных исследований, чем для создания крупномасштабных систем. Одной из причин является то, что у этих языков никогда не было стандартных портируемых связей с сервисами операционных систем, таких как стандартная библиотека C или базовые классы Java. Smalltalk компилируется в интерпретированный формат байт-код и может интерпретироваться в собственный код «на лету», как Java. Но Java улучшен в реализации за счет использования верификатора байт-кода для обеспечения корректности скомпилированного Java-кода. Этот верификатор дает Java преимущество в производительности перед Smalltalk, поскольку Java-код требует меньших проверок рабочего цикла программы. Верификатор байт-кода также помогает с решением проблем безопасности, в отличие от Smalltalk.

Методологии программирования, лежащие в основе Java

Java возникла на пике популярности технологий ***объектно-ориентированного*** программирования (ООП) и включает все основные ее парадигмы.

ООП — методология программирования, основанная на функционировании программного продукта как результата взаимодействия совокупности объектов, каждый из которых является экземпляром конкретного класса.

Объект — именованная модель реальной сущности, обладающая конкретными значениями свойств и проявляющая свое поведение.

Класс — модель информационной сущности, представляющая универсальный тип данных, состоящая из набора полей данных и методов их обработки.

В применении к объектно-ориентированным языкам программирования понятия объекта и класса конкретизируются. Во всех языках классы и объекты обладают рядом общих свойств, таких как инкапсуляция (использование закрытых данных и открытых методов), наследование (заимствование функциональности базовых классов производными), полиморфизм (возможность использования методов с

одинаковыми именами для обработки данных различного типа или объектов с частично совпадающим интерфейсом при наследовании).

В Java как и в других объектно-ориентированных языках, классы являются компонентами приложения. Скомпилированные классы Java распространяются в универсальном бинарном формате, содержащем байт-код Java и другую информацию класса. Классы могут сохраняться отдельно н храниться в файлах или архивах локально или на сетевом сервере. Классы подгружаются динамически во время исполнения, когда они необходимы приложению.

В дополнение к платформо-зависимым средам выполнения программ Java имеет ряд фундаментальных классов, которые содержат архитектурно-зависимые методы. Эти «родные» методы служат воротами между виртуальной машиной Java и реальным миром. Они реализованы в первоначально скомпилированном языке на платформе хоста и предоставляют доступ нижнего уровня к ресурсам, таким как сеть, система управления окнами н файловая система хоста.

Функциональное программирование было добавлено в очередную версию Java и ориентировано на вычисления и обработку информации. Этот подход предполагает формирование функции как объекта и передачу этого объекта в метод для использования или, наоборот, сама функция есть возвращаемое методом значение. Функциональный стиль программирования не заменит другие подходы к созданию кода. Многие действительно последовательные процессы, такие как поведение программных моделей в реальном времени, игровые и другие программы, организующие взаимодействие компьютера с человеком, не могут быть выражены в функциональном стиле. Функциональное программирование позволяет по-иному взглянуть на процесс программирования, а некоторые приемы программирования, которые предназначены для написания программ в чисто функциональном стиле, могут с успехом использоваться и в традиционном программировании совместно с ООП.

Безопасная разработка

Java разработан как безопасный язык. Но что подразумевается под безопасностью? Безопасность от чего или кого? Средства безопасности, которые привлекают много внимания к Java, это те свойства, которые делают возможным разработку безопасного портируемого (переносимого) программного обеспечения. Язык Java обеспечивает несколько уровней защиты от опасно поврежденного кода, а также других вредных явлений, таких как вирусы и троянские кони. Далее будет разъяснено то, как архитектура виртуальной машины определяет безопасность кода перед его запуском и как загрузчик классов (механизм загрузки интерпретатора Java в байт-коде) строит стену вокруг ненадежных классов. Эти средства

составляют основу для формирования высокоуровневой политики безопасности, с помощью которой можно разрешать или запрещать разного рода действия для всех приложений в рамках виртуальной машины.

Задача языка Java — быть как можно более защищенным как от простых ошибок, которые делает разработчик сам, так и от тех, которые попадают к нам с унаследованным ПО. Целью Java было оставить язык простым, предоставить инструменты, которые продемонстрируют свою полезность и позволят пользователю создавать при необходимости более сложные объекты на высшем уровне языка.

Забытый в современных версиях Java базовый принцип: «упрощение, упрощение, упрощение»

При разработке Java простота являлась правилом, поскольку язык Java начинался чистого листа, ему удалось избежать свойств, которые оказались запутанными или противоречивыми в других языках. Например, Java не допускает перегрузку операций, определенных программистом (которая в некоторых языках позволяет программистам переопределять значение базовых символов, таких, как ‘+’ и ‘-‘). У языка Java нет препроцессора исходного кода, так что в нем нет таких вещей, как макросы, инструкции: #define или компиляция условного источника. Условная компиляция также широко используется для отладки, но сложная оптимизация рабочего цикла Java и такие свойства, как утверждения, решают проблему более элегантно.

