

Лекция 1: Общие понятия о Java, история развития, основы использования

Java широко известна как новейший объектно-ориентированный язык, легкий в изучении и позволяющий создавать программы, которые могут исполняться на любой платформе без каких-либо доработок (*кроссплатформенность*). Еще с *Java* почему-то всегда связана тема кофе (изображения логотипов, названия продуктов и т.д.). Программисты могут добавить к этому описанию, что язык похож на упрощенный C или C++ с добавлением *garbage collector'a* - автоматического сборщика "мусора" (механизм освобождения памяти, которая больше не используется программой). Также известно, что *Java* ориентирована на Internet, и самое распространенное ее применение - небольшие программы, *апплеты*, которые запускаются в *браузере* и являются частью *HTML*-страниц.

Критики, в свою очередь, утверждают, что язык вовсе не так прост в применении, многие замечательные свойства лишь заявлены, а на самом деле не очень-то работают, а главное - программы на *Java* исполняются чрезвычайно медленно. Следовательно, это просто некая модная технология, которая только на время привлечет к себе внимание, а затем исчезнет, как и многие другие.

Однако некоторые факты не позволяют согласиться с такой оценкой. Во-первых, со времени официального объявления *Java* прошло достаточно много времени для "просто модной технологии". Во-вторых, конференция разработчиков *Java One*, которая впервые была организована в 1996 году, уже через год собрала более 10000 участников и стала крупнейшей конференцией по созданию программного обеспечения в мире (каждый следующий год число участников росло примерно на 5000). Специальная программа Sun, объединяющая разработчиков *Java* по всему миру, *Java Developer Connection*, также была запущена в 1996 году, через год она насчитывала более 100.000 разработчиков, а в 2000 году - более 1,5 миллионов. На сегодня число программистов на *Java* оценивается в 3 миллиона.

Было выпущено восемь основных версий языка, начиная с 1.0 в 1995 году. Последней, на сегодняшний день, является версия 8. Все версии и документацию к ним всегда можно было бесплатно получить на официальном web-сайте *Java* <http://java.sun.com/>. Один из первых продуктов для *Java* - *JDK 1.1* (средство разработки на *Java*) - в течение первых трех недель после объявления был загружен более 220.000 раз. Версия 1.4 была загружена более 2 миллионов раз за первые 5 месяцев. Практически все ведущие производители программного обеспечения лицензировали технологию *Java* и регулярно объявляют о выходе построенных на ней продуктов. Это и "голубой гигант" IBM, и создатель платформы Macintosh фирма Apple, и лидер в области реляционных БД Oracle, и даже главный конкурент фирмы Sun - корпорация Microsoft - лицензировала *Java* еще в марте 1996 года.

В следующем разделе описывается краткая история зарождения и развития идей, приведших к появлению *Java*, что поможет понять, чем на самом деле является эта технология, каковы ее свойства и отличительные черты, для чего она предназначена и откуда взялось такое разнообразие мнений о ней.

История развития Java

Теперь, когда за *Java* стояли не только несколько создателей, но еще и целая армия разработчиков, корпорация Sun имела возможность строить широкомасштабные планы развития технологии.

Браузеры

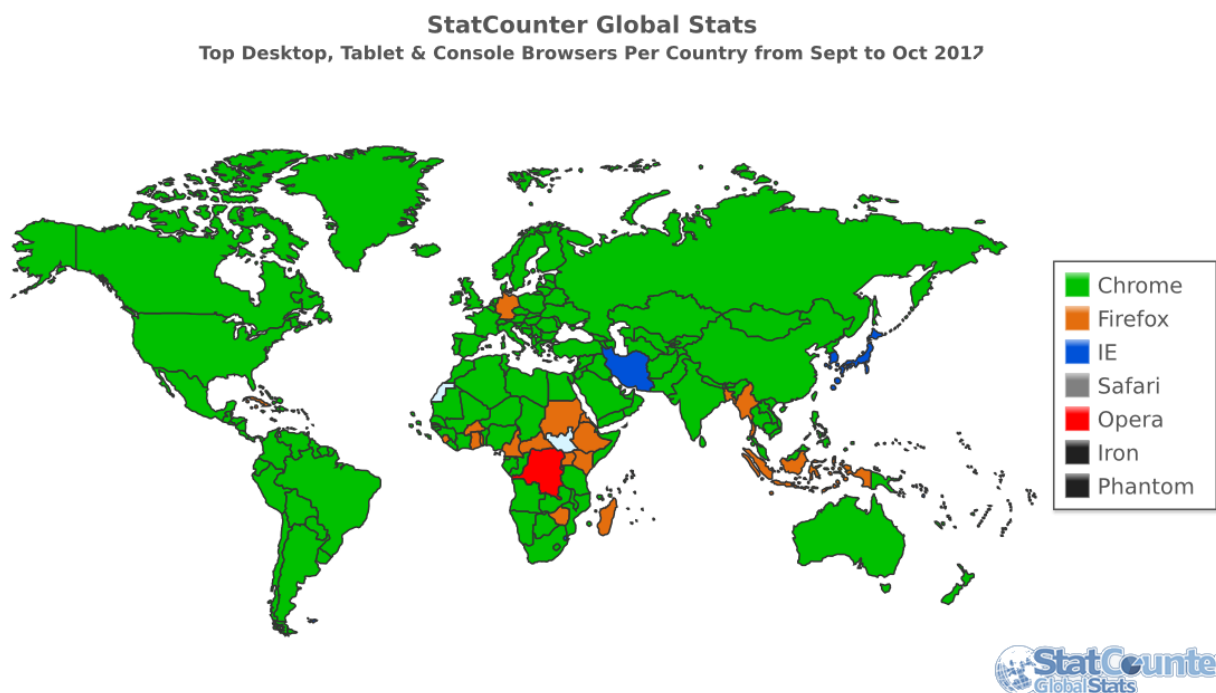
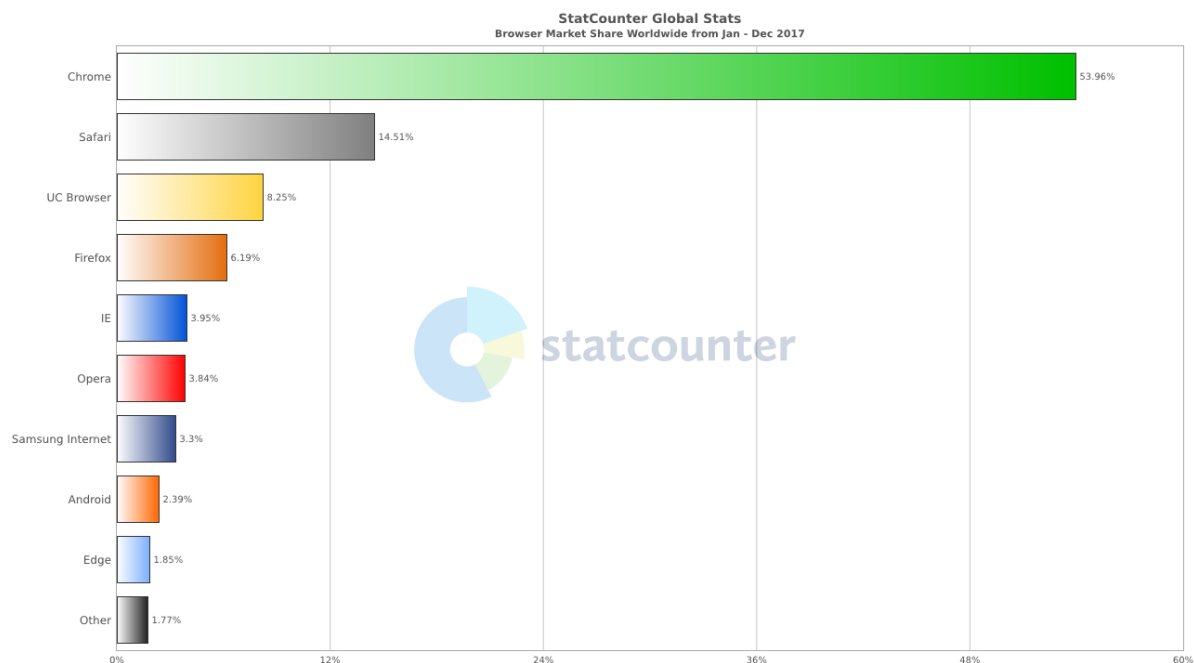
Конечно, основная линия развития оставалась связанной с *браузерами*. Хотя Internet только начинал наполняться все новыми технологиями, уже возникали проблемы совместимости. Под разными платформами работали настолько разные *браузеры*, что различались даже шрифты. В результате автор мог создать красивую аккуратную страницу, которая у клиента расплывалась.

