**Что такое Java?**

Что знают о Java обычные пользователи персональных компьютеров и Internet? Что говорят о нем разработчики, которые не занимаются этой технологией профессионально?

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 С или С++ с добавлением garbage collector'а - 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 пять основных версий языка, начиная с 1.0 в 1995 году и заканчивая 1.4 в феврале 2002 года. Следующая версия 1.5 выпущена в 2004 году. 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 создания Java

Если поискать в Internet историю создания Java, выясняется, что изначально язык назывался OaK ("дуб"), а работа по его созданию началась еще в 1990 году с довольно скандальной истории внутри корпорации Sun. Эти факты верны, однако на самом деле все было еще интереснее.

#### Сложности внутри Sun Microsystems

Действительно, события начинают разворачиваться в декабре 1990 года, когда бурное развитие WWW (World Wide Web - "всемирная паутина") никто не мог еще даже предсказать. Тогда компьютерная индустрия была поглощена взлетом персональных компьютеров. К сожалению, фирма Sun Microsystems, занимающая значительную долю рынка серверов и высокопроизводительных станций, по мнению многих сотрудников и независимых экспертов, не могла предложить ничего интересного для обычных пользователей "персоналок" - для них компьютеры от Sun представлялись "слишком сложными, очень некрасивыми и чересчур "тупыми" устройствами".

Поэтому Скотт МакНили (Scott McNealy), член совета директоров, президент и CEO (исполнительный директор) корпорации Sun, не был удивлен, когда 25-летний хорошо зарекомендовавший себя программист Патрик Нотон (Patrick Naughton), проработав всего 3 года, объявил о своем желании перейти в компанию NeXT. Они были друзьями, и Патрик объяснил свое решение просто и коротко: "Они все делают правильно". Скотт задумался на секунду и произнес историческую фразу. Он попросил Патрика перед уходом описать, что, по его мнению, в Sun делается неверно. Надо было не просто рассказать о проблеме, но предложить решение, не оглядываясь на существующие правила и традиции, как будто в его распоряжении имеются неограниченные ресурсы и возможности.

Патрик Нотон выполнил просьбу. Он безжалостно раскритиковал новую программную архитектуру NeWS, над которой фирма работала в то время, а также высоко оценил только что объявленную операционную систему NeXTstep. Нотон предложил привлечь профессиональных художников-дизайнеров, чтобы сделать пользовательские интерфейсы Sun более привлекательными; выбрать одно средство разработки и сконцентрировать усилия на одной оконной технологии, а не на нескольких сразу (Нотон был вынужден поддерживать сотни различных комбинаций технологий, платформ и интерфейсов, используемых в компании); наконец, уволить почти всех сотрудников из Window Systems Group (если выполнить предыдущие условия, они будут просто не нужны).

Конечно, Нотон был уверен, что его письмо просто проигнорируют, но все же отложил свой переход в NeXT в ожидании какой-нибудь ответной реакции. Однако она превзошла все ожидания.

МакНили разослал письмо Нотона всему управляющему составу корпорации, а те переслали его своим ведущим специалистам. Откликнулись буквально все, и, по общему мнению, Нотон описал то, о чем все думали, но боялись высказать. Решающей оказалась поддержка Билла Джоя (Bill Joy) и Джеймса Гослинга (James Gosling). Билл Джой - один из основателей и вице-президент Sun, а также участник проекта по созданию операционной системы UNIX в университете Беркли. Джеймс Гослинг пришел в Sun в 1984 году (до этого он работал в исследовательской лаборатории IBM) и был ведущим разработчиком, а также автором первой реализации текстового редактора EMACS на C. Эти люди имели огромный авторитет в корпорации.

Чтобы не останавливаться на достигнутом, Нотон решил предложить какой-то совершенно новый проект. Он объединился с группой технических специалистов, и они просидели до 4.30 утра, обсуждая базовые концепции такого проекта. Их получилось всего три: главное - потребитель, и все строится исключительно в соответствии с его интересами; небольшая команда должна спроектировать небольшую аппаратно-программную платформу; эту платформу нужно воплотить в устройстве, предназначенном для персонального пользования, удобном и простом в обращении - т.е. создать компьютер для обычных людей. Этих идей оказалось достаточно, чтобы Джон Гейдж (John Gage), руководитель научных исследований Sun, смог организовать презентацию для высшего руководства корпорации. Нотон изложил условия, которые он считал необходимыми для успешного развития этого предприятия: команда должна расположиться вне офиса Sun, чтобы не испытывать никакого сопротивления революционным идеям; проект будет секретным для всех, кроме высшего руководства Sun; аппаратная и программная платформы могут быть несовместимы с продуктами Sun; на первый год группе необходим миллион долларов.

#### Проект Green

5 декабря 1990 года, в день, когда Нотон должен был перейти в компанию NeXT, Sun сделала ему встречное предложение. Руководство согласилось со всеми его условиями. Поставленная задача - "создать что-нибудь необычайное". 1 февраля 1991 года Патрик Нотон, Джеймс Гослинг и Майк Шеридан (Mike Sheridan) вплотную приступили к реализации проекта, который получил название Green.

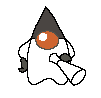
Цель они выбрали себе амбициозную - выяснить, какой будет следующая волна развития компьютерной индустрии (первыми считаются появление полупроводников и персональных компьютеров) и какие продукты необходимо разработать для успешного участия в ней. С самого начала проект не рассматривался как чисто исследовательский, задача была создать реальный продукт, устройство.

На ежегодном собрании Sun весной 1991 года Гослинг заметил, что компьютерные чипы получили необычайное распространение, они применяются в видеомагнитофонах, тостерах, даже в дверных ручках гостиниц! Тем не менее, до сих пор в каждом доме можно увидеть до трех пультов дистанционного управления - для телевизора, видеомагнитофона и музыкального центра. Так родилась идея разработать небольшое устройство с жидкокристаллическим сенсорным экраном, которое будет взаимодействовать с пользователем с помощью анимации, показывая, чем можно управлять и как. Чтобы создать такой прибор, Нотон начал работать над специализированной графической системой, Гослинг взялся за программное обеспечение, а Шеридан занялся бизнес-вопросами.

В апреле 1991 года команда покидает офис Sun, отключаясь даже от внутренней сети корпорации, и въезжает в новое помещение. Закупаются разнообразные бытовые электронные устройства, такие как игровые приставки Nintendo, телевизионные приставки, пульты дистанционного управления, и разработчики играют в различные игры целыми днями, чтобы лучше понять, как сделать пользовательский интерфейс легким в понимании и использовании. В качестве идеального примера Гослинг отмечал, что современные тостеры с микропроцессорами имеют точно такой же интерфейс, что и тостер его мамы, который служит уже 42 года. Очень быстро исследователи обнаружили, что практически все устройства построены на самых разных центральных процессорах. Это означает, что добавление новых функциональных возможностей крайне затруднено, так как необходимо учитывать ограничения и, как правило, довольно скудные возможности используемых чипов. Когда же Гослинг побывал на концерте, где смог воочию наблюдать сложное переплетение проводов, огромное количество колонок и полуавтоматических прожекторов, которые, казалось, согласованно двигаются в такт музыке, он понял, что будущее - за объединением сетей, компьютеров и других электронных устройств в единую согласованную инфраструктуру.

Сначала Гослинг попытался модифицировать С++, чтобы создать язык для написания программ, минимально ориентированных на конкретные платформы. Однако очень скоро стало понятно, что это практически невозможно. Основное достоинство С++ - скорость программ, но отнюдь не их надежность. А надежность работы для обычных пользователей должна быть так же абсолютно гарантирована, как совместимость обычных электрических вилки и розетки. Поэтому в июне 1991 года Гослинг, который написал свой первый язык программирования в 14 лет, начинает разработку замены C++. Создавая новый каталог и раздумывая, как его назвать, он выглянул в окно, и взгляд его остановился на растущем под ним дереве. Так язык получил свое первое название - OaK (дуб). Спустя несколько лет, после проведения маркетинговых исследований, имя сменили на Java.

Всего несколько месяцев потребовалось, чтобы довести разработку до стадии, когда стало возможным совместить новый язык с графической системой, над которой работал Нотон. Уже в августе команда смогла запустить первые программы, демонстрирующие возможности будущего устройства.



Само устройство, по замыслу создателей, должно было быть размером с обычный пульт дистанционного управления, работать от батареек, иметь привлекательный и забавный графический интерфейс и, в конце концов, стать любимой (и полезной!) домашней игрушкой. Чтобы построить этот не имеющий аналогов прибор, находчивые разработчики применили "технологию молотка". Они попросту находили какой-нибудь аппарат, в котором были подходящие детали или микросхемы, разбивали его молотком и таким образом добывали необходимые части. Так были получены основной жидкокристаллический экран, сенсорный экран и миниатюрные встроенные колонки. Центральный процессор и материнская плата были специально разработаны на основе высокопроизводительной рабочей станции Sun. Было придумано и оригинальное название - \*7, или Star7 (с помощью этой комбинации кнопок можно было ответить с любого аппарата в офисе на звонок любого другого телефона, а поскольку редко кого из них можно было застать на рабочем месте, эти слова очень часто громко кричали на весь офис). Для придания интерфейсу большей привлекательности разработчики создали забавного персонажа по имени Дьюк (Duke), который всегда был готов помочь пользователю выполнить его задачу. В дальнейшем он стал спутником Java, счастливым талисманом - его можно встретить во многих документах, статьях, примерах кода.

Задача была совершенно новая, не на что было опереться, не было никакого опыта, никаких предварительных наработок. Команда трудилась, не прерываясь ни на один день. В августе 1991 года состоялась первая демонстрация для Билла Джоя и Скотта МакНили. В ноябре группа снова подключилась к сети Sun по модемной линии. Чем дальше развивался проект, тем больше новых специалистов присоединялось к команде разработчиков. Примерно в то время было придумано название для той идеологии, которую они создавали,- 1st Person (условно можно перевести как "первое лицо").

Наконец, 4 сентября 1992 года Star7 был завершен и продемонстрирован МакНили. Это было небольшое устройство с 5" цветным (16 бит) сенсорным экраном, без единой кнопки. Чтобы включить его, надо было просто дотронуться до экрана. Весь интерфейс был построен как мультик - никаких меню! Дьюк перемещался по комнатам нарисованного дома, а чтобы управлять им, надо было просто водить по экрану пальцем - никаких специальных средств управления. Можно было взять виртуальную телепрограмму с нарисованного дивана, выбрать передачу и "перетащить" ее на изображение видеомагнитофона, чтобы запрограммировать его на запись.

Результат превзошел все ожидания! Стоит напомнить, что устройства типа карманных компьютеров (PDA), начиная с Newton, появились заметно позже, не говоря уже о цветном экране. Это было время 286i и 386i процессоров Intel (486i уже появились, но стоили очень дорого) и MS DOS, даже мышь еще не была обязательным атрибутом персонального компьютера.

Руководители Sun были просто в восторге - появилось отличное оружие против таких могучих конкурентов, как HP, IBM и Microsoft. Новая технология была способна не только демонстрировать мультики. Объектно-ориентированный язык OaK обещал стать достаточно мощным инструментом для написания программ, которые могут работать в сетевом окружении. Его объекты, свободно распространяемые по сети, работали бы на любом устройстве, начиная с персонального компьютера и заканчивая обычными бытовыми видеомагнитофонами и тостерами. На презентациях Нотон представлял области применения OaK, изображая домашние компьютеры, машины, телефоны, телевизоры, банки и соединяя их единой сетью. Целое приложение, например, для работы с электронной почтой, могло быть построено в виде группы таких объектов, причем они не обязательно должны были располагаться на одном устройстве. Более того, как язык, ориентированный на распределенную архитектуру, OaK имел механизмы безопасности, шифрования, процедур аутентификации, причем все эти возможности были встроенные, а значит, незаметные и удобные для пользователя.

#### Компания FirstPerson

Крупные компании-производители, такие как Mitsubishi Electric, France Telecom, Dolby Labs, заинтересовались новой технологией, начались переговоры. Шеридан подготовил бизнес-план с оригинальным названием "Beyond the Green Door" ("За зеленой дверью"), в котором предложил Sun учредить дочернюю компанию для продвижения платформы OaK на рынок. 1 ноября 1992 года создается компания FirstPerson, которую возглавила Вэйн Роузинг (Wayne Rosing), перешедшая из Sun Labs. Арендуется роскошный офис, число сотрудников возрастает с 14 до 60 человек.