Хотя при создании язык Java предполагался более простым, чем его синтаксический предок C++. Сегодня с появлением новых версий возможности языка Java существенно расширились и во многом перекрывают функциональность C++. Java уже не уступает по сложности предшественникам и ***называть его простым нельзя***.

Замечание.

Утверждение в Java – это лексема, которая обеспечивает правильность любых предположений, сделанных в программе. Когда утверждение выполнено, оно считается истинным. Если утверждение ложно, JVM выдаст ошибку. Они находит применение в первую очередь в тестировании. Утверждения используются вместе с логическими выражениями. Утверждения в Java обычно формулируются с помощью ключевого слова `assert`.

Язык Java предоставляет четкую структуру пакетов для организации: файлов классов. Компилятор также может работать непосредственно с компилированными файлами Java, потому что сохраняется вся информация

типа; нет необходимости во внешних «заголовочных» файлах, как в C/C++. Все это означает, что Java-коду нужно читать меньше контекста.

Современный язык

Java поддерживает только иерархию классов простого наследования (у каждого класса может быть только один «родительский» класс), но позволяет множественную имплементацию интерфейсов. Интерфейс как абстрактный класс в C++ устанавливает поведение объекта без определения его выполнения. Это очень мощный механизм, позволяющий разработчику определить «контекст» поведения объекта, который может использоваться и упоминаться независимо от частной реализации объекта. Интерфейсы в Java исключают потребность в множественном наследовании классов и связанные с этим проблемы.

Со временем, погружаясь в язык Java читатель увидит достаточно простой и изящный язык программирования, и в этом в первую очередь заключается его притягательная сила.

Безопасность типов и связывание методов

Один из атрибутов языка — это вид проверки типов, который он использует. Как правило, языки распределяются по категориям как статические и динамические, что относится к количеству информации о переменных, известных во время компиляции, против того, что известно во время работы приложения.

В статически типизированных языках, таких как C и C++, типы данных словно выбиты на камне, когда компилируется исходный код. Компилятор получает преимущество от этого, поскольку имеет достаточно информации для отлова многих типов ошибок перед выполнением кода. Например, компилятор не позволит вам хранить значение с плавающей точкой в переменной целого типа. Код тогда не нуждается в проверке типов во время выполнения, поэтому он может быть компилирован, чтобы стать компактным и быстрым. Но языки со статическими типами являются негибкими. Они не поддерживают библиотеки так же естественно, как языки с динамической проверкой типов, и делают для разрабатываемого приложения невозможным безопасный импорт новых типов данных, когда оно запускается на выполнение.

Для сравнения: такой динамический язык, как Smalltalk или Lisp, имеет исполняемую систему, которая управляет типами объектов и производит необходимую проверку типов, пока приложение выполняется. Эти языки позволяют более сложное поведение и во многих аспектах более

мощные. Однако в основном они медленнее, менее безопасны и тяжелы в отладке.

Разница между языками похожа на разницу между автомобилями. Языки со статическими типами, такие как C++, аналогичны спортивным машинам: довольно безопасны и быстры, но пригодны, только если вы едете по хорошо асфальтированной дороге. Высоко динамические языки, такие как Java, более похожи на внедорожники: они дают больше свободы, но могут быть немного неуклюжими.

Ещё одна особенность языка — это то, как он связывает вызов методов с их определениями. В таких статических языках, как C или C++, определения методов обычно связываются во время компиляции, если программист не укажет иное. Языки, подобные Smalltalk, называются также языками с отложенным связыванием, поскольку они локализируют определения методов динамически во время прогона. Раннее связывание важно из соображений выполнения; приложение может работать без затрат на поиск методов во время прогона. Но позднее связывание более гибкое, оно также необходимо в объектно-ориентированном языке, где новые типы могут подгружаться динамически, и только система выполнения может определять, какой метод запускать.

Язык Java предоставляет некоторые преимущества C++ и Smalltalk, это язык со статическими типами и отложенным связыванием. Каждый объект в Java имеет четко определенный тип, который известен во время компиляции. Это означает, что компилятор Java может выполнять статические проверки типов, как в C++. В результате разработчик не может связать объект с неправильным типом переменной или несуществующие методы с объектом. Компилятор Java идет еще дальше и не дает вам использовать неинициализированные переменные и создавать невозможные для исполнения инструкции.

Однако Java в полной мере является языком с динамическим выполнением. Система выполнения Java следит за всеми объектами и делает возможным определение их типов и связей во время выполнения. Это означает, что вы можете инспектировать объект во время выполнения, чтобы определить, что это за объект. В отличие от языка C или C++ смена одного объекта на другой отмечается системой выполнения и возможно использование новых типов динамически загружаемых объектов с сохранением достаточного уровня безопасности. И поскольку Java — язык с отложенным связыванием, подкласс всегда может переопределить методы в его родительском классе, даже если этот подкласс загружен во время выполнения.