С помощью *Java* web-страницу можно наполнить не только обычным текстом, но и динамическими элементами - простыми видеовставками типа вращающегося земного шара или Дьюка, машущего рукой (хотя сейчас такие задачи хорошо решает анимированный GIF, а в более сложных случаях - Macromedia Flash); интерактивными элементами типа вращающейся модели химической молекулы; бегущими строками, содержащими, например, биржевые индексы или прогноз погоды.

Но на самом деле *Java* – это больше, чем украшение *HTML*. Поскольку это полноценный язык программирования, с его помощью можно создать сложный пользовательский *интерфейс*. В самой первой версии *Java Development Kit* (средство разработки на *Java*) был пример *апплета*, представляющий простейшие электронные таблицы. Вскоре появился текстовый редактор, позволяющий менять стиль и цвет текста. Конечно, были игровые *апплеты*, обучающие, моделирующие физические и иные системы. Например, клиент, сделавший заказ в магазине или отправивший посылку почтой, получал возможность следить за доставкой через Internet.

В отличие от обычных программ, *апплеты* получили "в наследство" важное свойство *HTML*-страниц. Прочитав сегодня содержание страницы новостей, клиент не сохраняет ее на своем компьютере, а на следующий день читает обновленное содержание. Точно так же, скачав *апплет* и поработав с ним, можно удалить его, а в следующий раз получить более новую версию. Таким образом, программы появляются и исчезают с машины клиента безо всякого усилия, не требуются ни специальные знания, ни действия, и при этом автоматически поддерживаются самые последние версии.

С другой стороны, пользователь уже не привязан к своему основному рабочему месту, в любом Internet-кафе можно открыть нужную web-страницу и начать работу с привычными программами. И все это без каких-либо опасений подцепить вирус. Разработчиков очень заинтересовало, что их программы через день после выпуска могут увидеть пользователи всего мира, независимо от того, какой компьютер, операционную систему и *браузер* они используют. Хотя *браузер* на стороне клиента должен поддерживать *Java*, как уже говорилось, пользователям предлагался *HotJava*, доступный на любой платформе. Самый популярный в то время *браузер* Netscape Navigator, начиная с версии 2.0, также поддерживал *Java*. Однако, по состоянию на 1 января 2018, самый распространенный *браузер* – Google Chrome.



Компания Microsoft, добившись ошеломляющего успеха в области программного обеспечения для персональных компьютеров, стала (и в целом остается до сих пор) основным конкурентом в этой области для Sun, IBM, Netscape и других. Если в начале девяностых основные усилия Microsoft были направлены на операционную систему Windows и офисные приложения (MS Office), то в середине десятилетия стало очевидно, что пора всерьез заняться Internet. В начале 1995 года Билл Гейтс опубликовал "планы объявления войны" Netscape с целью занять такое же монопольное положение в WWW, как и в области операционных систем для персональных компьютеров. И когда вскоре Netscape подписала лицензионное соглашение с Sun, Microsoft оказалась в трудной ситуации.

Internet Explorer 2.0 не вызывал энтузиазма и никто не верил, что он может составить хоть сколько-нибудь заметную конкуренцию Netscape Navigator. А это значит, что новая версия IE 3.0 должна уметь все, что умеет только что вышедший NN 2.0. Поэтому 7 декабря 1995 года Microsoft объявляет о своем намерении лицензировать *Java*, а в марте 1996 года соглашение о лицензировании было подписано. Самая

крупная компания по производству программного обеспечения была вынуждена поддерживать своего, возможно, самого опасного конкурента.

Сейчас мы имеем возможность оглянуться назад и оценить последствия произошедших событий. Теперь уже очевидно, что Microsoft полностью удалось осуществить свой план. Если Netscape Navigator 3.x еще сохранял лидирующее положение, то Netscape 4.x уже начал уступать Internet Explorer 4.x. Версия NN 5.x так и не вышла, а NN 6.x стал очередным разочарованием для бывших поклонников "Навигатора". Версия 7.0 уже не занимала значительной доли рынка, в то время как Internet Explorer 5.0, 5.5 и 6.0 использовали более 95% пользователей.

Забавно, что многие ожесточенно обвиняли Microsoft в том, что компания боролась с Netscape "нерыночными" средствами. Однако сравним действия конкурентов. Среди многих шагов, предпринятых Microsoft, была и поддержка независимой организации W3C, которая тогда руководила разработкой нового стандарта *HTML 3*. Вначале компания Netscape считалась локомотивом индустрии, поскольку она постоянно развивала и модернизировала *HTML*, который изначально вообще-то не предназначался для графического оформления текста. Но Microsoft, вложив большое количество средств и человеческих ресурсов, смогла утвердить стандарты, которые отличались от уже реализованных в Netscape Navigator, причем отличия порой были чисто формальными. В результате оказалось, что страницы, созданные в соответствии с W3C-спецификациями, отображались в Navigator искаженно. Немаловажно и то, что NN необходимо было скачивать (пусть и бесплатно) и устанавливать вручную, а IE быстро стал встроенным компонентом Windows, готовым к использованию (и от которого, кстати, избавиться нельзя было принципиально).

А каким образом Netscape смог добиться лидирующего положения? В свое время подобными же методами компания пыталась (успешно, в конце концов) вытеснить с рынка NCSA Mosaic. Тогда *HTML* был не особенно богат интересными возможностями, а потому инновации, поддерживаемые Navigator, сразу привлекали внимание разработчиков и пользователей. Однако такие страницы некорректно отображались в Mosaic, что заставляло его пользователей задуматься о переходе к продуктам Netscape.

В результате в связи с забвением Netscape и его Navigator многие вздохнули с облегчением. Хотя, безусловно, потеря конкуренции на рынке и воцарение такого опасного монополиста, как Microsoft, никогда не идет на пользу конечным пользователям, однако многие устали от "войны стандартов", когда и без того небогатые возможности *HTML* приходилось изощренно подгонять таким образом, чтобы страницы выглядели одинаково в обоих *браузерах*.

Последнее, на чем стоит остановиться, - это язык *Java Script*, который также весьма распространен и который до сих пор многие связывают с *Java*, видимо, по причине схожести имен. Впрочем, некоторые общие черты у них действительно есть.