Однако позднее оказалось, что стоимость подобного решения (процессор, память, экран) составляет не менее $50. Производители же бытовой техники не привыкли платить значительные суммы за дополнительную функциональность, облегчающую использование их продуктов.

В это время внимание компьютерной индустрии захватывает идея интерактивного телевидения, создается ощущение, что именно оно станет следующим революционным прорывом. Поэтому, когда в марте 1993 года Time Warner объявляет конкурс для производителей компьютерных приставок к телевизору для развертывания пробной сети интерактивного телевидения, FirstPerson полностью переключается на эту задачу. И снова неудача - победителем оказывается Джеймс Кларк (James Clark), основатель Silicon Graphics Inc., несмотря на то, что технологически его предложение уступает OaK. Впрочем, через год проект Time Warner и SGI проваливается, а Джеймс Кларк создает компанию Netscape, которая еще сыграет важную роль в успехе Java.

Другим потенциальным клиентом стал производитель игровых приставок 3DO. Понадобилось всего 10 дней, чтобы импортировать OaK на эту платформу, однако после трехмесячных переговоров директор 3DO потребовал полные права на новый продукт, и сделка не состоялась.

Наконец, в начале 1994 года стало понятно, что идея интерактивного телевидения оказалась нежизнеспособной. Ожиданиям не суждено было стать реальностью. Анализ состояния FirstPerson показал, что компания не имеет ни одного клиента или партнера и ее дальнейшие перспективы довольно туманны. Руководство Sun требует немедленного составления нового бизнес-плана, позволяющего компании снова приносить прибыль.

#### World Wide Web

В погоне за призраком интерактивного телевидения многие участники компьютерного рынка пропустили поистине эпохальное событие. В апреле 1993 года Марк Андриссен (Marc Andreessen) и Эрик Бина (Eric Bina), работающие в Национальном центре суперкомпьютерных приложений (National Center for Supercomputing Applications, NCSA) при университете Иллинойс, выпустили первую версию графического браузера ("обозревателя") Mosaic 1.0 для WWW. Хотя Internet существовал на тот момент уже около 20 лет, имеющимися протоколами связи (FTP, telnet и др.) пользоваться было очень неудобно и Глобальная Сеть использовалась лишь в академической и государственной среде. Mosaic же основывался на новом языке разметки гипертекстовых документов (HyperText Markup Language, HTML ), который с 1991 года разрабатывался в Европейском институте физики частиц (CERN) специально для представления информации в Internet. Этот формат позволял просматривать текст и изображения, а главное - поддерживал ссылки, с помощью которых можно было одним нажатием мыши перейти как на другую часть той же страницы, так и на страницу, которая могла располагаться совсем в другой части сети и в любой точке планеты. Именно такие перекрестные обращения, используя которые, пользователь мог незаметно для себя посетить множество узлов Internet, и позволили считать все HTML -документы связанными частями единого целого - Всемирной Паутины (World Wide Web, WWW).

И самое важное - все эти новые достижения были совершенно бесплатны и доступны для всех желающих. Впервые обычные пользователи персональных компьютеров без всякой специальной подготовки могли пользоваться глобальной сетью не только для решения рабочих вопросов, но и для поиска информации на самые разные темы. Количество документов в пространстве WWW стало расти экспоненциально, и очень скоро сеть Internet стала поистине Всемирной. Правда, со временем обнаружилось, что такой способ организации и хранения информации очень напоминает свалку, в которой крайне трудно найти данные по какому-нибудь конкретному вопросу, однако эта тема относится к совершенно другому этапу развития компьютерного мира. Итак, каким-то непостижимым образом Sun не замечает зарождения новой эпохи. Технический директор Sun впервые увидел Mosaic лишь три месяца спустя! И это притом, что около 50% серверов и рабочих станций в сети Internet были произведены именно Sun.

Новый бизнес-план FirstPerson ставил цель, которая была неким промежуточным шагом от интерактивного телевидения к возможностям Internet. Идея заключалась в создании платформы для кабельных компаний, пользователями которой были бы обычные владельцы персональных компьютеров, объединенные сетями таких компаний. Используя технологию OaK, разработчики могли бы создавать приложения, по функциональности аналогичные программам, распространяемым на CD-ROM, однако обладающие интерактивностью, позволяющей людям обмениваться любой информацией через сеть. Ожидалось, что такие сети в итоге и разовьются в интерактивное телевидение, и тогда OaK станет полноценным решением для этой индустрии. Об Internet и Mosaic пока не говорилось ни слова.

По многим причинам этот план не устроил руководство Sun (он не вполне соответствовал главному ожиданию - новая разработка должна была привести к увеличению спроса на продукты Sun). Из-за отсутствия перспектив половина сотрудников FirstPerson была переведена в только что созданную команду Sun Interactive, которая продолжила заниматься мультимедиа-сервисами уже без OaK. Все предприятие оказалось под угрозой бесславной кончины, однако в этот момент Билл Джой снова оказал поддержку проекту, который вскоре дал миру платформу Java.

Когда создатели FirstPerson, наконец, обратили внимание на Internet, они поняли, что функциональность тех сетевых приложений, для которых создавался OaK, очень близка к WWW. Билл Джой вспомнил, как он двадцать лет назад принимал участие в разработке UNIX в Беркли и затем эта операционная система получила широчайшее распространение благодаря тому, что ее можно было загрузить по сети бесплатно. Такой принцип бесплатного распространения коммерческих продуктов создал саму WWW, тем же путем компания Netscape вскоре стала лидером рынка браузеров, так многие технологии получили возможность захватить долю рынка в кратчайшие сроки. Эти новые идеи при поддержке Джоя окончательно убедили руководство Sun, что Internet поможет воскресить платформу OaK (кстати, этот новый проект поначалу называли "Liveoak"). В итоге Джой садится писать очередной бизнес-план и отправляет Гослинга и Нотона начинать работу по адаптации OaK для Internet. Гослинг пересматривает программный код платформы, а Нотон берется за написание "убойного" приложения, которое сразу бы продемонстрировало всю мощь OaK для Internet.

В самом деле, эти технологии прекрасно подошли друг другу. Языки программирования всегда играли важную роль в развитии компьютерных технологий. Мэйнфреймы не были особенно полезны, пока не появился Cobol. Благодаря языку Fortran от IBM, компьютеры стали широко применяться для научных вычислений и исследований. Altair BASIC - самый первый продукт от Microsoft - позволил всем программистам-любителям создавать программы для своих персональных компьютеров. Язык С++ стал основой для развития графических пользовательских интерфейсов, таких как Mac OS и Windows. Создатели OaK сделали все, чтобы эта технология сыграла такую же роль в программировании для Internet.

Несмотря на то, что к середине 1994 года WWW достиг невиданных размеров (конечно, по меркам того времени), web-страницы по-прежнему были скорее похожи на обычные бумажные издания, чем на интерактивные приложения. По большей части вся работа в сети заключалась в отправке запроса на web-сервер и получении ответа, который содержал обычный статический HTML -файл, отображаемый браузером на стороне клиента. Уже тогда функциональность web-серверов расширялась с помощью CGI (Common Gateway Interface). Эта технология позволяла по запросу клиента запускать на сервере обычную программу и ее результат отсылать обратно в качестве ответа. Поскольку в то время скорость каналов связи была невысокой (хотя, похоже, пользователи никогда не будут удовлетворены возможностями аппаратуры), клиент мог ждать несколько минут, чтобы лишь увидеть сообщение о том, что он ошибся в одной букве запроса. Динамическое построение графиков при таком способе реализации означало бы генерацию GIF-файлов в реальном времени. А ведь зачастую клиентские машины являются полноценными персональными компьютерами, которые могли бы брать значительную часть работы взаимодействия с пользователем на себя, разгружая серверы.

Вообще, клиент-серверная архитектура, просто необходимая для большинства сложных корпоративных (enterprise) приложений, обладает рядом существенных технических сложностей. Основная идея - разместить общие данные на сервере, чтобы создать единое информационное пространство для работы многих пользователей, а программы, отображающие и позволяющие удобно редактировать эти данные, выполняются на клиентских машинах. Очень часто в корпорации используется несколько аппаратных платформ (это может быть как "историческое наследие", так и следствие того, что различные подразделения, решая свои задачи, нуждаются в различных компьютерах). Следовательно, приложение необходимо развивать сразу в нескольких вариантах, что существенно увеличивает стоимость поддержки. Кроме того, обновление клиентской части означает, что нужно перенастроить все компьютеры компании в кратчайший срок. А ведь обновлениями часто занимаются несколько групп разработчиков.

Попытка придать Internet- браузерам возможности полноценного клиентского приложения встречает еще большие трудности. Во-первых, обычные сложности предельно возрастают - в Internet представлены практически все существующие платформы, а количество и географическая распределенность пользователей делает быстрое обновление просто невозможным. Во-вторых, особенно остро встает вопрос безопасности. Через сеть удивительно быстро распространяется не только важная информация, но и вирусы. Текстовая информация и изображения не несут в себе никакой угрозы для клиентской машины, другое дело - исполняемый код. Наконец, приложения с красивым и удобным графическим интерфейсом, как правило, имели немаленький размер, недаром основным средством их распространения были CD-ROM'ы. Понятно, что для Internet необходимо было серьезно поработать над компактностью кода.

Если оглянуться на историю развития OaK, становится понятно, что эта платформа удивительным образом отвечает всем перечисленным требованиям Internet-программирования, хотя и создавалась во времена, когда про WWW никто даже и не думал. Видимо, это говорит о том, насколько верно предугадали развитие индустрии участники проекта Green.

#### Возрождение OaK

Для победного выхода OaK не хватало последнего штриха - браузера, который поддерживал бы эту технологию. Именно он должен был стать тем самым "убойным" приложением Нотона, которое завершало почти пятилетнюю подготовительную работу перед официальным объявлением новой платформы.

Браузер назвали WebRunner. Нотону потребовался всего один выходной, чтобы написать основную часть программы. Это было в июле, а в сентябре 1994 года WebRunner уже демонстрировался руководству Sun. Небольшие программы, написанные на OaK для распространения через Internet, назвали апплетами ( applets ).



Следующая демонстрация происходила на конференции, где встречались разработчики Internet-приложений и представители индустрии развлечений. Когда Гослинг начал презентацию WebRunner, слушатели не проявили большого интереса, решив, что это просто клон Mosaic. Тогда Гослинг провел мышкой над сложной трехмерной моделью химической молекулы.

Следуя за курсором, модель поворачивалась по всем направлениям! Сейчас данная функция, возможно, не производит такого впечатления, однако в то время это было подобно переходу от картинки к кинематографу. Следующий пример демонстрировал анимированную сортировку. Вначале изображался набор отрезков разной длины. Затем синяя и красная линии начинали бегать по этому набору, сортируя отрезки по размеру. Пример тоже нехитрый, однако наглядно демонстрирующий, что на стороне клиента появилась полноценная программная платформа. Оба эти апплета сейчас являются стандартными примерами и входят в состав Java Development Kit любой версии. Успех демонстрации, которая закончилась бурными аплодисментами, показал, что OaK и WebRunner способны устроить революцию в Internet, так как все участники конференции по-другому взглянули на возможности, которые предоставляет Всемирная Сеть.

Кстати, в начале 1995 года, когда стало ясно, что официальное объявление уже не за горами, за дело взялись маркетологи. В результате их исследований OaK был переименован в Java, а WebRunner стал называться HotJava. Многие тогда недоумевали, что же послужило поводом для такого решения. Легенда гласит, что Java - это сорт кофе (такой кофе действительно есть), который очень любили программисты. Видимо, похожим образом родилось и название HotJava ("горячая Java "). Тема кофе навсегда останется в названиях и логотипах ( технология создания компонентов названа Java Beans - зерна кофе, специальный формат для архивирования файлов с Java -программами JAR - банка с кофе и т.д.), а сам язык критики стали называть "для кофеварок". Впрочем, сейчас все уже привыкли и не задумываются над названием, возможно, на это и было рассчитано (а тем, кто продолжает выражать недовольство, приводят альтернативные варианты, которые рассматривались тогда - Neon, Lyric, Pepper или Silk).