Инкрементная разработка

Язык Java переносит все типы данных и информацию о сигнатуре метода с исходного кода в форму компилированного байт-кода. Это означает, что классы Java могут разрабатываться инкрементно. Собственный исходный код Java любого разработчика может также безопасно компилироваться с классами из других источников, которые компилятор никогда не видел. Другими словами, разработчик может писать новый код, ссылающийся на файлы бинарных классов без потери уровня безопасности.

Управление динамической памятью.

Некоторые из самых важных отличий между Java и такими языками, как C и C++, связаны с тем, как Java управляет памятью. Язык Java устраняет случайные «указатели», которые могут ссылаться на произвольные области памяти, и добавляет сборку мусора для объектов и массивы высокого уровня к языку. Эти свойства устраняют проблемы безопасности, портируемости и оптимизации, непреодолимые другим путем.

Сборка мусора спасла бесчисленное количество программистов от одного большого источника ошибок программирования на языках Си C++: явного распределения и освобождения памяти. В дополнение к управлению объектами памяти система выполнения Java отслеживает все ссылки на этот объект. Когда объект больше не используется, Java автоматически убирает объект из памяти, можно по большей части просто игнорировать объекты, которые больше не используются в программе, с уверенностью, что интерпретатор уберет их в нужное время.

Язык Java использует сложный сборщик мусора, который работает в фоновом режиме, то есть большая часть сборки мусора происходит в нерабочее время, между паузами ввода/вывода, щелчками мышью или нажатием клавиш. Более продвинутые системы выполнения, такие как HotSpot, имеют более развитую сборку мусора, которая умеет различать шаблоны использования объектов (например, кратковременные от долговременных) и оптимизировать их сборку. Время выполнения Java теперь может настраиваться автоматически для оптимального распределения памяти для разных типов приложений в зависимости от их поведения. С подобной градацией времени выполнения автоматическое управление памятью может осуществляться более быстро, чем управляемые программистами ресурсы, — то, во что программисты старой школы до сих пор слабо верят.

Известно, что у Java нет указателей. Собственно говоря, это утверждение верно, но оно также и обманчиво. Ссылки — вот что предлагает Java — безопасный тип указателей. Ссылка — это строго типизированный идентификатор объекта. Все объекты в Java, за исключением базовых числовых типов, доступны через ссылки. Можно использовать ссылки для всех обычных типов структур данных, при использовании которых программист на языке C привык создавать указатели. Единственное отличие в том, что со ссылками все обращения к памяти выполняются безопасным путем.

Другое важное отличие между ссылкой и указателем — это то, что невозможно играть в игры со ссылкой (осуществлять арифметику указателей), чтобы изменить их значения; они могут указывать только на определенные объекты или элементы массива.

Ссылка — это элементарная вещь; вы не можете управлять значением ссылки иначе, чем назначив ее объекту. Невозможно сослаться на объект более чем через один косвенный уровень (невозможно создать ссылку на ссылку). Защита ссылок — один из самых фундаментальных аспектов безопасности. Это означает, что код Java должен играть по правилам; он не может считываться в местах, где не следует, и обходить правила.

Ссылки Java могут указывать только на тины классов. Нет никаких указателей на методы. Некоторые жалуются на их отсутствие, но можно обнаружить, что многие задачи, которые требуют указателей на методы, могут выполняться более аккуратно при использовании интерфейсов и классов адаптеров.

В завершение необходимо отметить, что массивы в Java являются подлинными объектами. Они могут динамически назначаться и присваиваться как другие объекты. Массивы знают свои размеры и типы. Наличие подобных массивов в языке уменьшает нужду в арифметике указателей, которая используется в C или C++.

Обработка исключений

Корневые каталоги Java находятся в сетевых устройствах и встроенных системах. Для таких приложений очень важно иметь надежное и программируемое управление ошибками (исключительными ситуациями). У Java есть мощный механизм обработки исключений, что-то наподобие языка C++. Исключения обеспечивают более естественный и элегантный метод управления ошибками. Исключения позволяют отделить код управления ошибками от обычного кода, что способствует появлению более чистых и надежных приложений.

Когда возникает исключение, оно вызывает переход выполнения программы с обычного потока на запланированный блок кода «отлова». Исключение влечет за собой обращение к объекту, содержащего

информацию о ситуации, которая вызвала исключение. Это выдвигает информацию об исключительной ситуации на тот же уровень важности, что и передача аргументов, и возвращение значений методов. Программист Java, должен точно знать, с какими исключениями придется иметь дело, и у него есть помощь компилятора в написании правильного программного обеспечения, которое не оставит эти исключения необработанными.