4 декабря 1995 года компании Netscape и Sun совместно объявляют новый "язык сценариев" (scripting language) *Java Script*. Как следует из пресс-релиза, это открытый *кроссплатформенный* объектный язык сценариев для корпоративных сетей и Internet. Код *Java Script* описывается прямо в *HTML*-тексте (хотя можно и подгружать его из отдельных файлов с расширением .js). Этот язык предназначен для создания приложений, которые связывают объекты и ресурсы на клиентской машине или на сервере. Таким образом, *Java Script*, с одной стороны, расширяет и дополняет *HTML*, а с другой стороны - дополняет *Java*. С помощью *Java* пишутся объекты-*апплеты*, которыми можно управлять через язык сценариев.

Общие свойства *Java Script* и *Java*:

- легкость в освоении. По этому параметру *Java Script* сравнивают с Visual Basic - чтобы использовать эти языки, опыт программирования не требуется;

- *кроссплатформенность*. Код *Java Script* выполняется *браузером*. Подразумевается, что *браузеры* на разных платформах должны обеспечивать одинаковую функциональность для страниц, использующих язык сценариев. Однако это выполняется примерно в той же степени, что и поддержка самого *HTML*, - различий все же очень много;
- открытость; спецификация языка открыта для использования и обсуждения сообществом разработчиков;
- все перечисленные свойства позволяют утверждать, что *Java Script* хорошо приспособлен для Internet-программирования;
- синтаксисы языков *Java Script* и *Java* очень похожи. Впрочем, они также довольно сильно напоминают язык C;
- язык *Java Script* прототипно-ориентированный (поддерживаются некоторые аспекты объектно-ориентированного подхода), позволяет использовать различные объекты, предоставляемые *браузером*;
- похожая история появления и развития. Оба языка были объявлены компаниями Sun и Netscape с интервалом в несколько месяцев. Вышедший вскоре после этого Netscape Navigator 2.0 поддерживал обе новые технологии. Возможно, само название *Java Script* было дано для того, чтобы воспользоваться популярностью *Java*, либо для того, чтобы еще больше расширить понятие "платформа *Java*". Вполне вероятно, что основную работу по разработке языка провела именно Netscape.

Несмотря на большое количество схожих характеристик, *Java* и *Java Script* - совершенно различные языки, и в первую очередь - по назначению. Если изначально *Java* позиционировался как язык для создания Internet-приложений (*апплетов*), то сейчас уже очевидно, что *Java* - это полноценный язык программирования. Что касается *Java Script*, то он полностью оправдывает свое название языка сценариев, оставаясь расширением *HTML*. Впрочем, расширением довольно мощным, так как любители этой технологии ухитряются создавать вполне серьезные приложения, такие как 3D-игры от первого лица (в сильно упрощенном режиме, естественно), хотя это скорее случай из области курьезов.

В заключение отметим, что код *Java Script*, исполняющийся на клиенте, оказывается доступен всем в открытом виде, что затрудняет защиту авторских прав. С другой стороны, из-за отсутствия полноценной поддержки объявления новых типов программы со сложной функциональностью зачастую оказываются слишком запутанными для того, чтобы ими могли воспользоваться другие.

Сетевые компьютеры

Когда стало понятно, что новая технология пользуется небывалым спросом, разработчикам захотелось укрепить и развить успех и распространенность *Java*. Для того чтобы *Java* не разделила судьбу NeWS (эта оконная система упоминалась в начале лекции, она не получила развития, проиграв X Window), компания Sun старалась наладить сотрудничество с независимыми фирмами для производства различных библиотек, средств разработчика, инструментов. 9 января 1996 года было сформировано новое подразделение *JavaSoft*, которое и занялось разработкой новых *Java*-технологий и продвижением их на рынок. Главная цель - появление все большего количества самых разных приложений, написанных на этой платформе. Например, 1 июля 1997 года было объявлено, что ученые NASA (National Aeronautics and Space Administration, государственная организация США, занимающаяся исследованием космоса) с помощью *Java-апплетов* управляют роботом, изучающим поверхность Марса ("*Java* помогает делать историю!").

Пора остановиться подробнее на том, почему по отношению к *Java* используется термин "платформа", чем *Java* отличается от обычного языка программирования.

Как правило, платформой называют сочетание аппаратной архитектуры ("железо"), которая определяется типом используемого процессора (Intel x86, Sun SPARC, PowerPC и др.), с операционной системой (MS Windows, Sun Solaris, Linux, Mac OS и др.). При написании программ разработчик всегда пользуется средствами целевой платформы для доступа к сети, поддержки потоков исполнения, работы с графическим пользовательским *интерфейсом* (GUI) и другими возможностями. Конечно, различные платформы, в силу технических, исторических и других причин, поддерживают различные *интерфейсы* (API, Application Programming Interface), а значит, и программа может исполняться только под той платформой, под которую она была написана.

Однако часто заказчикам требуется одна и та же функциональность, а платформы они используют разные. Задача портирования приложений стоит перед разработчиками давно. Редко удастся перенести сложную программу без существенной переделки, очень часто различные платформы по-разному поддерживают многие возможности (например, операционная система Mac OS традиционно использует однокнопочную мышь, в то время как Windows изначально рассчитана на двухкнопочную).

А значит, и языки программирования должны быть изначально ориентированы на какую-то конкретную платформу. Синтаксис и основные концепции легко распространить на любую систему (хотя это и не всегда эффективно), но библиотеки, компилятор и, естественно, бинарный исполняемый код специфичны для каждой платформы. Так было с самого начала эпохи компьютерных вычислений, а потому лишь немногие, действительно удачные программы поддерживались сразу на нескольких системах, что приводило к некоторой изоляции миров программного обеспечения для различных операционных систем.

Было бы странно, если бы с развитием компьютерной индустрии разработчики не попытались создать универсальную платформу, под которой могли работать все программы. Особенно такому шагу способствовало бурное развитие Глобальной сети Internet, которая объединила пользователей независимо от типа используемых процессоров и операционных систем. Именно поэтому создатели *Java* задумали разработать не просто еще один язык программирования, а универсальную платформу для исполнения приложений, тем более что изначально *OaK* создавался для различных бытовых приборов, от которых ждать совместимости не приходится.

Каким же образом можно "сгладить" различия и многообразие операционных систем? Способ не новый, но эффективный - с помощью виртуальной машины. Приложения на языке *Java* исполняются в специальной, универсальной среде, которая называется ***Java Virtual Machine***. JVM - это программа, которая пишется специально для каждой реальной платформы, чтобы, с одной стороны, скрыть все ее особенности, а с другой - предоставить единую среду исполнения для *Java*-приложений. Фирма Sun и ее партнеры создали JVM практически для всех современных операционных систем. Когда речь идет о *браузере* с поддержкой *Java*, подразумевается, что в нем имеется встроенная виртуальная машина.

Подробнее JVM рассматривается ниже, но необходимо сказать, что разработчики Sun приложили усилия, чтобы сделать эту машину вполне реальной, а не только виртуальной. 29 мая 1996 года объявляется операционная система *Java OS* (финальная версия выпущена в марте следующего года). Согласно пресс-релизу, это была "возможно, самая небольшая и быстрая операционная система, поддерживающая *Java*". Действительно, разработчики стремились к тому, чтобы обеспечить возможность исполнять *Java*-приложения на самом широком спектре устройств - *сетевые компьютеры*, карманные компьютеры (PDA), принтеры, игровые приставки, мобильные телефоны и т.д. Ожидалось, что *Java OS* будет реализована на

всех аппаратных платформах. Это было необходимо для изначальной цели создателей *Java* - легкость добавления новой функциональности и совместимости в любые электрические приборы, которыми пользуется современный потребитель.