Согласно плану, спецификация Java, реализация платформы и HotJava должны были свободно распространяться через Internet. С одной стороны, это позволяло в кратчайшие сроки распространить технологию по всему миру и сделать ее стандартом де-факто для Internet-программирования. С другой стороны, при участии всего сообщества разработчиков, которые высказывали бы свои замечания, можно было гораздо быстрее устранить все возможные ошибки и недоработки. Однако в конце 1994 года лишь считаные копии были распространены за пределы Sun. В феврале 1995 года выходит, возможно, первый пресс-релиз, сообщающий, что вскоре будут доступны альфа-версии OaK и WebRunner.

Когда это произошло, команда стала подсчитывать случаи загрузки их продукта для просмотра. Вскоре пришлось считать уже сотнями. Затем решили, что если удастся достигнуть 10.000, то это будет просто ошеломляющий успех. Ждать пришлось совсем не так долго, как можно было предположить. Интерес нарастал лавинообразно, после просмотров приходило большое количество писем и мощности Internet-канала стало не хватать. На письма всегда отвечали очень подробно, что поначалу можно было делать, не отрываясь от работы. Затем по очереди стали назначать одного разработчика, чтобы он в течение недели только писал ответы. Наконец, потребовался специальный сотрудник, так как приходило уже по 2-3 тысячи писем в день. Вскоре руководство Sun осознало, что имея такой мощный успех Java не имеет никакого бюджета или плана для рекламы и других акций продвижения на рынок. Первым шагом в этом направлении становится публикация 23 марта 1995 года в газете Sun Jose Mercury News статьи с описанием новой технологии, где был приведен адрес официального сайта <http://java.sun.com/>, который и по сей день является основным источником информации по Java.

#### Java выходит в свет

Наконец, вся подготовительная работа стала подходить к своему логическому завершению. Официальное объявление Java, уже получившей широкое признание и подающей большие надежды, должно было произойти на конференции SunWorld. Ожидалось, что это будет короткое информационное объявление, так как главная цель этого мероприятия - UNIX-системы. Однако все произошло не так, как планировалось.

В четыре часа утра в день конференции, после длинных и сложных переговоров, Sun подписывает важнейшее соглашение. Вторая сторона - компания Netscape, основанная в апреле 1994 года Джеймсом Кларком (он уже сыграл роль в судьбе OaK два года назад, когда перехватил предложение от Time Warner) и Марком Андриссеном (создателем NCSA Mosaic). Эта компания стала лидером рынка браузеров после того, как в декабре 1994 года вышла первая версия Netscape Navigator, которая была открыта для бесплатного некоммерческого использования, что позволило занять на тот момент 75% рынка.

23 мая 1995 года технологии Java и HotJava были официально объявлены Sun и тогда же представители компании сообщили, что новая версия самого популярного браузера Netscape Navigator 2.0 будет поддерживать новую технологию. По сути, это означало, что отныне Java становится такой же неотъемлемой частью WWW, как и HTML. Уже второй раз презентация закончилась овацией - победное шествие Java началось.

### 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 расползалась.

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 сегодня, как известно, самый распространенный браузер - Microsoft Internet Explorer.

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". Сейчас появилась версия 7.0, однако она не занимает значительной доли рынка, в то время как Internet Explorer 5.0, 5.5 и 6.0 используют 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 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 людских 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.

Про HotJava, к сожалению, сказать особенно нечего. Некоторое время Sun поддерживала этот продукт и добавила возможность визуально генерировать web-страницы без знания HTML. Однако создать конкурентоспособный браузер не удалось и вскоре развитие HotJava было остановлено. Сейчас еще можно скачать и посмотреть последнюю версию 3.0.

И 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 С;
* язык Java Script не объектно-ориентированный (хотя некоторые аспекты объектно-ориентированного подхода поддерживаются), но 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 удается 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, так же, как домашние пользователи не имеют представления, на каких микросхемах собран их мобильный телефон или видеомагнитофон.

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. Эта тяжба продолжается до сих пор с переменным успехом сторон. Например, Microsoft исключила из виртуальной машины Internet Explorer библиотеку java.rmi, позволяющую создавать распределенные приложения, пытаясь привлечь внимание разработчиков к DCOM-технологии, жестко привязанной к платформе Win32. В ответ многие компании стали распространять специальное дополнение (patch), устраняющее этот недостаток. В результате Microsoft остановила свою поддержку Java на версии 1.1, которая на данный момент является устаревшей и не имеет многих полезных возможностей. Это, в свою очередь, практически остановило широкое распространение апплетов, кроме случаев либо совсем несложной функциональности (типа бегущей строки или диалога с несколькими полями ввода и кнопками), либо приложений для внутренних сетей корпораций. Для последнего случая 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). На данный момент Microsoft то включает, то исключает Java из своей операционной системы Windows XP, видимо, пытаясь найти самый выгодный для себя вариант.

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.

|  |
| --- |
| 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 то же 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 простыми, или примитивными (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 удается 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, С++ 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 С/С++ 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 С/С++, 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 считаные 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 получат 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 С/С++; * 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.х.х. 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 4 (последняя на момент создания курса). В каждой версии происходило существенное расширение стандартных библиотек (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 меняться (добавляться) аргументы утилит. Так, последняя версия JDK 1.0 - 1.0.2.  Хотя с развитием версии 1.х ничего не удаляется, конечно, какие-то функции или классы устаревают. Они объявляются deprecated, и хотя они будут поддерживаться до объявления 2.0 (а про нее пока ничего не было слышно), пользоваться ими не рекомендуется.  Вместе с первым успехом JDK 1.0 подоспела 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 еще 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 самый распространенный браузер MS IE до сих пор поддерживает 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.х.х, 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 предназначается 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. Последняя версия j2sdk 1.2 на данный момент - 1.2.2.  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 1.3 (последняя подверсия 1.3.1).  Наконец, последняя на данный момент 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.  За первые пять месяцев j2sdk 1.4 было скачано более двух миллионов раз. В августе 2002 года уже была предложена версия 1.4.1, остающаяся на данный момент самой современной.  В заключение для демонстрации уровня развития Standard Edition приведем стандартные диаграммы, описывающие все составляющие технологии, из документации к версиям 1.3 и 1.4.  1.1.Составляющие технологии версии 1.3.  1.1.Составляющие технологии версии 1.3.  1.2. Составляющие технологии версии 1.4.  1.2. Составляющие технологии версии 1.4. |
| http://www.intuit.ru/img/empty.gif |
| http://www.intuit.ru/img/empty.gif |

### 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.

Глобальная сеть появилась в апреле 1993 года с выходом первого браузера Mosaic 1.0 и завоевывала пользовательскую аудиторию с поразительной скоростью. 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. Благодаря подписанию лицензионного соглашения с Netscape, это событие стало поистине триумфальным.

Были рассмотрены различные варианты применения Java. Отдельно был описан язык Java Script, который, несмотря на сходство в названии, имеет не так много общего с Java. Подробно рассмотрены отличительные особенности Java. Описаны базовые продукты от Sun: JDK и JRE. Кратко освещена история развития версий платформы Java, включая добавляемые технологии и продукты.

**Методология процедурно-ориентированного программирования**

Появление первых электронных вычислительных машин, или компьютеров, ознаменовало новый этап в развитии техники вычислений. Казалось, достаточно разработать последовательность элементарных действий, каждое из которых можно преобразовать в понятные компьютеру инструкции, и любая вычислительная задача будет решена. Эта идея оказалась настолько жизнеспособной, что долгое время доминировала над всем процессом разработки программ. Появились специализированные языки программирования, созданные для разработки программ, предназначенных для решения вычислительных задач. Примерами таких языков могут служить FOCAL (FOrmula CALculator) и FORTRAN (FORmula TRANslator).

Основой такой методологии разработки программ являлась процедурная, или алгоритмическая, организация структуры программного кода. Это было настолько естественно для решения вычислительных задач, что целесообразность такого подхода ни у кого не вызывала сомнений. Исходным в данной методологии было понятие алгоритма. Алгоритм - это способ решения вычислительных и других задач, точно описывающий определенную последовательность действий, которые необходимо выполнить для достижения заданной цели. Примерами алгоритмов являются хорошо известные правила нахождения корней квадратного уравнения или системы линейных уравнений.

При увеличении объемов программ для упрощения их разработки появилась необходимость разбивать большие задачи на подзадачи. В языках программирования возникло и закрепилось новое понятие процедуры. Использование процедур позволило разбивать большие задачи на подзадачи и таким образом упростило написание больших программ. Кроме того, процедурный подход позволил уменьшить объем программного кода за счет написания часто используемых кусков кода в виде процедур и их применения в различных частях программы.

Как и алгоритм, процедура представляет собой законченную последовательность действий или операций, направленных на решение отдельной задачи. В языках программирования появилась специальная синтаксическая конструкция, которая также получила название процедуры. Например, на языке Pascal описание процедуры выглядит следующим образом:

Procedure printGreeting(name: String)

Begin

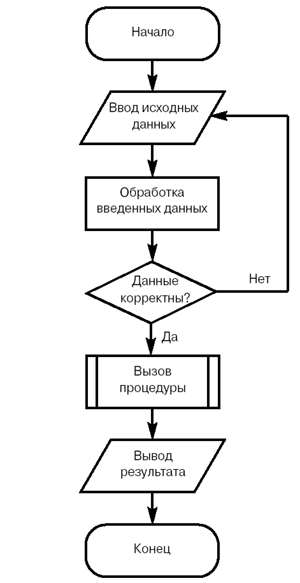
Print("Hello, ");

PrintLn(name);

End;

Назначение данной процедуры - вывести на экран приветствие Hello, Name, где Name передается в процедуру в качестве входного параметра.

Со временем вычислительные задачи становились все сложнее, а значит, и решающие их программы увеличивались в размерах. Их разработка превратилась в серьезную проблему. Когда программа становится все больше, ее приходится разделять на все более мелкие фрагменты. Основой для такого разбиения как раз и стала процедурная декомпозиция, при которой отдельные части программы, или модули, представляли собой совокупность процедур для решения одной или нескольких задач. Одна из основных особенностей процедурного программирования заключается в том, что оно позволило создавать библиотеки подпрограмм (процедур), которые можно было бы использовать повторно в различных проектах или в рамках одного проекта. При процедурном подходе для визуального представления алгоритма выполнения программы применяется так называемая блок-схема . Соответствующая система графических обозначений была зафиксирована в ГОСТ 19.701-90. Пример блок-схемы изображен на рисунке ([рис. 2.1](http://www.intuit.ru/department/pl/javapl/2/#image.2.1)).



**Рис. 2.1.**  Пример блок-схемы.

Появление и интенсивное использование условных операторов и оператора безусловного перехода стало предметом острых дискуссий среди специалистов по программированию. Дело в том, что бесконтрольное применение в программе оператора безусловного перехода goto может заметно осложнить понимание кода. Такие запутанные программы сравнивали с порцией спагетти (bowl of spaghetti), имея в виду многочисленные переходы от одного фрагмента программы к другому, или, что еще хуже, возврат от конечных операторов программы к начальным. Ситуация казалась настолько драматичной, что многие предлагали исключить оператор goto из языков программирования. Именно с этого времени отсутствие безусловных переходов стали считать хорошим стилем программирования.

Дальнейшее увеличение программных систем способствовало формированию новой точки зрения на процесс разработки программ и написания программных кодов, которая получила название методологии структурного программирования. Ее основой является процедурная декомпозиция предметной области решаемой задачи и организация отдельных модулей в виде совокупности процедур. В рамках этой методологии получило развитие нисходящее проектирование программ, или проектирование "сверху вниз". Пик популярности идей структурного программирования приходится на конец 70-х - начало 80-х годов.

В этот период основным показателем сложности разработки программы считался ее размер. Вполне серьезно обсуждались такие оценки сложности программ, как количество строк программного кода. Правда, при этом делались некоторые предположения относительно синтаксиса самих строк, которые должны были соответствовать определенным требованиям. Например, каждая строка кода должна была содержать не более одного оператора. Общая трудоемкость разработки программ оценивалась специальной единицей измерения - "человеко-месяц", или "человеко-год". А профессионализм программиста напрямую связывался с количеством строк программного кода, который он мог написать и отладить в течение, скажем, месяца.