Потоки

Современные приложения требуют высокой степени параллелизма, даже очень узкоспециализированное приложение может иметь сложный интерфейс пользователя, что требует параллельных операций. В то время как машины становятся быстрее, пользователи становятся более требовательными ко времени, которое тратится на ожидание, Потоки обеспечивают эффективную многопроцессорную обработку и распределение задач между клиентом и серверными приложениями. Язык Java делает потоки: легкими в использовании, потому что их поддержка встроена в язык,

Параллелизм хорош, но это в большей степени относится к программированию с потоками, чем просто к выполнению множества задач одновременно. В большинстве случаев потоки должны быть синхронизированы (координированы), что может быть сложно без ясной языковой поддержки. Язык Java поддерживает синхронизацию, основанную на модели «наблюдение и условие» — тип системы «замка и ключа» для доступа к ресурсам. Ключевое слово синхронизирует объявленные методы и блоки кода для безопасного упорядоченного доступа внутри объекта, Также есть простые, примитивные методы для ясного ожидания и передачи значений между потоками, заинтересованными в одном объекте,

У Java также есть пакет параллельной обработки, который предоставляет мощные утилиты, адресующиеся к общим шаблонам в многопоточном программировании такие как пул потоков, координация задач и сложное связывание.

Хотя некоторым разработчикам, возможно, никогда не доведется написать многопоточный код, обучение программированию потоками является важным слагаемым в мастерстве программирования на Java и тем, что все разработчики должны знать.

Масштабируемость

На низшем уровне программы Java состоят из классов. Классы были задуманы как маленькие модульные компоненты. Над классами Java

предоставляет пакеты, слой структуры, которая группирует классы в функциональные единицы. Пакеты предоставляют соглашения о наименовании для организации классов и второй слой организации контроля над видимостью переменных и методов в приложениях Java.

Внутри пакета класс является либо публично видимым, либо защищенным от внешнего доступа. Они пригодны для создания компонентов многоразового использования, работающими вместе в системе. Пакеты также помогают в создании масштабируемого приложения, которое может расти.

Безопасность реализации

Одно дело — создать язык, который не даст разработчику навредить самому себе; другое — создать язык, благодаря которому другие не будут приносить вред.

Инкапсуляция — это принцип сокрытия данных и поведения внутри класса; это важная часть объектно-ориентированного программирования. Она помогает вам писать чистое, модульное программное обеспечение. Во многих языках, однако, видимость элементов данных — просто часть связи между программистом и компилятором. Это вопрос семантики, а не утверждение о действительной безопасности данных в контексте выполнения программной оболочки.

Когда Бьерн Страуструп выбрал ключевое слово private для обозначения скрытых членов классов в C++, скорее всего, он думал о защите разработчика от беспорядочных деталей кода другого разработчика, а не о проблемах защиты классов и объектов этого разработчика от атак чьих-то вирусов и троянских коней. Арифметика указателей, а также возможность обращения к произвольным областям памяти через нетипизированные указатели в языке C или C++ делают обычным делом нарушение доступов к свойствам классов без нарушения концептуальных правил языка.

Если приложение Java предназначено для динамической загрузки кода из ненадежного источника во всемирной паутине и запуска его параллельно приложениями, которые могут содержать конфиденциальную информацию, защита должна стать очень глубокой. Модель безопасности Java оборачивает импортированные классы в три уровня защиты (Рисунок 2).