Это был первый шаг, продвигающий платформу *Java* на один уровень вниз - на уровень операционных систем. Предполагалось сделать и следующий шаг - создать аппаратную архитектуру, центральный процессор, который бы напрямую выполнял инструкции *Java* безо всякой виртуальной машины. Устройство с такой реализацией стало бы полноценным *Java*-устройством.

Кроме бытовых приборов, компания Sun позиционировала данное решение и для компьютерной индустрии - *сетевые компьютеры* должны были заменить разнородные платформы персональных рабочих станций. Такой подход хорошо укладывался в основную концепцию Sun, выраженную в лозунге "Сеть — это компьютер". Возможности одного компьютера никогда не сравнятся с возможностями сети, объединяющей все ресурсы компании, а тем более - всего мира. Наверное, сегодня это уже очевидно, но во времена, когда WWW еще не опутала планету, идея была революционной.

Если же строить многофункциональную сеть, то к ее рабочим станциям предъявляются совсем другие требования - они не должны быть особенно мощными, вычислительные задачи можно переложить на серверы. Это даже более выгодно, так как позволяет централизовать поддержку и обновление программного обеспечения, а также не вынуждает сотрудников быть привязанными к своим рабочим местам. Достаточно войти с любого терминала в сеть, авторизоваться - и можно продолжать работу с того места, на котором она была оставлена. Это можно сделать в кабинете, зале для презентаций, кафе, в кресле самолета, дома - где угодно!

Кроме очевидных удобств, это начинание было с большим энтузиазмом поддержано индустрией и в силу того, что оно являлось сильнейшим оружием в борьбе с крупнейшим производителем программного обеспечения - Microsoft. Тогда (да и сейчас) самой распространенной платформой являлась операционная система Windows на базе процессоров Intel (с чьей-то легкой руки теперь многими называемая Wintel). Этим компаниям удалось создать замкнутый круг, гарантирующий успех, - все пользовались их платформой, так как под нее написано больше всего программ, что, в свою очередь, заставляло разработчиков создавать новые продукты именно для платформы Wintel. Поскольку корпорация Microsoft всегда очень агрессивно развивала свое преимущество в области персональных компьютеров (вспомним, как Netscape Navigator безнадежно проиграл конкуренцию MS Internet Explorer), это не могло не вызывать сильное беспокойство других представителей компьютерной индустрии. Понятно, что концепция *сетевых компьютеров* свела бы на нет преимущества Wintel в случае широкого распространения. Разработчики просто перестали бы задумываться, что находится внутри их рабочей станции, также как домашние пользователи не имеют представления, на каких микросхемах собран их мобильный телефон или видеомаягнитофон.

Мы уже рассказывали о том, как и почему Microsoft лицензировала *Java*, хотя, казалось бы, этот шаг лишь способствовал опасному распространению новой технологии, ведь Internet Explorer завоевывал все большую популярность. Однако вскоре разразился судебный скандал. 30 сентября 1997 года вышел новый IE 4.0, а уже 7 октября Sun объявила, что этот продукт не проходит тесты на соответствие со спецификацией виртуальной машины. 18 ноября Sun обращается в суд, чтобы запретить использование логотипа "Совместимый с *Java*" ("*Java compatible*") для MS IE 4.0. Оказалось, что разработчики Microsoft слегка "улучшили" язык *Java*, добавив несколько новых ключевых слов и библиотек. Не то что бы это были сверхмощные расширения, однако достаточно привлекательные для того, чтобы значительная часть разработчиков начала ее использовать. К счастью, в Sun быстро осознали всю степень опасности такого

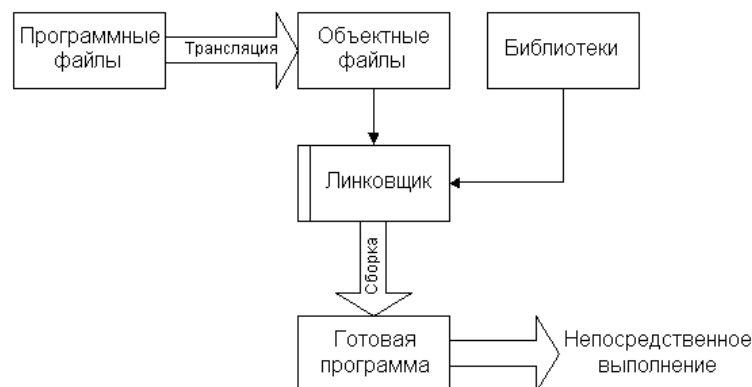
шага. *Java* могла потерять звание универсальной платформы, для которой верен знаменитый девиз "Write once, run everywhere" ("Написано однажды, работает везде"). В таком случае она утратила бы основу своего успеха, превратившись всего лишь в "еще один язык программирования".

Компании Sun удалось отстоять свою технологию. 24 марта 1998 года суд согласился с требованиями компании (конечно, это было только предварительное решение, дело завершилось лишь 23 января 2001 года - Sun получил компенсацию в 20 миллионов долларов и добился выполнения лицензионного соглашения), а уже 12 мая Sun снова выступает с требованием обязать Microsoft включить полноценную версию *Java* в Windows 98 и другие программные продукты. Sun выпустил специальный продукт *Java Plug-in*, который встраивается в MS IE и NN, позволяя им исполнять *апплеты* на основе *Java* самых последних версий, причем полное соответствие спецификациям гарантируется (первоначально продукт назывался *Java Activator* и впервые был объявлен 10 декабря 1997 года).

Что же касается *сетевых компьютеров* и *Java OS*, то, увы, они пока не нашли своих потребителей. Видимо, обычные персональные рабочие станции в совокупности с *JVM* требуют гораздо меньше технологических и маркетинговых усилий и при этом вполне успешно справляются с прикладными задачами. А *Java*, в свою очередь, стала позиционироваться для создания сложных серверных приложений.

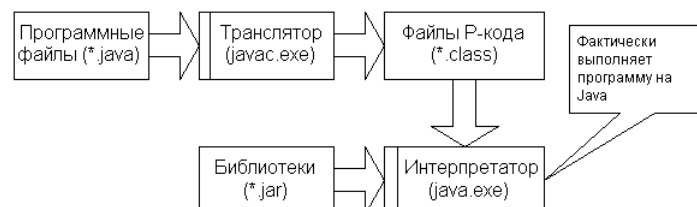
Жизненный цикл программы на Java

Под жизненным циклом мы будем понимать процесс, необходимый для создания работающего приложения. Для программ на Java он отличается от жизненного цикла программ на других языках программирования. Типичная картина жизненного цикла для большинства языков программирования выглядит примерно так.



Примерно такая схема действует в случае использования таких популярных языков как C++ или VB.

Для Java картина иная.



Из рисунка видно, что исходная Java-программа должна быть в файле с расширением **java**. Программа транслируется в байт-код компилятором **javac.exe**. Оттранслированная в байт-код программа имеет расширение **class**. Для запуска программы нужно вызвать интерпретатор **java.exe**, указав в параметрах вызова, какую программу ему следует выполнять. Кроме того, ему нужно указать, какие библиотеки нужно использовать при выполнении программы. Библиотеки размещены в файлах с

расширением **jar** (в предыдущих версиях SDK использовались файлы ***.zip** и некоторые библиотеки все еще в таких файлах).

Платформа Java

Итак, *Java* обладает длинной и непростой историей развития, однако настало время рассмотреть, что же получилось у создателей, какими свойствами обладает данная технология.