**Методология объектно-ориентированного программирования**

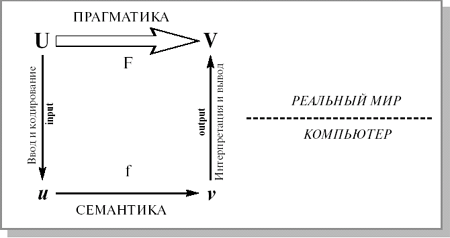
Увеличение размеров программ приводило к необходимости привлечения большего числа программистов, что, в свою очередь, потребовало дополнительных ресурсов для организации их согласованной работы. В процессе разработки приложений заказчик зачастую изменял функциональные требования, что еще более усложняло процесс создания программного обеспечения.

Но не менее важными оказались качественные изменения, связанные со смещением акцента использования компьютеров. В эпоху "больших машин" основными потребителями программного обеспечения были такие крупные заказчики, как большие производственные предприятия, финансовые компании, государственные учреждения. Стоимость таких вычислительных устройств для небольших предприятий и организаций была слишком высока.

Позже появились персональные компьютеры, которые имели гораздо меньшую стоимость и были значительно компактнее. Это позволило широко использовать их в малом и среднем бизнесе. Основными задачами в этой области являются обработка данных и манипулирование ими, поэтому вычислительные и расчетно-алгоритмические задачи с появлением персональных компьютеров отошли на второй план. Как показала практика, традиционные методы процедурного программирования не способны справиться ни с нарастающей сложностью программ и их разработки, ни с необходимостью повышения их надежности. Во второй половине 80-х годов возникла настоятельная потребность в новой методологии программирования, которая была бы способна решить весь этот комплекс проблем. Ею стало объектно-ориентированное программирование (ООП).

После составления технического задания начинается этап проектирования, или дизайна, будущей системы. Объектно-ориентированный подход к проектированию основан на представлении предметной области задачи в виде множества моделей для независимой от языка разработки программной системы на основе ее прагматики.

Последний термин нуждается в пояснении. Прагматика определяется целью разработки программной системы, например, обслуживание клиентов банка, управление работой аэропорта, обслуживание чемпионата мира по футболу и т.п. В формулировке цели участвуют предметы и понятия реального мира, имеющие отношение к создаваемой системе (см. [рисунок 2.2](http://www.intuit.ru/department/pl/javapl/2/#image.2.2) [[3](http://www.intuit.ru/department/pl/javapl/popup.lit.html#3)]). При объектно-ориентированном подходе эти предметы и понятия заменяются моделями, т.е. определенными формальными конструкциями.



**Рис. 2.2.**  Семантика (смысл программы с точки зрения выполняющего ее компьютера) и прагматика (смысл программы с точки зрения ее пользователей) [3].

Модель содержит не все признаки и свойства представляемого ею предмета или понятия, а только те, которые существенны для разрабатываемой программной системы. Таким образом, модель "беднее", а следовательно, проще представляемого ею предмета или понятия.

Простота модели по отношению к реальному предмету позволяет сделать ее формальной. Благодаря такому характеру моделей при разработке можно четко выделить все зависимости и операции над ними в создаваемой программной системе. Это упрощает как разработку и изучение (анализ) моделей, так и их реализацию на компьютере.

Объектно-ориентированный подход обладает такими преимуществами, как:

* уменьшение сложности программного обеспечения;
* повышение надежности программного обеспечения;
* обеспечение возможности модификации отдельных компонентов программного обеспечения без изменения остальных его компонентов;
* обеспечение возможности повторного использования отдельных компонентов программного обеспечения.

Более детально преимущества и недостатки объектно-ориентированного программирования будут рассмотрены в конце лекции, так как для их понимания необходимо знание основных понятий и положений ООП.

Систематическое применение объектно-ориентированного подхода позволяет разрабатывать хорошо структурированные, надежные в эксплуатации, достаточно просто модифицируемые программные системы. Этим объясняется интерес программистов к объектно-ориентированному подходу и объектно-ориентированным языкам программирования. ООП является одним из наиболее интенсивно развивающихся направлений теоретического и прикладного программирования.

### Объекты

По определению будем называть объектом понятие, абстракцию или любой предмет с четко очерченными границами, имеющий смысл в контексте рассматриваемой прикладной проблемы. Введение объектов преследует две цели:

* понимание прикладной задачи (проблемы);
* введение основы для реализации на компьютере.

Примеры объектов: форточка, Банк "Империал", Петр Сидоров, дело № 7461, сберкнижка и т.д.

Каждый объект имеет определенное время жизни. В процессе выполнения программы, или функционирования какой-либо реальной системы, могут создаваться новые объекты и уничтожаться уже существующие.

Гради Буч дает следующее определение объекта:

Объект - это мыслимая или реальная сущность, обладающая характерным поведением и отличительными характеристиками и являющаяся важной в предметной области [[2](http://www.intuit.ru/department/pl/javapl/popup.lit.html#2)].

Каждый объект имеет состояние, обладает четко определенным поведением и уникальной идентичностью.

#### Состояние

Рассмотрим пример. Любой человек может находиться в некотором положении (состоянии): стоять, сидеть, лежать, и - в то же время совершать какие-либо действия.

Например, человек может прыгать, если он стоит, и не может - если он лежит, для этого ему потребуется сначала встать. Также в объектно-ориентированном программировании состояние объекта может определяться наличием или отсутствием связей между моделируемым объектом и другими объектами. Более подробно все возможные связи между объектами будут рассмотрены в разделе "Типы отношений между классами ".

Например, если у человека есть удочка (у него есть связь с объектом "Удочка"), он может ловить рыбу, а если удочки нет, то такое действие невозможно. Из этих примеров видно, что набор действий, которые может совершать человек, зависит от параметров объекта, его моделирующего.

Для рассмотренных выше примеров такими характеристиками, или атрибутами, объекта "Человек" являются:

* текущее положение человека (стоит, сидит, лежит);
* наличие удочки (есть или нет).

В конкретной задаче могут появиться и другие свойства, например, физическое состояние, здоровье (больной человек обычно не прыгает).

Состояние (state) - совокупный результат поведения объекта: одно из стабильных условий, в которых объект может существовать, охарактеризованных количественно; в любой момент времени состояние объекта включает в себя перечень (обычно статический) свойств объекта и текущие значения (обычно динамические) этих свойств [[2](http://www.intuit.ru/department/pl/javapl/popup.lit.html#2)].

#### Поведение

Для каждого объекта существует определенный набор действий, которые с ним можно произвести. Например, возможные действия с некоторым файлом операционной системы ПК:

* создать;
* открыть;
* читать из файла;
* писать в файл;
* закрыть;
* удалить.

Результат выполнения действий зависит от состояния объекта на момент совершения действия, т.е. нельзя, например, удалить файл, если он открыт кем-либо (заблокирован). В то же время действия могут менять внутреннее состояние объекта - при открытии или закрытии файла свойство "открыт" принимает значения "да" или "нет", соответственно.

Программа, написанная с использованием ООП, обычно состоит из множества объектов, и все эти объекты взаимодействуют между собой. Обычно говорят, что взаимодействие между объектами в программе происходит посредством передачи сообщений между ними.

В терминологии объектно-ориентированного подхода понятия "действие", "сообщение" и "метод" являются синонимами. Т.е. выражения "выполнить действие над объектом ", "вызвать метод объекта " и "послать сообщение объекту для выполнения какого-либо действия" эквивалентны. Последняя фраза появилась из следующей модели. Программу, построенную по технологии ООП, можно представить себе как виртуальное пространство, заполненное объектами, которые условно "живут" некоторой жизнью. Их активность проявляется в том, что они вызывают друг у друга методы, или посылают друг другу сообщения. Внешний интерфейс объекта, или набор его методов,- это описание того, какие сообщения он может принимать.

Поведение (behavior) - действия и реакции объекта, выраженные в терминах передачи сообщений и изменения состояния ; видимая извне и воспроизводимая активность объекта [[2](http://www.intuit.ru/department/pl/javapl/popup.lit.html#2)].

#### Уникальность

Уникальность - это то, что отличает объект от других объектов. Например, у вас может быть несколько одинаковых монет. Даже если абсолютно все их свойства (атрибуты) одинаковы (год выпуска, номинал и т.д.) и при этом вы можете использовать их независимо друг от друга, они по-прежнему остаются разными монетами.

В машинном представлении под параметром уникальности объекта чаще всего понимается адрес размещения объекта в памяти.

Identity ( уникальность ) объекта состоит в том, что всегда можно определить, указывают две ссылки на один и тот же объект или на разные объекты. При этом два объекта могут во всем быть похожими, их образ в памяти может представляться одинаковыми последовательностями байтов, но, тем не менее, их Identity может быть различна.

Наиболее распространенной ошибкой является понимание уникальности как имени ссылки на объект. Это неверно, т.к. на один объект может указывать несколько ссылок, и ссылки могут менять свои значения (ссылаться на другие объекты).

Итак, уникальность (identity) - свойство объекта; то, что отличает его от других объектов (автор не согласен с переводом русского издания [[2](http://www.intuit.ru/department/pl/javapl/popup.lit.html#2)], поэтому здесь приводится авторский перевод).