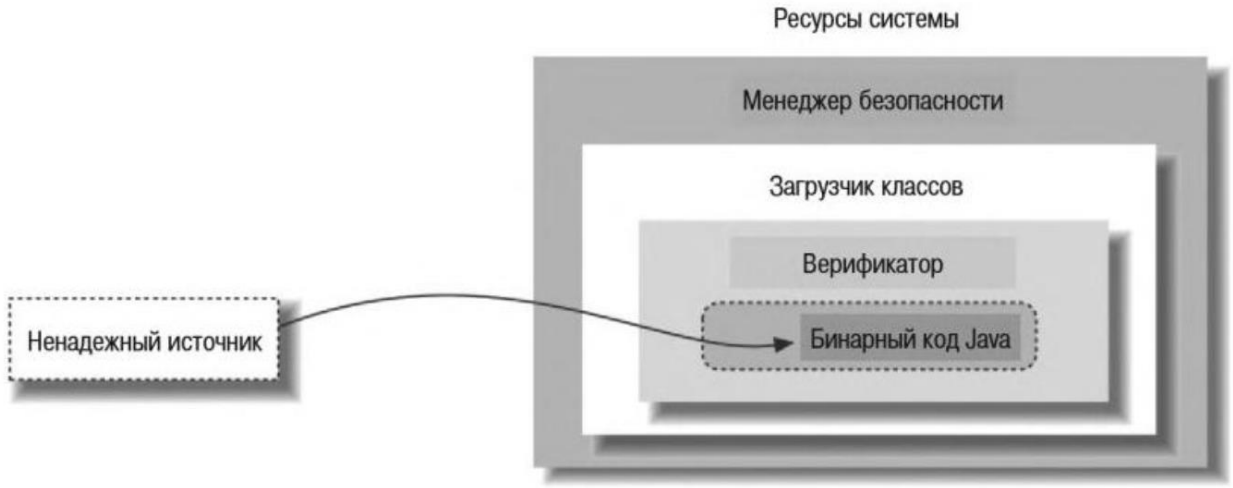

Рисунок 2 – Модель безопасности Java

Снаружи решение безопасности на уровне приложения производится диспетчером (менеджером) безопасности совместно с гибкой политикой безопасности, Диспетчер (менеджер) безопасности контролирует доступ к системам ресурсов, таким как файловые системы, сетевые порты и системы управления окнами. Диспетчер (менеджер) безопасности полагается на возможность загрузчика классов защитить базовую систему классов. Загрузчик классов обрабатывает загружаемые классы отдельно от локального хранилища или сети. На внутреннем уровне вся безопасность системы в конечном счете лежит на верификаторе Java, что гарантирует целостность входящих классов.

Верификатор байт-кода Java является частью системы выполнения Java. Загрузчики классов и диспетчеры (менеджеры) безопасности (или, если быть более точными, политика безопасности), однако, являются компонентами, которые могут выполняться по-разному в разных приложениях, таких как серверы или веб-браузеры. Все эти три компонента (Рисунок 2) должны функционировать должным образом, чтобы обеспечивать безопасность среды Java.

Верификатор

Первая линия защиты языка Java — байт-кодовый верификатор. Верификатор считывает байт-код перед его запуском и убеждается, что он работает хорошо и подчиняется общим правилам языка Java. Надежный компилятор Java не производит код, который ведет себя по-другому.

Однако какой-то злой человек может преднамеренно собрать плохой байт-код Java. Работа верификатора заключается в том, чтоб обнаружить это.

После того как код был проверен, он считается застрахованным от невнимательности или злоумышленных ошибок. Например, проверенный

код не может изобрести ссылки или нарушить уровень доступности объекта. Он не может выполнять нелегальных приведений типов или использовать объект не предусмотренными путями. Он даже не может вызвать определенные типы внутренних ошибок, такие как переполнение или очистку стека. Эти фундаментальные гарантии лежат в основе безопасности Java.

Возможно, возникает вопрос, не используется ли подобный тип безопасности в большинстве интерпретируемых языков? Что ж, хотя и правда вряд ли сможете разрушить интерпретатор Basic поддельной строчкой кода Basic, помните, что защита во многих интерпретируемых языках происходит на высоком уровне. Вероятно, эти языки имеют тяжеловесные интерпретаторы, которые выполняют большую часть работы, поэтому они непременно более медленные и громоздкие.

В то же время байт-код Java является относительно простым набором инструкций низкого уровня. Способность статически проверять байт-код Java перед выполнением позволяет интерпретатору Java работать на полной скорости с полной безопасностью после, без долгих по времени проверок. Это было одним из базовых нововведений Java.

Верификатор — это тип математического «доказательства теорем». Он последовательно читает весь код Java и применяет простые, индуктивные правила для выявления определенных аспектов того, как себя поведет байт-код. Такой тип анализа возможен, поскольку компилируемый байт-код Java содержит намного больше информации о типе, чем объектный код другого языка этого типа. Байт-код также должен следовать нескольким дополнительным правилам, которые упрощают его поверку.

Во-первых, большинство инструкций байт-кода работает только с пользовательскими типами данных. Например, в стековых операциях есть отдельные инструкции для ссылок объекта и для каждого из числовых типов в Java. Проще говоря, для перемещения каждого типа значения в локальную переменную и из нее инструкции разные.

Во-вторых, тип объекта, который получится в результате любой операции, всегда известен заранее. Никакая операция байт-кода не уничтожает переменную и не производит более одного возможного типа переменных на выходе. В результате всегда можно взглянуть на следующую инструкцию и ее операнды, а также узнать тип значения, который получится в результате.

Поскольку операция всегда производит известный тип, можно определить типы всех элементов стека и локальные переменные в любой момент в будущем, посмотрев на исходное состояние. Коллекция всей этой информации типов в любое предоставленное время называется состоянием типов стека; это то, что Java старается проанализировать перед запуском приложения. Язык Java ничего не знает о действительных значениях стека и элементах переменных в данное время; он только знает, что это за

элементы. Однако этой информации достаточно, чтобы задействовать правила безопасности и удостовериться в том, что объекты не управляются нелегально.

Чтобы сделать реальным анализ состояния типов стека, Java налагает дополнительное ограничение на то, как выполняются инструкции байт-кода Java: все пути к одной точке в коде должны достигаться одним и тем же состоянием типа.

Загрузчики классов

Язык аа добавляет второй уровень защиты при помощи загрузчиков классов. Загрузчик класса ответственен за доставку байт-кода для классов Java в интерпретатор. Каждое приложение, которое загружает классы из сети, должно использовать загрузчик класса, чтобы справиться с этой задачей.