Самое широко известное и, в тоже время, вызывающее самые бурные споры, свойство — много-или *кроссплатформенность*. Уже говорилось, что оно достигается за счет использования виртуальной машины *JVM*, которая является обычной программой, исполняемой операционной системой и предоставляющей *Java*-приложениям все необходимые возможности. Поскольку все параметры *JVM* специфицированы, то остается единственная задача - реализовать виртуальные машины на всех существующих и используемых платформах.

Наличие виртуальной машины определяет многие свойства *Java*, однако сейчас остановимся на следующем вопросе - является *Java* языком компилируемым или интерпретируемым? На самом деле, используются оба подхода.

Исходный код любой программы на языке *Java* представляется обычными текстовыми файлами, которые могут быть созданы в любом текстовом редакторе или специализированном средстве разработки и имеют расширение `.java`. Эти файлы подаются на вход *Java*-компилятора, который транслирует их в специальный *Java байт-код*. Именно этот компактный и эффективный набор инструкций поддерживается *JVM* и является неотъемлемой частью платформы *Java*.

Результат работы компилятора сохраняется в бинарных файлах с расширением `.class`. *Java*-приложение, состоящее из таких файлов, подается на вход виртуальной машине, которая начинает их исполнять, или интерпретировать, так как сама является программой.

Многие разработчики поначалу жестко критиковали смелый лозунг Sun "Write once, run everywhere", обнаруживая все больше и больше несоответствий и нестыковок на различных платформах. Однако надо признать, что они просто были слишком нетерпеливы. *Java* только появилась на свет, а первые версии спецификаций были недостаточно исчерпывающими.

Очень скоро специалисты Sun пришли к выводу, что просто свободно публиковать спецификации (что уже делалось задолго до *Java*) недостаточно. Необходимо еще и создавать специальные процедуры проверки новых продуктов на соответствие стандартам. Первый такой тест для *JVM* содержал всего около 600 проверок, через год их число выросло до десяти тысяч и с тех пор все время увеличивается (именно его в свое время не смог пройти MS IE 4.0). Безусловно, авторы виртуальных машин все время совершенствовали их, устраняя ошибки и оптимизируя работу. Все-таки любая, даже очень хорошо задуманная технология требует времени для создания высококачественной реализации. Аналогичный путь развития сейчас проходит *Java 2 Micro Edition (J2ME)*, но об этом позже.

Следующим по важности свойством является объектная ориентированность *Java*, что всегда упоминается во всех статьях и пресс-релизах. Сам объектно-ориентированный подход (ООП) рассматривается в следующей лекции, однако важно подчеркнуть, что в *Java* практически все реализовано в виде объектов - потоки выполнения (threads) и потоки данных (streams), работа с сетью, работа с изображениями, с пользовательским *интерфейсом*, обработка ошибок и т.д. В конце концов, любое приложение на *Java* - это набор классов, описывающих новые типы объектов.

Подробное рассмотрение объектной модели *Java* проводится на протяжении всего курса, однако обозначим основные особенности. Прежде всего, создатели отказались от множественного наследования.

Было решено, что оно слишком усложняет и запутывает программы. В языке используется альтернативный подход - специальный тип "*интерфейс*". Он подробно рассматривается в соответствующей лекции.

Далее, в *Java* применяется *строгая типизация*. Это означает, что любая переменная и любое выражение имеет тип, известный уже на момент компиляции. Такой подход применен для упрощения выявления проблем, ведь компилятор сразу сообщает об ошибках и указывает их расположение в коде. Поиск же исключительных ситуаций (exceptions - так в *Java* называются некорректные ситуации) во время исполнения программы (runtime) потребует сложного тестирования, при этом причина дефекта может обнаружиться совсем в другом классе. Таким образом, нужно прикладывать дополнительные усилия при написании кода, зато существенно повышается его надежность (а это одна из основополагающих целей, для которых и создавался новый язык).

В *Java* существует всего 8 типов данных, которые не являются объектами. Они были определены с самой первой версии и никогда не менялись. Это пять целочисленных типов: byte, short, int, long, а также к ним относят символьный char. Затем два дробных типа float и double и, наконец, булевский тип boolean. Такие *типы* называются *простые*, или *примитивные* (от английского *primitive*), и они подробно рассматриваются в лекции, посвященной типам данных. Все остальные *типы* - *объектные* или *ссылочные* (англ. **reference**).

Синтаксис *Java* почему-то многих ввел в заблуждение. Он действительно создан на основе синтаксиса языков C/C++, так что если посмотреть на исходный код программ, написанных на этих языках и на *Java*, то не сразу удастся понять, какая из них на каком языке написана. Это почему-то дало многим повод думать, что *Java* - это упрощенный C++ с дополнительными возможностями, такими как *garbage collector*. Автоматический сборщик мусора (*garbage collector*) мы рассмотрим чуть ниже, но считать, что *Java* такой же язык, как и C++, - большое заблуждение.

Конечно, разрабатывая новую технологию, авторы *Java* опирались на широко распространенный язык программирования по целому ряду причин. Во-первых, они сами на тот момент считали C++ своим основным инструментом. Во-вторых, зачем придумывать что-то новое, когда есть вполне подходящее старое? Наконец, очевидно, что незнакомый синтаксис отпугнет разработчиков и существенно осложнит внедрение нового языка, а ведь *Java* должна была максимально быстро получить широкое распространение. Поэтому синтаксис был лишь слегка упрощен, чтобы избежать слишком запутанных конструкций.

Но, как уже говорилось, C++ принципиально не годился для новых задач, которые поставили себе разработчики из компании Sun, поэтому модель *Java* была построена заново, причем в соответствии с совсем другими целями. Дальнейшие лекции будут постепенно раскрывать конкретные различия.

Что же касается объектной модели, то она скорее была построена по образцу таких языков, как Smalltalk от IBM, или разработанный еще в 60-е годы в Норвежском Вычислительном Центре язык Simula, на который ссылается сам создатель *Java* Джеймс Гослинг.

Другое немаловажное свойство *Java* - легкость в освоении и разработке - также получило неоднозначную оценку. Действительно, авторы потрудились избавить программистов от наиболее распространенных ошибок, которые порой допускают даже опытные разработчики на C/C++. И первое место здесь занимает работа с памятью.

В *Java* с самого начала был введен *механизм автоматической сборки мусора* (от английского *garbage collector*). Предположим, программа создает некоторый объект, работает с ним, а дальше наступает момент, когда он больше уже не нужен. Необходимо освободить занимаемую память, чтобы не мешать операционной системе нормально функционировать. В C/C++ это необходимо делать явным образом из

программы. Очевидно, что при таком подходе существует две опасности - либо удалить объект, который еще кому-то необходим (и если к нему действительно произойдет обращение, то возникнет ошибка), либо не удалять объект, ставший ненужным, а это означает утечку памяти, то есть программа начинает потреблять все большее количество оперативной памяти.

При разработке на *Java* программист вообще не думает об освобождении памяти. Виртуальная машина сама подсчитывает количество ссылок на каждый объект, и если оно становится равным нулю, то такой объект помечается для обработки *garbage collector*. Таким образом, программист должен следить лишь за тем, чтобы не оставалось ссылок на ненужные объекты. Сборщик мусора - это фоновый поток исполнения, который регулярно просматривает существующие объекты и удаляет уже не нужные. Из программы никак нельзя повлиять на работу *garbage collector*, можно только явно инициировать его очередной проход с помощью стандартной функции. Ясно, что это существенно упрощает разработку программ, особенно для начинающих программистов.