|  |
| --- |
| Классы Все монеты из предыдущего примера принадлежат одному и тому же классу объектов (именно с этим связана их одинаковость). Номинальная стоимость монеты, металл, из которого она изготовлена, форма - это атрибуты класса. Совокупность атрибутов и их значений характеризует объект. Наряду с термином "атрибут" часто используют термины "свойство" и "поле", которые в объектно-ориентированном программировании являются синонимами.  Все объекты одного и того же класса описываются одинаковыми наборами атрибутов. Однако объединение объектов в классы определяется не наборами атрибутов, а семантикой. Так, например, объекты "конюшня" и "лошадь" могут иметь одинаковые атрибуты: цена и возраст. При этом они могут относиться к одному классу, если рассматриваются в задаче просто как товар, либо к разным классам, если в рамках поставленной задачи будут использоваться по-разному, т.е. над ними будут совершаться различные действия.  Объединение объектов в классы позволяет рассмотреть задачу в более общей постановке. Класс имеет имя (например, "лошадь"), которое относится ко всем объектам этого класса. Кроме того, в классе вводятся имена атрибутов, которые определены для объектов. В этом смысле описание класса аналогично описанию типа структуры или записи (record), широко применяющихся в процедурном программировании; при этом каждый объект имеет тот же смысл, что и экземпляр структуры (переменная или константа соответствующего типа).  Формально класс - это шаблон поведения объектов определенного типа с заданными параметрами, определяющими состояние. Все экземпляры одного класса ( объекты, порожденные от одного класса ) имеют один и тот же набор свойств и общее поведение, то есть одинаково реагируют на одинаковые сообщения.  http://www.intuit.ru/department/pl/javapl/2/2pic1.gif  В соответствии с UML (Unified Modelling Language - унифицированный язык моделирования), класс имеет следующее графическое представление.  Класс изображается в виде прямоугольника, состоящего из трех частей. В верхней части помещается название класса, в средней - свойства объектов класса, в нижней - действия, которые можно выполнять с объектами данного класса (методы).  Каждый класс также может иметь специальные методы, которые автоматически вызываются при создании и уничтожении объектов этого класса:   * конструктор (constructor) - выполняется при создании объектов ; * деструктор (destructor) - выполняется при уничтожении объектов.   Обычно конструктор и деструктор имеют специальный синтаксис, который может отличаться от синтаксиса, используемого для написания обычных методов класса. Инкапсуляция Инкапсуляция (encapsulation) - это сокрытие реализации класса и отделение его внутреннего представления от внешнего (интерфейса). При использовании объектно-ориентированного подхода не принято применять прямой доступ к свойствам какого-либо класса из методов других классов. Для доступа к свойствам класса принято задействовать специальные методы этого класса для получения и изменения его свойств.  Внутри объекта данные и методы могут обладать различной степенью открытости (или доступности). Степени доступности, принятые в языке Java, подробно будут рассмотрены в лекции 6. Они позволяют более тонко управлять свойством инкапсуляции.  Открытые члены класса составляют внешний интерфейс объекта. Это та функциональность, которая доступна другим классам. Закрытыми обычно объявляются все свойства класса, а также вспомогательные методы, которые являются деталями реализации и от которых не должны зависеть другие части системы.  Благодаря сокрытию реализации за внешним интерфейсом класса можно менять внутреннюю логику отдельного класса, не меняя код остальных компонентов системы. Это свойство называется модульность.  Обеспечение доступа к свойствам класса только через его методы также дает ряд преимуществ. Во-первых, так гораздо проще контролировать корректные значения полей, ведь прямое обращение к свойствам отслеживать невозможно, а значит, им могут присвоить некорректные значения.  Во-вторых, не составит труда изменить способ хранения данных. Если информация станет храниться не в памяти, а в долговременном хранилище, таком как файловая система или база данных, потребуется изменить лишь ряд методов одного класса, а не вводить эту функциональность во все части системы.  Наконец, программный код, написанный с использованием данного принципа, легче отлаживать. Для того, чтобы узнать, кто и когда изменил свойство интересующего нас объекта, достаточно добавить вывод отладочной информации в тот метод объекта, посредством которого осуществляется доступ к свойству этого объекта. При использовании прямого доступа к свойствам объектов программисту пришлось бы добавлять вывод отладочной информации во все участки кода, где используется интересующий нас объект. Наследование Наследование (inheritance) - это отношение между классами, при котором класс использует структуру или поведение другого класса (одиночное наследование ), или других (множественное наследование ) классов. Наследование вводит иерархию "общее/частное", в которой подкласс наследует от одного или нескольких более общих суперклассов. Подклассы обычно дополняют или переопределяют унаследованную структуру и поведение.  В качестве примера можно рассмотреть задачу, в которой необходимо реализовать классы "Легковой автомобиль" и "Грузовой автомобиль". Очевидно, эти два класса имеют общую функциональность. Так, оба они имеют 4 колеса, двигатель, могут перемещаться и т.д. Всеми этими свойствами обладает любой автомобиль, независимо от того, грузовой он или легковой, 5- или 12-местный. Разумно вынести эти общие свойства и функциональность в отдельный класс, например, "Автомобиль" и наследовать от него классы "Легковой автомобиль" и "Грузовой автомобиль", чтобы избежать повторного написания одного и того же кода в разных классах.  http://www.intuit.ru/department/pl/javapl/2/2pic2.gif  Отношение обобщения обозначается сплошной линией с треугольной стрелкой на конце. Стрелка указывает на более общий класс ( класс-предок или суперкласс ), а ее отсутствие - на более специальный класс ( класс-потомок или подкласс ).  Использование наследования способствует уменьшению количества кода, созданного для описания схожих сущностей, а также способствует написанию более эффективного и гибкого кода.  В рассмотренном примере применено одиночное наследование. Некоторый класс также может наследовать свойства и поведение сразу нескольких классов. Наиболее популярным примером применения множественного наследования является проектирование системы учета товаров в зоомагазине.  Все животные в зоомагазине являются наследниками класса "Животное", а также наследниками класса "Товар". Т.е. все они имеют возраст, нуждаются в пище и воде и в то же время имеют цену и могут быть проданы.  Множественное наследование на диаграмме изображается точно так же, как одиночное, за исключением того, что линии наследования соединяют класс-потомок сразу с несколькими суперклассами.  Не все объектно-ориентированные языки программирования содержат языковые конструкции для описания множественного наследования.  В языке Java множественное наследование имеет ограниченную поддержку через интерфейсы и будет рассмотрено в лекции 8. Полиморфизм Полиморфизм является одним из фундаментальных понятий в объектно-ориентированном программировании наряду с наследованием и инкапсуляцией. Слово " полиморфизм " греческого происхождения и означает "имеющий много форм". Чтобы понять, что оно означает применительно к объектно-ориентированному программированию, рассмотрим пример.  Предположим, мы хотим создать векторный графический редактор, в котором нам нужно описать в виде классов набор графических примитивов - Point, Line, Circle, Box и т.д. У каждого из этих классов определим метод draw для отображения соответствующего примитива на экране.  Очевидно, придется написать код, который при необходимости отобразить рисунок будет последовательно перебирать все примитивы, на момент отрисовки находящиеся на экране, и вызывать метод draw у каждого из них. Человек, не знакомый с полиморфизмом, вероятнее всего, создаст несколько массивов (отдельный массив для каждого типа примитивов) и напишет код, который последовательно переберет элементы из каждого массива и вызовет у каждого элемента метод draw. В результате получится примерно следующий код:  ...  //создание пустого массива, который может  // содержать объекты Point с максимальным  // объемом 1000  Point[] p = new Point[1000];  Line[] l = new Line[1000];  Circle[] c = new Circle[1000];  Box[] b = new Box[1000];  ...  // предположим, в этом месте происходит  // заполнение всех массивов соответствующими  // объектами  ...  for(int i = 0; i < p.length;i++) {  //цикл с перебором всех ячеек массива.  //вызов метода draw() в случае,  // если ячейка не пустая.  if(p[i]!=null) p[i].draw();  }  for(int i = 0; i < l.length;i++) {  if(l[i]!=null) l[i].draw();  }  for(int i = 0; i < c.length;i++) {  if(c[i]!=null) c[i].draw();  }  for(int i = 0; i < b.length;i++) {  if(b[i]!=null) b[i].draw();  }  ...  Недостатком написанного выше кода является дублирование практически идентичного кода для отображения каждого типа примитивов. Также неудобно то, что при дальнейшей модернизации нашего графического редактора и добавлении возможности рисовать новые типы графических примитивов, например Text, Star и т.д., при таком подходе придется менять существующий код и добавлять в него определения новых массивов, а также обработку содержащихся в них элементов.  Используя полиморфизм, мы можем значительно упростить реализацию подобной функциональности. Прежде всего, создадим общий родительский класс для всех наших классов. Пусть таким классом будет Point. В результате получим иерархию классов, которая изображена на [рисунке 2.3](http://www.intuit.ru/department/pl/javapl/2/3.html#image.2.3).  У каждого из дочерних классов метод draw переопределен таким образом, чтобы отображать экземпляры каждого класса соответствующим образом.  Для описанной выше иерархии классов, используя полиморфизм, можно написать следующий код:  ...  Point p[] = new Point[1000];  p[0] = new Circle();  p[1] = new Point();  p[2] = new Box();  p[3] = new Line();  ...  for(int i = 0; i < p.length;i++) {  if(p[i]!=null) p[i].draw();  }  ...  В описанном выше примере массив p[] может содержать любые объекты, порожденные от наследников класса Point. При вызове какого-либо метода у любого из элементов этого массива будет выполнен метод того объекта, который содержится в ячейке массива. Например, если в ячейке p[0] находится объект Circle, то при вызове метода draw следующим образом:  p[0].draw()  нарисуется круг, а не точка.  В заключение приведем формальное определение полиморфизма.  Полиморфизм (polymorphism) - положение теории типов, согласно которому имена (например, переменных) могут обозначать объекты разных (но имеющих общего родителя) классов. Следовательно, любой объект, обозначаемый полиморфным именем, может по-своему реагировать на некий общий набор операций [[2](http://www.intuit.ru/department/pl/javapl/popup.lit.html#2)].  В процедурном программировании тоже существует понятие полиморфизма, которое отличается от рассмотренного механизма в ООП. Процедурный полиморфизм предполагает возможность создания нескольких процедур или функций с одним и тем же именем, но разным количеством или различными типами передаваемых параметров. Такие одноименные функции называются перегруженными, а само явление - перегрузкой ( overloading ). Перегрузка функций существует и в ООП и называется перегрузкой методов.  Пример иерархии классов.  **Рис. 2.3.**  Пример иерархии классов.  Примером использования перегрузки методов в языке Java может служить класс PrintWriter, который применяется, в частности, для вывода сообщений на консоль. Этот класс имеет множество методов println, которые различаются типами и/или количеством входных параметров. Вот лишь несколько из них:  void println()  // переход на новую строку  void println(boolean x)  // выводит значение булевской  // переменной (true или false)  void println(String x)  // выводит строку - значение  // текстового параметра.  Определенные сложности возникают при вызове перегруженных методов. В Java существуют специальные правила, которые позволяют решать эту проблему. Они будут рассмотрены в соответствующей лекции. |
| http://www.intuit.ru/img/empty.gif |
| http://www.intuit.ru/img/empty.gif |

### Типы отношений между классами

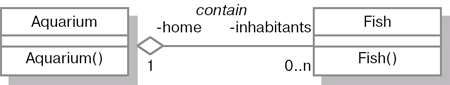
Как правило, любая программа, написанная на объектно-ориентированном языке, представляет собой некоторый набор связанных между собой классов. Можно провести аналогию между написанием программы и строительством дома. Подобно тому, как стена складывается из кирпичей, компьютерная программа с использованием ООП строится из классов. Причем эти классы должны иметь представление друг о друге, для того чтобы сообща выполнять поставленную задачу.

Возможны следующие связи между классами в рамках объектной модели (приводятся лишь наиболее простые и часто используемые виды связей, подробное их рассмотрение выходит за рамки этой ознакомительной лекции):

* агрегация ( Aggregation );
* ассоциация ( Association );
* наследование ( Inheritance );
* метаклассы ( Metaclass ).

#### Агрегация

Отношение между классами типа "содержит" (contain) или "состоит из" называется агрегацией, или включением. Например, если аквариум наполнен водой и в нем плавают рыбки, то можно сказать, что аквариум агрегирует в себе воду и рыбок.



Такое отношение включения, или агрегации (aggregation), изображается линией с ромбиком на стороне того класса, который выступает в качестве владельца, или контейнера. Необязательное название отношения записывается посередине линии.

В нашем примере отношение contain является двунаправленным. Объект класса Aquarium содержит несколько объектов Fish. В то же время каждая рыбка "знает", в каком именно аквариуме она живет. Каждый класс имеет свою роль в агрегации, которая указывает, какое место занимает класс в данном отношении. Имя роли не является обязательным элементом обозначений и может отсутствовать на диаграмме. В примере можно видеть роль home класса Aquarium (аквариум является домом для рыбок), а также роль inhabitants класса Fish (рыбки являются обитателями аквариума). Название роли обычно совпадает с названием соответствующего поля в классе. Изображение такого поля на диаграмме излишне, если уже указано имя роли. Т.е. в данном случае класс Aquarium будет иметь свойство (поле) inhabitants, а класс Fish - свойство home.

Число объектов, участвующих в отношении, записывается рядом с именем роли. Запись " 0..n " означает "от нуля до бесконечности". Приняты также обозначения:

* " 1..n " - от единицы до бесконечности;
* " 0 " - ноль;
* " 1 " - один;
* " n " - фиксированное количество;
* " 0..1 " - ноль или один.

Код, описывающий рассмотренную модель и явление агрегации, может выглядеть, например, следующим образом:

// определение класса Fish

public class Fish {

// определения поля home

// (ссылка на объект Aquarium)

private Aquarium home;

public Fish() {

}

}

// определение класса Aquarium

public class Aquarium {

// определения поля inhabitants

// (массив ссылок на объекты Fish)

private Fish inhabitants[];

public Aquarium() {

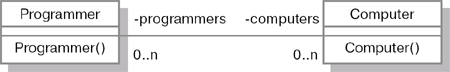
}

}

#### Ассоциация

Если объекты одного класса ссылаются на один или более объектов другого класса, но ни в ту, ни в другую сторону отношение между объектами не носит характера "владения", или контейнеризации, такое отношение называют ассоциацией (association). Отношение ассоциации изображается так же, как и отношение агрегации, но линия, связывающая классы,- простая, без ромбика.

В качестве примера можно рассмотреть программиста и его компьютер. Между этими двумя объектами нет агрегации, но существует четкая взаимосвязь. Так, всегда можно установить, за какими компьютерами работает какой-либо программист, а также какие люди пользуются отдельно взятым компьютером. В рассмотренном примере имеет место ассоциация "многие-ко-многим".



В данном случае между экземплярами классов Programmer и Computer в обе стороны используется отношение " 0..n ", т.к. программист, в принципе, может не работать с компьютером (если он теоретик или на пенсии). В свою очередь, компьютер может никем не использоваться (если он новый и еще не установлен).