После того как класс загружен и прошел через верификатор, он остановится связанным со своим загрузчиком классов. В результате классы эффективно секционируются в отдельные пространства имен, основанные на их происхождении. Когда загрузчик класса ссылается на другое имя класса, местоположение нового класса предоставляется исходным загрузчиком класса. Это означает, что классы, полученные из специфического источника, ограничены только взаимодействием только с другими классами, полученными из того же места. Например, веб-браузер с запущенным приложением Java может использовать загрузчик классов, чтобы создать отдельное пространство для всех классов, загруженных с определенного URL-адреса.

Поиск классов всегда начинается с тех, что встроены в систему Java. Эти классы загружаются из мест, определяемых путями к классу интерпретатора Java. Классы и пути к ним загружаются системой только однажды и не могут быть заменены. Это означает, что приложение не может заменить фундаментальную систему классов своей собственной версией, которая изменит их функциональность.

Диспетчеры безопасности

Диспетчер (менеджер) безопасности отвечает за принятие решений по безопасности на уровне приложения. Диспетчер (менеджер) безопасности является объектом, который может быть установлен приложением для ограничения доступа к ресурсам системы. К диспетчеру (менеджеру) безопасности обращаются каждый раз, когда приложение пытается получить доступ к таким элементам, как файловые системы,

сетевые порты, внешние процессы и система управления окнами; диспетчер (менеджер) безопасности может разрешить и запретить запрос.

Диспетчеры (менеджеры) безопасности, прежде всего, ценны для приложений, которые выполняют ненадежный код как часть своей нормальной работы.

Диспетчер (менеджер) безопасности работает совместно с инспектором доступа, что позволяет применять политику безопасности на высоком уровне редактированием декларативного файла политики безопасности. Политика доступа может быть простой или сложной как специфический гарант безопасности приложения. Иногда достаточно просто запретить доступ ко всем ресурсам или общим категориям услуг таким как файловая система или сеть. Но также возможно принимать сложные решения, основанные на информации высокого уровня. Например, веб-браузеры с запущенным приложением Java могут использовать политику доступа, которая позволяет пользователям определять, насколько можно доверять приложению или которая разрешает или запрещает доступ к специфическим ресурсам от случая к случаю. Конечно, это предполагает, что браузер может определить, какому приложению можно доверять.

Надежность работы диспетчера (менеджера) безопасности основана на защите, предоставленной на низких уровнях модели безопасности Java. Без гарантий, предоставленных верификатором и загрузчиком классов, утверждения о высоком уровне безопасности ресурсов системы являются бессмысленными, Безопасность, обеспеченная верификатором байт-кода Java, означает, что интерпретатор не может быть поврежден или разрушен и что Java-код должен использовать компоненты по назначению. Это в свою очередь означает, что загрузчик классов может гарантировать, что приложение использует системные классы ядра Java и эти классы являются единственным способом доступа к базовым системным ресурсам. С такими ограничениями можно централизовать контроль над этими ресурсами на высоком уровне, с диспетчером (менеджером) безопасности и определяемой пользователем политикой.

Инструменты для разработки Java-приложений

Java Development Kit (JDK) является одним из трех основных технологий, используемых в программировании на языке Java. К ним также относятся JVM (Java Virtual Machine) и JRE (Java Runtime Environment). Важно их различать, а также понимать, как они связаны (Рисунок 3):

- JVM отвечает за исполнение Java-программ, а также осуществляет контроль за безопасностью исполнения приложений;
- JRE – создает (компилирует) и запускает JVM;
- JDK позволяет разработчикам создавать программы, которые могут выполняться и запускаться посредством JVM и JRE;

JRE может использоваться как отдельный компонент для простого запуска уже готовой Java-программы, но он также является частью JDK. JDK требует JRE, потому что запуск программ является частью их разработки.

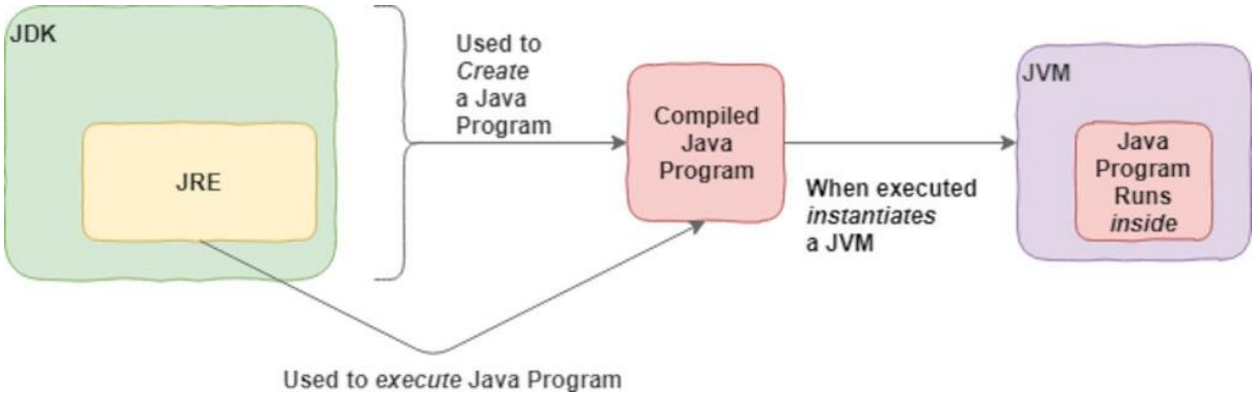

Рисунок 3 – Взаимосвязь трех основных технологий

Замечание
Установка JDK «тянет» за собой установку JRE и JVM.