Однако опытные разработчики были недовольны тем, что они не могут полностью контролировать все, что происходит с их системой. Нет точной информации, когда именно будет удален объект, ставший ненужным, когда начнет работать (а значит, и занимать системные ресурсы) поток сборщика мусора и т.д. Но, при всем уважении к опыту таких программистов, необходимо отметить, что подавляющее количество сбоев программ, написанных на C/C++, приходится именно на некорректную работу с памятью, причем порой это случается даже с широко распространенными продуктами весьма серьезных компаний.

Кроме того, особый упор делался на легкость освоения новой технологии. Как уже было сказано, ожидалось (и эти ожидания оправдались, в подтверждение правильности выбранного пути!), что *Java* должна получить максимально широкое применение, даже в тех компаниях, где никогда до этого не занимались программированием на таком уровне (бытовая техника типа тостеров и кофеварок, создание игр и других приложений для сотовых телефонов и т.д.). Был и целый ряд других соображений. Продукты для обычных пользователей, а не профессиональных программистов, должны быть особенно надежными. Internet стал Всемирной Сетью, поскольку появились непрофессиональные пользователи, а возможность создавать *апплеты* для них не менее привлекательна. Им требовался простой инструмент для создания надежных приложений.

Наконец, Internet-бум 90-х годов набирал обороты и выдвигал новые, более жесткие требования к срокам разработки. Многолетние проекты, которые были в прошлом обычным делом, перестали отвечать потребностям заказчиков, новые системы надо было создавать максимум за год, а то и за считанные месяцы.

Кроме введения *garbage collector*, были предприняты и другие шаги для облегчения разработки. Некоторые из них уже упоминались - отказ от множественного наследования, упрощение синтаксиса и др. Возможность создания многопоточных приложений была реализована в первой же версии *Java* (исследования показали, что это очень удобно для пользователей, а существующие стандарты опираются на телетайпные системы, которые устарели много лет назад). Другие особенности будут рассмотрены в следующих лекциях. Однако то, что создание и поддержка систем действительно проще на *Java*, чем на C/C++, давно является общепризнанным фактом. Впрочем, все-таки эти языки созданы для разных целей, и каждый имеет свои неоспоримые преимущества.

Следующее важное свойство *Java* - *безопасность*. Изначальная нацеленность на распределенные приложения, и в особенности решение исполнять *апплеты* на клиентской машине, сделали вопрос защиты одним из самых приоритетных. При работе любой виртуальной машины *Java* действует целый комплекс мер. Далее приводится лишь краткое описание некоторых из них.

Во-первых, это правила работы с памятью. Уже говорилось, что очистка памяти производится автоматически. Резервирование ее также определяется *JVM*, а не компилятором, или явным образом из программы, разработчик может лишь указать, что он хочет создать еще один новый объект. Указатели по физическим адресам отсутствуют принципиально.

Во-вторых, наличие виртуальной машины-интерпретатора значительно облегчает отсечение опасного кода на каждом этапе работы. Сначала байт-код загружается в систему, как правило, в виде class-файлов. *JVM* тщательно проверяет, все ли они подчиняются общим правилам *безопасности Java* и не созданы ли злоумышленниками с помощью каких-то других средств (и не искажены ли при передаче). Затем, во время исполнения программы, интерпретатор легко может проверить каждое действие на допустимость. Возможности классов, которые были загружены с локального диска или по сети, существенно различаются (пользователь легко может назначать или отменять конкретные права). Например, *апплеты* по умолчанию никогда не получают доступ к локальной файловой системе. Такие встроенные ограничения есть во всех стандартных библиотеках *Java*.

Наконец, существует механизм подписания *апплетов* и других приложений, загружаемых по сети. Специальный сертификат гарантирует, что пользователь получил код именно в том виде, в каком его выпустил производитель. Это, конечно, не дает дополнительных средств защиты, но позволяет клиенту либо отказаться от работы с приложениями ненадежных производителей, либо сразу увидеть, что в программу внесены неавторизованные изменения. В худшем случае он знает, кто ответственен за причиненный ущерб.

Совокупность описанных свойств *Java* позволяет утверждать, что язык весьма приспособлен для разработки Internet- и интранет (внутренние сети корпораций)-приложений.

Наконец, важная отличительная особенность *Java* - это его *динамичность*. Язык очень удачно задуман, в его развитии участвуют сотни тысяч разработчиков и многие крупные компании. Основные этапы этого развития кратко освещены в следующем разделе.

Итак, подведем итоги. *Java*-платформа обладает следующими преимуществами:

- переносимость, или *кроссплатформенность*;
- объектная ориентированность, создана эффективная объектная модель;
- привычный синтаксис C/C++;
- встроенная и прозрачная модель *безопасности*;
- ориентация на Internet-задачи, сетевые распределенные приложения;
- *динамичность*, легкость развития и добавления новых возможностей;
- простота освоения.

Но не следует считать, что более легкое освоение означает, что изучать язык не нужно вовсе. Чтобы писать действительно хорошие программы, создавать большие сложные системы, необходимо четкое понимание всех базовых концепций *Java* и используемых библиотек. Именно этому и посвящен данный курс.

Основные версии и продукты Java

Сразу оговоримся, что под продуктами здесь понимаются программные решения от компании Sun, являющиеся "образцами реализации" (reference implementation).

Итак, впервые *Java* была объявлена 23 мая 1995 года. Основными продуктами, доступными на тот момент в виде бета-версий, были:

- *Java language specification, JLS*, спецификация языка *Java* (описывающая лексику, типы данных, основные конструкции и т.д.);
- спецификация *JVM*;
- *Java Development Kit, JDK* - средство разработчика, состоящее в основном из утилит, стандартных библиотек классов и демонстрационных примеров.

Спецификация языка была составлена настолько удачно, что практически без изменений используется и по сей день. Конечно, было внесено большое количество уточнений, более подробных описаний, были добавлены и некоторые новые возможности (например, объявление внутренних классов), однако основные концепции остаются неизменными. Данный курс в большой степени опирается именно на спецификацию языка.

Спецификация *JVM* предназначена в первую очередь для создателей виртуальных машин, а потому практически не используется *Java*-программистами.

JDK долгое время было базовым средством разработки приложений. Оно не содержит никаких текстовых редакторов, а оперирует только уже существующими *Java*-файлами. Компилятор представлен утилитой *javac* (*java compiler*). Виртуальная машина реализована программой *java*. Для тестовых запусков *апплетов* существует специальная утилита *appletviewer*. Наконец, для автоматической генерации документации на основе исходного кода прилагается средство *javadoc*.

Первая версия содержала всего 8 стандартных библиотек:

- *java.lang* - базовые классы, необходимые для работы любого приложения (название - сокращение от language);
- *java.util* - многие полезные вспомогательные классы;
- *java.applet* - классы для создания *апплетов*;
- *java.awt, java.awt.peer* - библиотека для создания графического *интерфейса* пользователя (*GUI*), называется *Abstract Window Toolkit, AWT*, подробно описывается в лекции 11;
- *java.awt.image* - дополнительные классы для работы с изображениями;
- *java.io* - работа с потоками данных (*streams*) и с файлами;
- *java.net* - работа с сетью.

Таким образом, все библиотеки начинаются с *java*, именно они являются стандартными. Все остальные (начинающиеся с *com, org* и др.) могут меняться в любой версии без поддержки совместимости.

Финальная версия *JDK 1.0* была выпущена в январе 1996 года.

Сразу поясним систему именования версий. Обозначение версии состоит из трех цифр. Первой пока всегда стоит 1. Это означает, что поддерживается полная совместимость между всеми версиями 1.x.x. То есть программа, написанная на более старом *JDK*, всегда успешно выполнится на более новом. По возможности соблюдается и обратная совместимость - если программа откомпилирована более новым *JDK*, а никакие новые библиотеки не использовались, то в большинстве случаев старые виртуальные машины смогут выполнить такой код.