Код, соответствующий рассмотренному примеру, будет, например, следующим:

public class Programmer {

private Computer computers[];

public Programmer() {

}

}

public class Computer {

private Programmer programmers[];

public Computer() {

}

}

#### Наследование

Наследование является важным случаем отношений между двумя или более классами. Подробно оно рассматривалось выше.

#### Метаклассы

Итак, любой объект имеет структуру, состоящую из полей и методов. Объекты, имеющие одинаковую структуру и семантику, описываются одним классом, который и является, по сути, определением структуры объектов, порожденных от него.

В свою очередь, каждый класс, или описание, всегда имеет строгий шаблон, задаваемый языком программирования или выбранной объектной моделью. Он определяет, например, допустимо ли множественное наследование, какие существуют ограничения на именование классов, как описываются поля и методы, набор существующих типов данных и многое другое. Таким образом, класс можно рассматривать как объект, у которого есть свойства (имя, список полей и их типы, список методов, список аргументов для каждого метода и т.д.). Также класс может обладать поведением, то есть поддерживать методы. А раз для любого объекта существует шаблон, описывающий свойства и поведение этого объекта, значит, его можно определить и для класса. Такой шаблон, задающий различные классы, называется метаклассом.

Чтобы представить себе, что такое метакласс, рассмотрим пример некой бюрократической организации. Будем считать, что все классы в такой системе представляют собой строгие инструкции, которые описывают, что нужно сделать, чтобы породить новый объект (например, нанять нового служащего или открыть новый отдел). Как и полагается классам, они описывают все свойства новых объектов (например, зарплату и профессиональный уровень для сотрудников, площадь и имущество для отделов) и их поведение (обязанности служащих и функции подразделений).

В свою очередь, написание новой инструкции можно строго регламентировать. Скажем, необходимо использовать специальный бланк, придерживаться правил оформления и заполнить все обязательные поля (например, номер инструкции и фамилии ответственных работников). Такая "инструкция инструкций" и будет представлять собой метакласс в ООП.

Итак, объекты порождаются от классов, а классы - от метакласса. Он, как правило, в системе только один. Но существуют языки программирования, в которых можно создавать и использовать собственные метаклассы, например язык Python. В частности, функциональность метакласса может быть следующая: при формировании класса он будет просматривать список всех методов в классе и, если имя метода имеет вид set\_XXX или get\_XXX, автоматически создавать поле с именем XXX, если такого не существует.

Поскольку метакласс сам является классом, то нет никакого смысла в создании "мета-мета-классов".

В языке Java также есть метакласс. Это класс, который так и называется - Class (описывает классы ), он располагается в основной библиотеке java.lang. Виртуальная машина использует его по прямому назначению. Когда загружается очередной .class -файл, содержащий описание нового класса, JVM порождает объект класса Class, который будет хранить его структуру. Таким образом, Java использует концепцию метакласса в самых практических целях. С помощью Class реализована поддержка статических ( static ) полей и методов. Наконец, этот класс содержит ряд методов, полезных для разработчиков. Они будут рассмотрены в следующих лекциях.

**Достоинства ООП**

От любой методики разработки программного обеспечения мы ждем, что она поможет нам в решении наших задач. Но одной из самых значительных проблем проектирования является сложность. Чем больше и сложнее программная система, тем важнее разбить ее на небольшие, четко очерченные части. Чтобы справиться со сложностью, необходимо абстрагироваться от деталей. В этом смысле классы представляют собой весьма удобный инструмент.

* Классы позволяют проводить конструирование из полезных компонентов, обладающих простыми инструментами, что позволяет абстрагироваться от деталей реализации.
* Данные и операции над ними образуют определенную сущность, и они не разносятся по всей программе, как нередко бывает в случае процедурного программирования, а описываются вместе. Локализация кода и данных улучшает наглядность и удобство сопровождения программного обеспечения.
* Инкапсуляция позволяет привнести свойство модульности, что облегчает распараллеливание выполнения задачи между несколькими исполнителями и обновление версий отдельных компонентов.

ООП дает возможность создавать расширяемые системы. Это одно из основных достоинств ООП, и именно оно отличает данный подход от традиционных методов программирования. Расширяемость означает, что существующую систему можно заставить работать с новыми компонентами, причем без внесения в нее каких-либо изменений. Компоненты могут быть добавлены на этапе исполнения программы.

Полиморфизм оказывается полезным преимущественно в следующих ситуациях.

* Обработка разнородных структур данных. Программы могут работать, не различая вида объектов, что существенно упрощает код. Новые виды могут быть добавлены в любой момент.
* Изменение поведения во время исполнения. На этапе исполнения один объект может быть заменен другим, что позволяет легко, без изменения кода, адаптировать алгоритм в зависимости от того, какой используется объект.
* Реализация работы с наследниками. Алгоритмы можно обобщить настолько, что они уже смогут работать более чем с одним видом объектов.
* Создание "каркаса" (framework). Независимые от приложения части предметной области могут быть реализованы в виде набора универсальных классов, или каркаса (framework), и в дальнейшем расширены за счет добавления частей, специфичных для конкретного приложения.

Часто многоразового использования программного обеспечения не удается добиться из-за того, что существующие компоненты уже не отвечают новым требованиям. ООП помогает этого достичь без нарушения работы уже имеющихся компонентов, что позволяет извлечь максимум из многоразового использования компонентов.

* Сокращается время на разработку, которое может быть отдано другим задачам.
* Компоненты многоразового использования обычно содержат гораздо меньше ошибок, чем вновь разработанные, ведь они уже не раз подвергались проверке.
* Когда некий компонент используется сразу несколькими клиентами, улучшения, вносимые в его код, одновременно оказывают положительное влияние и на множество работающих с ним программ.
* Если программа опирается на стандартные компоненты, ее структура и пользовательский интерфейс становятся более унифицированными, что облегчает ее понимание и упрощает использование.

**Недостатки ООП**

Документирование классов - задача более трудная, чем это было в случае процедур и модулей. Поскольку любой метод может быть переопределен, в документации должно говориться не только о том, что делает данный метод, но и о том, в каком контексте он вызывается. Ведь переопределенные методы обычно вызываются не клиентом, а самим каркасом. Таким образом, программист должен знать, какие условия выполняются, когда вызывается данный метод. Для абстрактных методов, которые пусты, в документации должно говориться о том, для каких целей предполагается использовать переопределяемый метод.

В сложных иерархиях классов поля и методы обычно наследуются с разных уровней. И не всегда легко определить, какие поля и методы фактически относятся к данному классу. Для получения такой информации нужны специальные инструменты, вроде навигаторов классов. Если конкретный класс расширяется, то каждый метод обычно сокращают перед передачей сообщения базовому классу. Реализация операции, таким образом, рассредотачивается по нескольким классам, и чтобы понять, как она работает, нам приходится внимательно просматривать весь код.

Методы, как правило, короче процедур, поскольку они осуществляют только одну операцию над данными, зато их намного больше. В коротких методах легче разобраться, но они неудобны тем, что код для обработки сообщения иногда "размазан" по многим маленьким методам.

Инкапсуляцией данных не следует злоупотреблять. Чем больше логики и данных скрыто в недрах класса, тем сложнее его расширять. Отправной точкой здесь должно быть не то, что клиентам не разрешается знать о тех или иных данных, а то, что клиентам для работы с классом этих данных знать не требуется.

Многие считают, что ООП является неэффективным. Как же обстоит дело в действительности? Мы должны проводить четкую грань между неэффективностью на этапе выполнения, неэффективностью в смысле распределения памяти и неэффективностью, связанной с излишней универсализацией.

1. Неэффективность на этапе выполнения. В языках типа Smalltalk сообщения интерпретируются во время выполнения программы путем осуществления их поиска в одной или нескольких таблицах и за счет выбора подходящего метода. Конечно, это медленный процесс. И даже при использовании наилучших методов оптимизации Smalltalk-программы в десять раз медленнее оптимизированных C-программ.

В гибридных языках типа Oberon-2, Object Pascal и C++ отправка сообщения приводит лишь к вызову через указатель процедурной переменной. На некоторых машинах сообщения выполняются лишь на 10% медленнее, чем обычные процедурные вызовы. И поскольку сообщения встречаются в программе гораздо реже других операций, их воздействие на время выполнения влияния практически не оказывает.

Однако существует другой фактор, который влияет на время выполнения: это инкапсуляция данных. Рекомендуется не предоставлять прямой доступ к полям класса, а выполнять каждую операцию над данными через методы. Такая схема приводит к необходимости выполнения процедурного вызова каждый раз при доступе к данным. Однако если инкапсуляция используется только там, где она необходима (т.е. в тех случаях, когда это становится преимуществом), то замедление вполне приемлемое.

1. Неэффективность в смысле распределения памяти. Динамическое связывание и проверка типа на этапе выполнения требуют по ходу работы информации о типе объекта. Такая информация хранится в дескрипторе типа и он выделяется один на класс. Каждый объект имеет невидимый указатель на дескриптор типа для своего класса. Таким образом, в объектно-ориентированных программах необходимая дополнительная память выражается в одном указателе для объекта и в одном дескрипторе типа для класса.
2. Излишняя универсальность. Неэффективность также может означать, что в программе реализованы избыточные возможности. В библиотечном классе часто содержится больше методов, чем это реально необходимо. А поскольку лишние методы не могут быть удалены, они становятся мертвым грузом. Это не влияет на время выполнения, но сказывается на размере кода.

Одно из возможных решений - строить базовый класс с минимальным числом методов, а затем уже реализовывать различные расширения этого класса, которые позволят нарастить функциональность. Другой подход - дать компоновщику возможность удалять лишние методы. Такие интеллектуальные компоновщики уже существуют для различных языков и операционных систем.

Но нельзя утверждать, что ООП неэффективно. Если классы используются лишь там, где это действительно необходимо, то потеря эффективности из-за повышенного расхода памяти и меньшей производительности незначительна. Кроме того, надежность программного обеспечения и быстрота его написания часто бывает важнее, чем производительность.

**Заключение**

В этой лекции мы рассказали об объектно-ориентированном подходе к разработке ПО, а также о том, что послужило предпосылками к его появлению и сделало его популярным. Были рассмотрены ключевые понятия ООП - объект и класс. Далее были описаны основные свойства объектной модели - инкапсуляция, наследование, полиморфизм. Основными видами отношений между классами являются наследование, ассоциация, агрегация, метакласс. Также были описаны правила изображения классов и связей между ними на языке UML.

### Кодировка

Технология Java, как платформа, изначально спроектированная для Глобальной сети Internet, должна быть многоязыковой, а значит, обычный набор символов ASCII (American Standard Code for Information Interchange, Американский стандартный код обмена информацией), включающий в себя лишь латинский алфавит, цифры и простейшие специальные знаки (скобки, знаки препинания, арифметические операции и т.д.), недостаточен. Поэтому для записи текста программы применяется более универсальная кодировка Unicode UTF-16.

Как известно, Unicode UTF-16 представляет символы кодом из 2 байт, описывая, таким образом, 65535 символов. Это позволяет поддерживать практически все распространенные языки мира. Первые 128 символов совпадают с набором ASCII. Однако понятно, что требуется некоторое специальное обозначение, чтобы иметь возможность задавать в программе любой символ Unicode, ведь никакая клавиатура не позволяет вводить более 65 тысяч различных знаков. Эта конструкция представляет символ Unicode, используя только символы ASCII. Например, если в программу нужно вставить знак с кодом 6917, необходимо его представить в шестнадцатеричном формате (1B05) и записать:

\u1B05,

причем буква u должна быть строчной, а шестнадцатеричные цифры A, B, C, D, E, F можно использовать произвольно, как заглавные, так и строчные. Таким образом можно закодировать все символы Unicode от \u0000 до \uFFFF. Буквы русского алфавита начинаются с \u0410 (только буква Ё имеет код \u0401 ) по \u044F (код буквы ё \u0451 ). В последних версиях JDK в состав демонстрационных приложений и апплетов входит небольшая программа SymbolTest, позволяющая просматривать весь набор символов Unicode. Ее аналог несложно написать самостоятельно. Для перекодирования больших текстов служит утилита native2ascii, также входящая в JDK. Она может работать как в прямом режиме — переводить из разнообразных кодировок в Unicode, записанный ASCII -символами, так и в обратном (опция -reverse ) — из Unicode в стандартную кодировку операционной системы.