Стоит отметить, что существуют разные реализации JDK, хотя все они используют один и тот же язык - Java. Две наиболее популярных реализации - Oracle JDK и OpenJDK. В чем их разница?

Oracle JDK всецело развивается компанией Oracle. OpenJDK же развивается как компанией Oracle, так и еще рядом компаний совместно.

Наибольшие различия с точки зрения лицензирования. Согласно лицензии Oracle JDK можно использовать бесплатно для персональных нужд, а также для разработки, тестирования и демонстрации приложений. В остальных случаях (например, для получения поддержки) необходима коммерческая лицензия в виде подписки. А OpenJDK полностью бесплатна.

В плане функционала, набора возможностей Oracle JDK и OpenJDK практически не должны отличаться. А вот в плане производительности отмечается, что Oracle JDK работает несколько быстрее, чем OpenJDK. Кроме того, некоторые разработчики отмечают, что OpenJDK чуть более глючная, а Oracle JDK более стабильная.

В целом рекомендуется для учебных целей использовать Oracle JDK, однако если будет использоваться OpenJDK, никаких проблем не должно возникнуть.

Загрузить и установить соответствующую версию JDK можно с с официального сайта Oracle: https://www.oracle.com/java/technologies/javase-downloads.html

Краткое описание установки JDK можно на сайте METANIT.COM (статья: https://metanit.com/java/tutorial/1.1.php).

Обзор online-компиляторов для Java

Тренд последних пяти лет - онлайн компиляторы. Традиционным offline компиляторам на смену приходят интерактивные online-сервисы. Теперь, для того чтобы выполнить программу, написанную на практически на любом языке программирования, вовсе не обязательно ставить на персональном компьютере соответствующий софт. Можно просто зайти на сайт, добавить или написать свой код и отправить его на исполнение.

Причины использовать онлайн компилятор:

- требуется быстро проверить некую идею или алгоритм;
- надо в интерактивную лекцию встроить онлайн пример кода;
- online-компилятор нужен эпизодически и нет смысла ставить его на компьютер;
- online-компиляторы часто позволяют сохранить ссылку на выполненный код и передать эту ссылку другому пользователю.

Недостатков в использовании online-компиляторов немного: требуется подключение к интернету; приходится дольше ждать результат компиляции и, самое главное, приходится довольствоваться той конфигураций компилятора, которая есть - нельзя добавить свои библиотеки или пакеты. Однако для учебных целей в начале семестра эти недостатки не существенны, зато у студентов нет никаких проблем с их инсталляцией.

В завершение приведем пару ссылок на очень простые и удобные компиляторы с возможность ввода своих данных.

Первым рассмотрим online-компилятор для Java на сайте «OnlineGDB» (https://www.onlinegdb.com/online_java_compiler , Рисунок 4):

Рисунок 4 – Среда разработки для Java-кода

Если осуществляется вход через URL: https://www.onlinegdb.com, то справа в верхнем углу в раскрывающемся списке следует выбрать язык Java (Рисунок 5). Обращает на себя то, что в принципе версия Java не указывается.

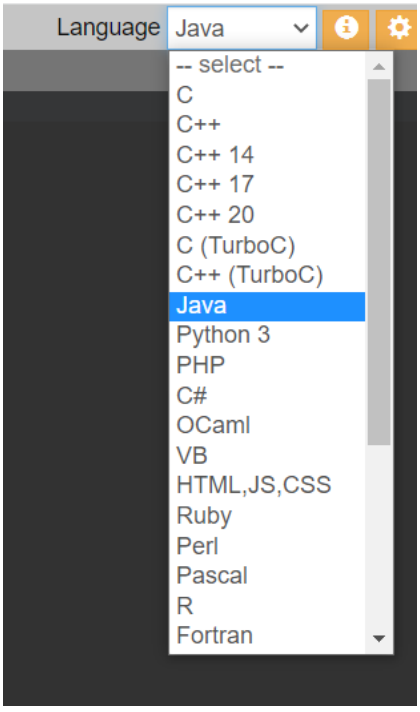

Рисунок 5 – Раскрывающийся список выбора языка программирования

Далее:

- с помощью стандартной комбинации клавиш Ctrl+V вставляем ранее набранный код в рабочее поле или с помощью клавиатуры набираем требуемую программу;

Замечание.

Для выполнения программы необходимо чтобы класс в котором расположен метод main() (точка входа в приложение) также имел название Main (с заглавной буквы).