Вторая цифра изменилась от 0 до 8. Начиная с 1.5, версии именуются просто цифрой, например, *Java 8*. В каждой версии происходило существенное расширение стандартных библиотек (например, 212, 504, 1781, 2130 и 2738 - количество классов и *интерфейсов* с 1.0 по 1.4), а также добавлялись некоторые новые возможности в сам язык. Менялись и утилиты, входящие в *JDK*.

Наконец, третья цифра означает развитие одной версии. В языке или библиотеках ничего не меняется, лишь устраняются ошибки, производится оптимизация, могут изменяться аргументы утилит.

Хотя с развитием версии 1.x ничего не удаляется, конечно, какие-то функции или классы устаревают. Они объявляются *deprecated*, и хотя они будут поддерживаться до объявления 2.0, пользоваться ими не рекомендуется.

Вместе с первым успехом *JDK 1.0* подспела и критика. Основные недостатки, обнаруженные разработчиками, были следующими. Во-первых, конечно, производительность. Первая виртуальная машина работала очень медленно. Это связано с тем, что *JVM*, по сути, представляет собой интерпретатор, который работает всегда медленнее, чем исполняется откомпилированный код. Однако успешная оптимизация, устранившая этот недостаток, была еще впереди. Также отмечались довольно бедные возможности *AWT*, отсутствие работы с базами данных и другие.

В декабре 1996 года объявляется новая версия *JDK 1.1*, сразу выкладывается для свободного доступа бета-версия. В феврале 1997 года выходит финальная версия. Что было добавлено в новом выпуске *Java*?

Конечно, особое внимание было уделено производительности. Многие части виртуальной машины были оптимизированы и переписаны с использованием *Assembler*, а не *C*, как до этого. Кроме того, с октября 1996 года Sun развивает новый продукт - *Just-In-Time* компилятор, *JIT*. Его задача - транслировать *Java байт-код* программы в "родной" код операционной системы. Таким образом, время запуска программы увеличивается, но зато выполнение может ускоряться в некоторых случаях до 50 раз! С июля 1997 года появляется реализация под *Windows* и *JIT* стандартно входит в *JDK* с возможностью отключения.

Были добавлены многие новые важные возможности. *JavaBeans* - технология, объявленная ещё в 1996 году, позволяет создавать визуальные компоненты, которые легко интегрируются в визуальные средства разработки. *JDBC* (*Java DataBase Connectivity*) обеспечивает доступ к базам данных. *RMI* (*Remote Method Invocation*) позволяет легко создавать распределенные приложения. Были усовершенствованы поддержка национальных языков и система безопасности.

За первые три недели *JDK 1.1* был скачан более 220.000 раз, менее чем через год - более двух миллионов раз. На данный момент версия 1.1 считается полностью устаревшей и ее развитие остановилось на 1.1.8. Однако из-за того, что браузер *MS IE* ещё шесть лет назад поддерживал только эту версию, она продолжает использоваться для написания небольших апплетов.

Кроме того, с 11 марта 1997 года компания Sun начала предлагать *Java Runtime Environment, JRE* (среду выполнения *Java*). По сути дела, это минимальная реализация виртуальной машины, необходимая для исполнения *Java*-приложений, без компилятора и других средств разработки. Если пользователь хочет только запускать программы, это именно то, что ему нужно.

Как видно, самым главным недостатком осталась слабая поддержка графического интерфейса пользователя (*GUI*). В декабре 1996 года компании Sun и Netscape объявляют новую библиотеку *IFC* (*Internet Foundation Classes*), разработанную Netscape полностью на *Java* и предназначенную как раз для создания сложного оконного интерфейса. В апреле 1997 года объявляется, что компании планируют объединить технологии *AWT* от Sun и *IFC* от Netscape для создания нового продукта *Java Foundation Classes, JFC*, в который должны войти:

- усовершенствованный оконный интерфейс, который получил особое название - *Swing*;
- реализация *Drag-and-Drop*;
- поддержка 2D-графики, более удобная работа с изображениями;
- *Accessibility API* для пользователей с ограниченными возможностями и другие функции.

Компания IBM также поддержала разработку новой технологии. В июле 1997 года стала доступна первая версия *JFC*. Первоначально библиотеки назывались, например, `com.sun.java.swing` для компонентов *Swing*. В марте 1998 года вышла финальная версия этой технологии. За полгода продукт был скачан более 500.000 раз.

Выход следующей версии *Java 1.2* много раз откладывался, но в итоге она настолько превзошла предыдущую 1.1, что ее и все последующие версии начали называть платформой *Java 2* (хотя номера, конечно, по-прежнему отсчитывались как 1.x.x, см. выше описание правил нумерации). Первая бета-версия стала доступной в декабре 1997 года, а финальная версия была выпущена 8 декабря 1998 года, и за первые восемь месяцев ее скачали более миллиона раз.

Список появившихся возможностей очень широк, поэтому перечислим наиболее значимые из них:

- существенно переработанная модель *безопасности*, введены понятия политики (policy) и разрешения (permission);
- *JFC* стал стандартной частью *JDK*, причем библиотеки стали называться, например, `javax.swing` для *Swing* (название `javax` указывает, что до этого библиотека считалась расширением *Java*);
- полностью переработанная библиотека коллекций (collection framework) - классов для хранения набора объектов;
- *Java Plug-in* был включен в *JDK*;
- улучшения в производительности, глобализации (независимости от особенностей разных платформ и стран), защита от "проблемы-2000".

С февраля 1999 года исходный код самой *JVM* был открыт для бесплатного доступа всем желающим.

Самое же существенное изменение произошло 15 июня 1999 года, спустя полгода после выхода *JDK 1.2*. На конференции разработчиков *JavaOne* компания Sun объявила о разделении развития платформы *Java 2* на три направления:

- *Java 2 Platform, Standard Edition (J2SE)*;
- *Java 2 Platform, Enterprise Edition (J2EE)*;
- *Java 2 Platform, Micro Edition (J2ME)*.

На самом деле, подобная классификация уже давно назрела, в частности, различных спецификаций и библиотек насчитывалось несколько десятков, а потому они нуждались в четкой структуризации. Кроме того, такое разделение облегчало развитие и продвижение на рынок технологии *Java*.

J2SE предназначена для использования на рабочих станциях и персональных компьютерах. Standard Edition - основа технологии *Java* и прямое развитие *JDK* (средство разработчика было переименовано в `j2sdk`).

J2EE содержит все необходимое для создания сложных, высоконадежных, распределенных серверных приложений. Условно можно сказать, что Enterprise Edition - это набор мощных библиотек (например, Enterprise *Java Beans*, EJB) и пример реализации платформы (сервера приложений, Application Server), которая их поддерживает. Работа такой платформы всегда опирается на `j2sdk`.

J2ME является усечением Standard Edition, чтобы удовлетворять жестким аппаратным требованиям небольших устройств, таких как карманные компьютеры и сотовые телефоны.

Далее развитие этих технологий происходит разными темпами. Если *J2SE* уже была доступна более полугода, то финальная версия *J2EE* вышла лишь в декабре 1999 года.