В версиях языка Java до 1.1 применялся Unicode версии 1.1.5, в последнем выпуске 1.4 используется 3.0. Таким образом, Java следит за развитием стандарта и базируется на современных версиях. Для любой JDK точную версию Unicode, используемую в ней, можно узнать из документации к классу Character. Официальный web-сайт стандарта, где можно получить дополнительную информацию,— <http://www.unicode.org/>.

Итак, используя простейшую кодировку ASCII, можно ввести произвольную последовательность символов Unicode. Далее будет показано, что Unicode используется не для всех лексем, а только для тех, для которых важна поддержка многих языков, а именно: комментарии, идентификаторы, символьные и строковые литералы. Для записи остальных лексем вполне достаточно ASCII -символов.

### Анализ программы

Компилятор, анализируя программу, сразу разделяет ее на:

* пробелы (white spaces);
* комментарии (comments);
* основные лексемы (tokens).

#### Пробелы

Пробелами в данном случае называют все символы, разбивающие текст программы на лексемы. Это как сам символ пробела (space, \u0020, десятичный код 32), так и знаки табуляции и перевода строки. Они используются для разделения лексем, а также для оформления кода, чтобы его было легче читать. Например, следующую часть программы (вычисление корней квадратного уравнения):

double a = 1, b = 1, c = 6;

double D = b \* b - 4 \* a \* c;

if (D >= 0) {

double x1 = (-b + Math.sqrt (D)) / (2 \* a);

double x2 = (-b - Math.sqrt (D)) / (2 \* a);

}

можно записать и в таком виде:

double a=1,b=1,c=6;double D=b\*b-4\*a\*c;if(D>=0)

{double x1=(-b+Math.sqrt(D))/(2\*a);double

x2=(-b-Math.sqrt(D))/(2\*a);}

В обоих случаях компилятор сгенерирует абсолютно одинаковый код. Единственное соображение, которым должен руководствоваться разработчик,— легкость чтения при дальнейшей поддержке такого кода.

Для разбиения текста на строки в ASCII используется два символа - "возврат каретки" ( carriage return, CR, \u000d, десятичный код 13) и символ новой строки ( linefeed, LF, \u000a, десятичный код 10). Чтобы не зависеть от особенностей используемой платформы, в Java применяется наиболее гибкий подход. Завершением строки считается:

* ASCII -символ LF, символ новой строки;
* ASCII -символ CR, "возврат каретки";
* символ CR, за которым сразу же следует символ LF.

Разбиение на строки важно для корректного разбиения на лексемы (как уже говорилось, завершение строки также служит разделителем между лексемами), для правильной работы со строковыми комментариями (см. следующую тему "Комментарии"), а также для вывода отладочной информации (при выводе ошибок компиляции и времени исполнения указывается, на какой строке исходного кода они возникли). Итак, пробелами в Java считаются:

* ASCII -символ SP, space, пробел, \u0020, десятичный код 32;
* ASCII -символ HT, horizontal tab, символ горизонтальной табуляции, \u0009, десятичный код 9;
* ASCII -символ FF, form feed, символ перевода страницы (был введен для работы с принтером), \u000c, десятичный код 12;
* завершение строки.

#### Комментарии

Комментарии не влияют на результирующий бинарный код и используются только для ввода пояснений к программе.

В Java комментарии бывают двух видов:

* строчные
* блочные

Строчные комментарии начинаются с ASCII -символов // и длятся до конца текущей строки. Как правило, они используются для пояснения именно этой строки, например:

int y=1970; // год рождения

Блочные комментарии располагаются между ASCII -символами /\* и \*/, могут занимать произвольное количество строк, например:

/\*

Этот цикл не может начинаться с нуля

из-за особенностей алгоритма

\*/

for (int i=1; i<10; i++) {

...

}

Часто блочные комментарии оформляют следующим образом (каждая строка начинается с \* ):

/\*

\* Описание алгоритма работы

\* следующего цикла while

\*/

while (x > 0) {

...

}

Блочный комментарий не обязательно должен располагаться на нескольких строках, он может даже находиться в середине оператора:

float s = 2\*Math.PI/\*getRadius()\*/;

// Закомментировано для отладки

В этом примере блочный комментарий разбивает арифметические операции. Выражение Math.PI предоставляет значение константы PI, определенное в классе Math. Вызов метода getRadius() теперь закомментирован и не будет произведен, переменная s всегда будет принимать значение 2 PI. Завершает строку строчный комментарий.

Комментарии не могут находиться в символьных и строковых литералах, идентификаторах (эти понятия подробно рассматриваются далее в этой лекции). Следующий пример содержит случаи неправильного применения комментариев:

// В этом примере текст /\*…\*/ станет просто

// частью строки s

String s = "text/\*just text\*/";

/\*

Следующая строка станет причиной ошибки

при компиляции, так как комментарий разбил

имя метода getRadius()

\*/

circle.get/\*comment\*/Radius();

А такой код допустим:

// Комментарий может разделять вызовы функций:

circle./\*comment\*/getRadius();

// Комментарий может заменять пробелы:

int/\*comment\*/x=1;

В последней строке между названием типа данных int и названием переменной x обязательно должен быть пробел или, как в данном примере, комментарий.

Комментарии не могут быть вложенными. Символы /\*, \*/, // не имеют никакого особенного значения внутри уже открытых комментариев, как строчных, так и блочных. Таким образом, в примере

/\* начало комментария /\* // /\*\* завершение: \*/

описан только один блочный комментарий. А в следующем примере (строки кода пронумерованы для удобства)

1. /\*

2. comment

3. /\*

4. more comments

5. \*/

6. finish

7. \*/

компилятор выдаст ошибку. Блочный комментарий начался в строке 1 с комбинации символов /\*. Вторая открывающая комбинация /\* на строке 3 будет проигнорирована, так как находится уже внутри комментария. Символы \*/ в строке 5 завершат его, а строка 7 породит ошибку – попытка закрыть комментарий, который не был начат.

Любые комментарии полностью удаляются из программы во время компиляции, поэтому их можно использовать неограниченно, не опасаясь, что это повлияет на бинарный код. Основное их предназначение - сделать программу простой для понимания, в том числе и для других разработчиков, которым придется в ней разбираться по какой-либо причине. Также комментарии зачастую используются для временного исключения частей кода, например:

int x = 2;

int y = 0;

/\*

if (x > 0)

y = y + x\*2;

else

y = -y - x\*4;

\*/

y = y\*y;// + 2\*x;

В этом примере закомментировано выражение if-else и оператор сложения +2\*x.

Как уже говорилось выше, комментарии можно писать символами Unicode, то есть на любом языке, удобном разработчику.

Кроме этого, существует особый вид блочного комментария – комментарий разработчика. Он применяется для автоматического создания документации кода. В стандартную поставку JDK, начиная с версии 1.0, входит специальная утилита javadoc. На вход ей подается исходный код классов, а на выходе получается удобная документация в HTML-формате, которая описывает все классы, все их поля и методы. При этом активно используются гиперссылки, что существенно упрощает изучение программы (например, читая описание метода, можно с помощью одного нажатия мыши перейти на описание типов, используемых в качестве аргументов или возвращаемого значения). Однако понятно, что одного названия метода и перечисления его аргументов недостаточно для понимания его работы. Необходимы дополнительные пояснения от разработчика.

Комментарий разработчика записывается так же, как и блочный. Единственное различие в начальной комбинации символов – для документации комментарий необходимо начинать с /\*\*. Например:

/\*\*

\* Вычисление модуля целого числа.

\* Этот метод возвращает

\* абсолютное значение аргумента x.

\*/

int getAbs(int x) {

if (x>=0)

return x;

else

return -x;

}

Первое предложение должно содержать краткое резюме всего комментария. В дальнейшем оно будет использовано как пояснение этой функции в списке всех методов класса (ниже будут описаны все конструкции языка, для которых применяется комментарий разработчика).

Поскольку в результате создается HTML-документация, то и комментарий необходимо писать по правилам HTML. Допускается применение тегов, таких как <b> и <p>. Однако теги заголовков с <h1> по <h6> и <hr> использовать нельзя, так как они активно применяются javadoc для создания структуры документации.

Символ \* в начале каждой строки и предшествующие ему пробелы и знаки табуляции игнорируются. Их можно не использовать вообще, но они удобны, когда необходимо форматирование, скажем, в примерах кода.

/\*\*

\* Первое предложение - краткое

\* описание метода.

\* <p>

\* Так оформляется пример кода:

\* <blockquote>

\* <pre>

\* if (condition==true) {

\* x = getWidth();

\* y = x.getHeight();

\* }

\* </pre></blockquote>

\* А так описывается HTML-список:

\* <ul>

\* <li>Можно использовать наклонный шрифт

\* <i>курсив</i>,

\* <li>или жирный <b>жирный</b>.

\* </ul>

\*/

public void calculate (int x, int y) {

...

}

Из этого комментария будет сгенерирован HTML-код, выглядящий примерно так:

Первое предложение – краткое описание метода.

Так оформляется пример кода:

if (condition==true) {

x = getWidth();

y = x.getHeight();

}

А так описывается HTML-список:

\* Можно использовать наклонный шрифт курсив,

\* или жирный жирный.

Наконец, javadoc поддерживает специальные теги. Они начинаются с символа @. Подробное описание этих тегов можно найти в документации. Например, можно использовать тег @see, чтобы сослаться на другой класс, поле или метод, или даже на другой Internet-сайт.

/\*\*

\* Краткое описание.

\*

\* Развернутый комментарий.

\*

\* @see java.lang.String

\* @see java.lang.Math#PI

\* @see <a href="java.sun.com">Official

\* Java site</a>

\*/

Первая ссылка указывает на класс String ( java.lang – название библиотеки, в которой находится этот класс), вторая – на поле PI класса Math (символ # разделяет название класса и его полей или методов), третья ссылается на официальный сайт Java.

Комментарии разработчика могут быть записаны перед объявлением классов, интерфейсов, полей, методов и конструкторов. Если записать комментарий /\*\* … \*/ в другой части кода, то ошибки не будет, но он не попадет в документацию, генерируемую javadoc. Кроме того, можно описать пакет (так называются библиотеки, или модули, в Java). Для этого необходимо создать специальный файл package.html, сохранить в нем комментарий и поместить его в каталог пакета. HTML-текст, содержащийся между тегами <body> и </body>, будет помещен в документацию, а первое предложение будет использоваться для краткой характеристики этого пакета.

#### Лексемы

Итак, мы рассмотрели пробелы (в широком смысле этого слова, т.е. все символы, отвечающие за форматирование текста программы) и комментарии, применяемые для ввода пояснений к коду. С точки зрения программиста они применяются для того, чтобы сделать программу более читаемой и понятной для дальнейшего развития.

С точки зрения компилятора, а точнее его части, отвечающей за лексический разбор, основная роль пробелов и комментариев – служить разделителями между лексемами, причем сами разделители далее отбрасываются и на компилированный код не влияют. Например, все следующие примеры объявления переменной эквивалентны:

// Используем пробел в качестве разделителя.

int x = 3 ;

// здесь разделителем является перевод строки

int

x

=

3

;

// здесь разделяем знаком табуляции

int x = 3 ;

/\*

\* Единственный принципиально необходимый

\* разделитель между названием типа данных

\* int и именем переменной x здесь описан

\* комментарием блочного типа.

\*/

int/\*\*/x=3;

Конечно, лексемы очень разнообразны, и именно они определяют многие свойства языка. Рассмотрим все их виды более подробно.

### Виды лексем

Ниже перечислены все виды лексем в Java:

* идентификаторы (identifiers);
* ключевые слова (key words);
* литералы (literals);
* разделители (separators);
* операторы (operators).

Рассмотрим их по отдельности.

#### Идентификаторы

Идентификаторы – это имена, которые даются различным элементам языка для упрощения доступа к ним. Имена имеют пакеты, классы, интерфейсы, поля, методы, аргументы и локальные переменные (все эти понятия подробно рассматриваются в следующих лекциях). Идентификаторы можно записывать символами Unicode, то есть на любом удобном языке. Длина имени не ограничена.