- нажимаем кнопку Run и ждем результатов выполнения программы. Если будут обнаружены синтаксические ошибки, то они отобразятся во вкладке stderr, которая откроется автоматически в нижней части окна;

- необходимо исправить ошибки в соответствии с инструкциями компилятора и повторно нажать Run. Если ошибок не будет, то автоматически во окне Input появиться результат выполнения программы.

Замечание.

OnlineGDB - отличный и одновременно простой онлайн компилятор Java.

Вторым рассмотрим JDOODLE online-компилятор (https://www.jdoodle.com/online-java-compiler/, Рисунок 6). Сразу при входе обращает на себя внимание то, что класс содержащий метод main() произвольное наименование. Можете сменить его при необходимости. Входящие данные вам необходимо внести в окно Stdin Inputs (справа внизу) еще до запуска программы. Запуск на выполнение также осуществляется

кнопкой Execute. Внизу в черном окне Result можно увидеть либо список ошибок компиляции либо результат выполнения программы.

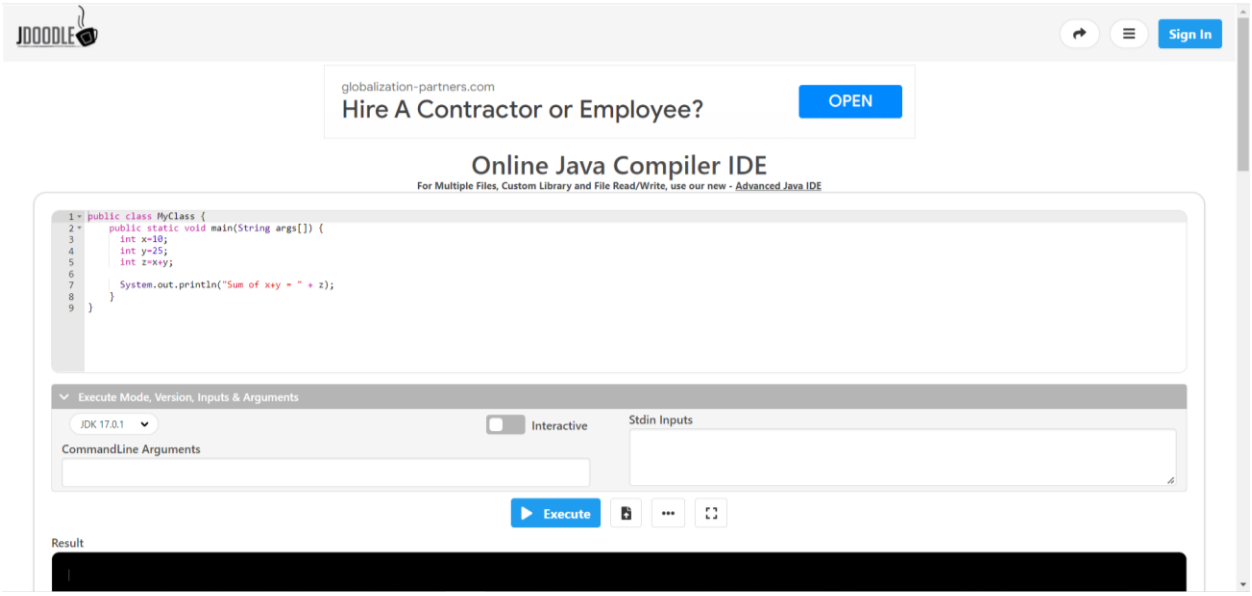

Рисунок 6 - JDOODLE online-компилятор

Также следует упомянуть online-компилятор CodingGround Online Java Compiler (https://www.tutorialspoint.com/compile_java_online.php);

Литература

1. Нимейер, П. Программирование на Java / П. Нимейер, Д. Леук - Москва: Эксимо, 2014. - 1216 с.
2. Блинов, И.Н. Java from EPAM / И.Н. Блинов, В.С. Романчик. - Минск: Четыре четверти, 2020. - 560 с.
3. Руководство по языку программирования Java/ METANIT.COM - [Электронный ресурс], URL: https://metanit.com/java/tutorial/ (дата обращения: 08.06.22)

Учебное издание

Кравчук Александр Степанович
Кравчук Анжелика Ивановна
Кремень Елена Васильевна

Язык Java.
Общие сведения

Учебные материалы
для студентов специальности 1-31 03 08
«Математика и информационные технологии
(по направлениям)»

В авторской редакции

Ответственный за выпуск *Е. В. Кремень*

Подписано в печать 22.12.2022. Формат 60×84/16. Бумага офсетная.
Усл. печ. л. 1,86. Уч.- изд. л. 1,65. Тираж 50 экз. Заказ

Белорусский государственный университет.
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 1/270 от 03.04.2014.
Пр. Независимости 4, 220030, Минск.

Отпечатано с оригинал-макета заказчика
на копировально-множительной технике
механико-математического факультета
Белорусского государственного университета.
Пр. Независимости 4, 220030, Минск.