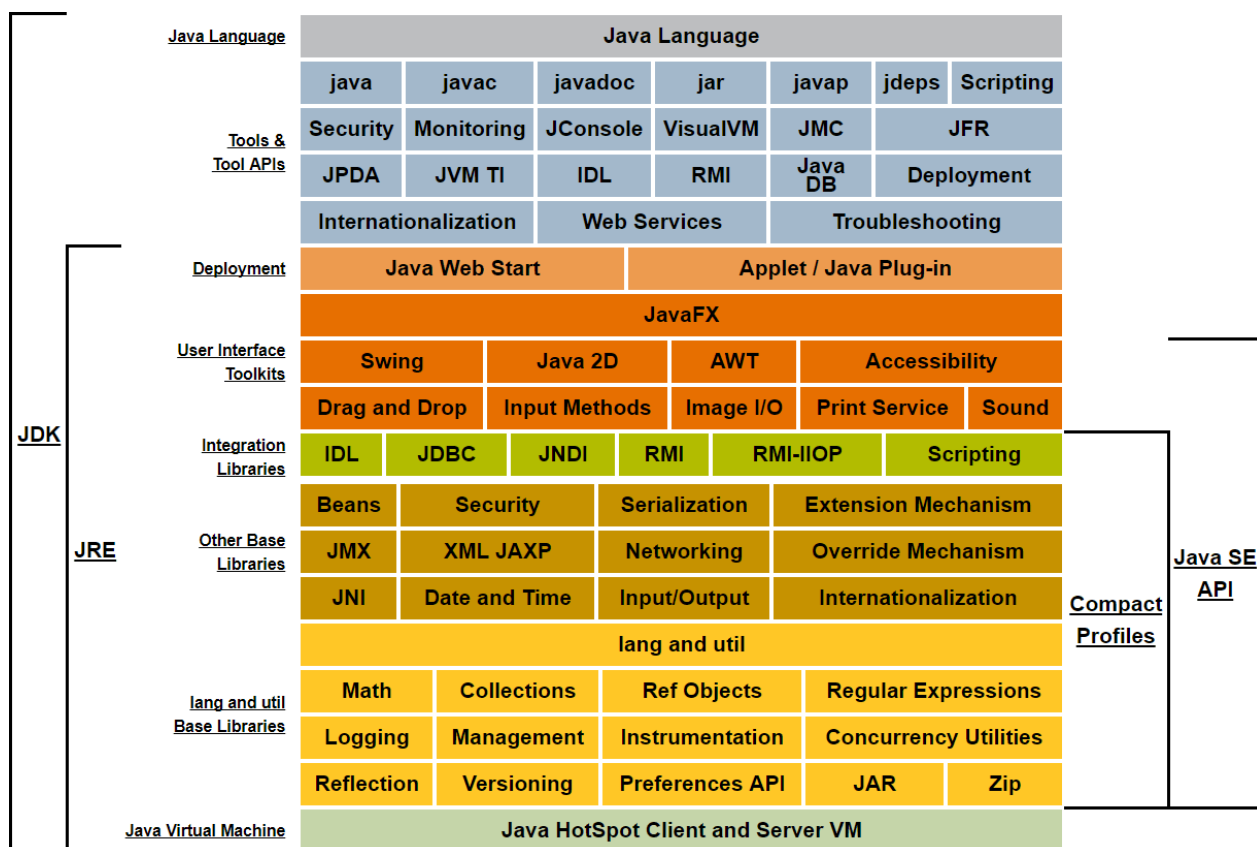
Тем временем борьба за производительность продолжалась, и Sun пытался еще больше оптимизировать виртуальную машину. В марте 1999 года объявляется новый продукт - высокоскоростная платформа (engine) *Java HotSpot*. Была оптимизирована работа с потоками исполнения, существенно переработаны алгоритмы автоматического сборщика мусора (*garbage collector*) и многое другое. Ускорение действительно было очень существенным, всегда заметное невооруженным взглядом за несколько минут работы с *Java*-приложением.

Новая платформа может работать в двух режимах - клиентском и серверном. Режимы различались настройками и другими оптимизирующими алгоритмами. По умолчанию работа идет в клиентском режиме.

Развитие HotSpot продолжалось более года, пока в начале мая 2000 года высокопроизводительная *JVM* не вошла в состав новой версии *J2SE*. В эту версию было внесено еще множество улучшений и исправлений, но именно прогресс в скорости работы стал ключевым изменением нового j2sdk 1.

Версия *J2SE* 1.4 вышла в феврале 2002 года. Она была разработана для более полной поддержки web-сервисов (web services). Поэтому основные изменения коснулись работы с *XML* (Extensible Markup Language). Другое революционное добавление - выражение `assert`, позволяющее в отладочном режиме проверять верность условий, что должно серьезно упростить разработку сложных приложений. Наконец, были добавлены классы для работы с регулярными выражениями.

В заключение для демонстрации уровня развития Standard Edition приведем стандартную диаграмму, описывающую все составляющие технологии.



Заключение

В этой лекции мы рассказали о том, какая непростая ситуация сложилась в корпорации Sun в эпоху развития персональных компьютеров в конце 1990 года. Патрик Нотон в своем письме сумел выявить истинные причины такого положения и обозначить истинные цели для создания успешного продукта. Благодаря этому при поддержке Джеймса Гослинга начался проект *Green*. Одним из продуктов, созданных в

рамках этого проекта, стала совершенно новая платформа *OaK*. Для ее продвижения Sun учредила дочернюю компанию FirstPerson, но настоящий успех пришел, когда платформу, переименовав в *Java*, сориентировали на применение в Internet. Первым примером *Java*-приложений стали *апплеты*, запускаемые при помощи специально созданного *браузера HotJava*. Наконец, после почти четырехлетней истории создания и развития, *Java* была официально представлена миру.

Литература и ресурсы

За последнее время вышло ряд книг по Java. Но интерес представляют только те из них, в которых описывается Java 2. Ниже приведен список из нескольких таких книг.

- Брюс Эккель. Философия Java. Библиотека программиста. 4-е изд., "Питер", 2010 г., 640 стр.
- Майкл Морган. Java 2. Руководство разработчика. Издательский дом "Вильямс". Москва. 2000 г.
- Эллиот Расти Гарольд. JavaBeans. Издательство "ЛЮРИ". Москва. 1999 г.
- П.Ноутон, Г.Шилдт - Java2. Наиболее полное руководство. Издательство: БХВ-Петербург, 2008 г., 1072 стр., с ил.
- Васильев А. Java. Объектно-ориентированное программирование. Учебное пособие. Стандарт третьего поколения, "Питер", 2011 г., 400 стр.

В Internet можно найти много литературы по Java, начиная от фирменной документации от Sun до учебников и различных обучающих курсов по Java.

- Русскоязычная версия сайта Sun Microsystems. <http://www.sun.ru/java> .
- Одной из популярных книг по Java является "Thinking in Java", Bruce Eckel. Она может быть получена с <http://www.bruceeckel.com/> .
- Документация от Sun доступна по адресу <http://java.sun.com/products/jdk/1.3/docs/> .
- Краткое описание доступной документации <http://java.sun.com/products/jdk/1.3/devdocs-vs-specs.html> .
- Обучающие курсы для разработчиков <http://developer.java.sun.com/developer/onlineTraining/> .
- Русскоязычный сайт по Java <http://www.javable.com/> .
- Популярный англоязычный сайт <http://www.javaworld.com/> .
- Книга Фролов А.В., Фролов Г.В. "Создание приложений Java" <http://www.sun.ru/java/books/online/> или http://athena.vvsu.ru/docs/c-java/java_f/ .
- Java FAQ (на русском) <http://www.sun.ru/java/start/questions/faq/faq.html> .
- Замечательная книга по Swing'y: *Swing* by Matthew Robinson and Pavel Vorobiev. <http://manning.spindoczone.com/sbe/> .