Идентификатор состоит из букв и цифр. Имя не может начинаться с цифры. Java-буквы, используемые в идентификаторах, включают в себя ASCII -символы A-Z ( \u0041 - \u005a ), a-z ( \u0061 - \u007a ), а также знаки подчеркивания \_ ( ASCII underscore, \u005f ) и доллара $ ( \u0024 ). Знак доллара используется только при автоматической генерации кода (чтобы исключить случайное совпадение имен), либо при использовании каких-либо старых библиотек, в которых допускались имена с этим символом. Java-цифры включают в себя обычные ASCII -цифры 0-9 ( \u0030 - \u0039 ).

Для идентификаторов не допускаются совпадения с зарезервированными словами (это ключевые слова, булевские литералы true и false и null- литерал null ). Конечно, если 2 идентификатора включают в себя разные буквы, которые одинаково выглядят (например, латинская и русская буквы A ), то они считаются различными.

В этой лекции уже применялись следующие идентификаторы:

Character, a, b, c, D, x1, x2, Math, sqrt, x,

y, i, s, PI, getRadius, circle, getAbs,

calculate, condition, getWidth, getHeight,

java, lang, String

Также допустимыми являются идентификаторы:

Computer, COLOR\_RED, \_, aVeryLongNameOfTheMethod

#### Ключевые слова

Ключевые слова – это зарезервированные слова, состоящие из ASCII -символов и выполняющие различные задачи языка. Вот их полный список (48 слов):

abstract double int strictfp

boolean else interface super

break extends long switch

byte final native synchronized

case finally new this

catch float package throw

char for private throws

class goto protected transient

const if public try

continue implements return void

default import short volatile

do instanceof static while

Ключевые слова goto и const зарезервированы, но не используются. Это сделано для того, чтобы компилятор мог правильно отреагировать на их использование в других языках. Напротив, оба булевских литерала true, false и null- литерал null часто считают ключевыми словами (возможно, потому, что многие средства разработки подсвечивают их таким же образом), однако это именно литералы.

Значение всех ключевых слов будет рассматриваться в следующих лекциях.

#### Литералы

Литералы позволяют задать в программе значения для числовых, символьных и строковых выражений, а также null- литералов . Всего в Java определено 6 видов литералов:

* целочисленный (integer);
* дробный (floating-point);
* булевский (boolean);
* символьный (character);
* строковый (string);
* null- литерал (null-literal).

Рассмотрим их по отдельности.

##### Целочисленные литералы

Целочисленные литералы позволяют задавать целочисленные значения в десятичном, восьмеричном и шестнадцатеричном виде. Десятичный формат традиционен и ничем не отличается от правил, принятых в других языках. Значения в восьмеричном виде начинаются с нуля, и, конечно, использование цифр 8 и 9 запрещено. Запись шестнадцатеричных чисел начинается с 0x или 0X (цифра 0 и латинская ASCII -буква X в произвольном регистре). Таким образом, ноль можно записать тремя различными способами:

0

0X0

0x0

Как обычно, для записи цифр 10 - 15 в шестнадцатеричном формате используются буквы A, B, C, D, E, F, прописные или строчные. Примеры таких литералов:

0xaBcDeF, 0xCafe, 0xDEC

Типы данных рассматриваются ниже, однако здесь необходимо упомянуть два целочисленных типа int и long длиной 4 и 8 байт, соответственно (или 32 и 64 бита, соответственно). Оба эти типа знаковые, т.е. тип int хранит значения от -231 до 231-1, или от -2.147.483.648 до 2.147.483.647. По умолчанию целочисленный литерал имеет тип int, а значит, в программе допустимо использовать литералы только от 0 до 2147483648, иначе возникнет ошибка компиляции. При этом литерал 2147483648 можно использовать только как аргумент унарного оператора - :

int x = -2147483648; \\ верно

int y = -5-2147483648; \\ здесь возникнет

\\ ошибка компиляции

Соответственно, допустимые литералы в восьмеричной записи должны быть от 00 до 017777777777 ( =231-1 ), с унарным оператором - допустимо также -020000000000 ( = -231 ). Аналогично для шестнадцатеричного формата – от 0x0 до 0x7fffffff ( =231-1 ), а также -0x80000000 ( = -231 ).

Тип long имеет длину 64 бита, а значит, позволяет хранить значения от -263 до 263-1. Чтобы ввести такой литерал, необходимо в конце поставить латинскую букву L или l, тогда все значение будет трактоваться как long. Аналогично можно выписать максимальные допустимые значения для них:

9223372036854775807L

0777777777777777777777L

0x7fffffffffffffffL

// наибольшие отрицательные значения:

-9223372036854775808L

-01000000000000000000000L

-0x8000000000000000L

Другие примеры целочисленных литералов типа long:

0L, 123l, 0xC0B0L

##### Дробные литералы

Дробные литералы представляют собой числа с плавающей десятичной точкой. Правила записи таких чисел такие же, как и в большинстве современных языков программирования.

Примеры:

3.14

2.

.5

7e10

3.1E-20

Таким образом, дробный литерал состоит из следующих составных частей:

* целая часть;
* десятичная точка (используется ASCII -символ точка);
* дробная часть;
* порядок (состоит из латинской ASCII -буквы E в произвольном регистре и целого числа с опциональным знаком + или - );
* окончание-указатель типа.

Целая и дробная части записываются десятичными цифрами, а указатель типа (аналог указателя L или l для целочисленных литералов типа long ) имеет два возможных значения – латинская ASCII -буква D (для типа double ) или F (для типа float ) в произвольном регистре. Они будут подробно рассмотрены ниже.

Необходимыми частями являются:

* хотя бы одна цифра в целой или дробной части;
* десятичная точка или показатель степени, или указатель типа.

Все остальные части необязательные. Таким образом, "минимальные" дробные литералы могут быть записаны, например, так:

1.

.1

1e1

1f

В Java есть два дробных типа, упомянутые выше, – float и double. Их длина – 4 и 8 байт или 32 и 64 бита, соответственно. Дробный литерал имеет тип float, если он заканчивается на латинскую букву F в произвольном регистре. В противном случае он рассматривается как значение типа double и может включать в себя окончание D или d, как признак типа double (используется только для наглядности).

// float-литералы:

1f, 3.14F, 0f, 1e+5F

// double-литералы:

0., 3.14d, 1e-4, 31.34E45D

В Java дробные числа 32-битного типа float и 64-битного типа double хранятся в памяти в бинарном виде в формате, стандартизированном спецификацией IEEE 754 (полное название – IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985 (IEEE, New York)). В этой спецификации описаны не только конечные дробные величины, но и еще несколько особых значений, а именно:

* положительная и отрицательная бесконечности (positive/negative infinity);
* значение "не число", Not-a-Number, сокращенно NaN;
* положительный и отрицательный нули.

Для этих значений нет специальных обозначений. Чтобы получить такие величины, необходимо либо произвести арифметическую операцию (например, результатом деления ноль на ноль 0.0/0.0 является NaN ), либо обратиться к константам в классах Float и Double, а именно POSITIVE\_INFINITY, NEGATIVE\_INFINITY и NaN. Более подробно работа с этими особенными значениями рассматривается в следующей лекции.

Типы данных накладывают ограничения на возможные значения литералов, как и для целочисленных типов. Максимальное положительное конечное значение дробного литерала:

* для float: 3.40282347e+38f
* для double: 1.79769313486231570e+308

Кроме того, для дробных величин становится важным еще одно предельное значение – минимальное положительное ненулевое значение:

* для float: 1.40239846e-45f
* для double: 4.94065645841246544e-324

Попытка указать литерал со слишком большим абсолютным значением (например, 1e40F ) приведет к ошибке компиляции. Такая величина должна представляться бесконечностью. Аналогично, указание литерала со слишком малым ненулевым значением (например, 1e-350 ) также приводит к ошибке. Это значение должно быть округлено до нуля. Однако если округление приводит не к нулю, то компилятор произведет его сам:

// ошибка, выражение должно быть округлено до 0

0.00000000000000000000000000000000000000000001f

// ошибки нет, компилятор сам округляет до 1

1.00000000000000000000000000000000000000000001f

Стандартных возможностей вводить дробные значения не в десятичной системе в Java нет, однако классы Float и Double предоставляют много вспомогательных методов, в том числе и для такой задачи.

##### Логические литералы

Логические литералы имеют два возможных значения – true и false. Эти два зарезервированных слова не являются ключевыми, но также не могут использоваться в качестве идентификатора.

##### Символьные литералы

Символьные литералы описывают один символ из набора Unicode, заключенный в одиночные кавычки, или апострофы ( ASCII -символ single quote, \u0027 ). Например:

'a' // латинская буква а

' ' // пробел

'K' // греческая буква каппа

Также допускается специальная запись для описания символа через его код (см. тему "Кодировка"). Примеры:

'\u0041' // латинская буква A

'\u0410' // русская буква А

'\u0391' // греческая буква A

Символьный литерал должен содержать строго один символ, или специальную последовательность, начинающуюся с \. Для записи специальных символов (неотображаемых и служебных, таких как ", ', \ ) используются следующие обозначения:

\b \u0008 backspace BS – забой

\t \u0009 horizontal tab HT – табуляция

\n \u000a linefeed LF – конец строки

\f \u000c form feed FF – конец страницы

\r \u000d carriage return CR –

возврат каретки

\" \u0022 double quote " – двойная кавычка

\' \u0027 single quote ' – одинарная кавычка

\\ \u005c backslash \ – обратная косая черта

\шестнадцатеричный код от \u0000 до \u00ff символа

в шестнадцатеричном формате.

Первая колонка описывает стандартные обозначения специальных символов, используемые в Java-программах. Вторая колонка представляет их в стандартном виде Unicode -символов. Третья колонка содержит английские и русские описания. Использование \ в комбинации с другими символами приведет к ошибке компиляции.

Поддержка ввода символов через восьмеричный код обеспечивается для совместимости с С. Например:

'\101' // Эквивалентно '\u0041'

Однако таким образом можно задать лишь символы от \u0000 до \u00ff (т.е. с кодом от 0 до 255), поэтому Unicode -последовательности предпочтительней.

Поскольку обработка Unicode -последовательностей ( \uhhhh ) производится раньше лексического анализа, то следующий пример является ошибкой:

'\u000a' // символ конца строки

Компилятор сначала преобразует \u000a в символ конца строки и кавычки окажутся на разных строках кода, что является ошибкой. Необходимо использовать специальную последовательность:

'\n' // правильное обозначение конца строки

Аналогично и для символа \u000d (возврат каретки) необходимо использовать обозначение \r.

Специальные символы можно использовать в составе как символьных, так и строковых литералов.

##### Строковые литералы

Строковые литералы состоят из набора символов и записываются в двойных кавычках. Длина может быть нулевой или сколь угодно большой. Любой символ может быть представлен специальной последовательностью, начинающейся с \ (см. "Символьные литералы ").

"" // литерал нулевой длины

"\"" //литерал, состоящий из одного символа "

"Простой текст" //литерал длины 13

Строковый литерал нельзя разбивать на несколько строк в коде программы. Если требуется текстовое значение, состоящее из нескольких строк, то необходимо воспользоваться специальными символами \n и/или \r. Если же текст просто слишком длинный, чтобы уместиться на одной строке кода, можно использовать оператор конкатенации строк +. Примеры строковых литералов:

// выражение-константа, составленное из двух

// литералов

"Длинный текст " +

"с переносом"

/\*

\* Строковый литерал, содержащий текст

\* из двух строк:

\* Hello, world!

\* Hello!

\*/

"Hello, world!\r\nHello!"

На строковые литералы распространяются те же правила, что и на символьные в отношении использования символов новой строки \u000a и \u000d.

Каждый строковый литерал является экземпляром класса String. Это определяет некоторые необычные свойства строковых литералов, которые будут рассмотрены в следующей лекции.

##### Null-литерал

Null- литерал может принимать всего одно значение: null. Это литерал ссылочного типа, причем эта ссылка никуда не ссылается, объект отсутствует. Разумеется, его можно применять к ссылкам любого объектного типа данных. Типы данных подробно рассматриваются в следующей лекции.

##### Разделители

Разделители – это специальные символы, которые используются в служебных целях языка. Назначение каждого из них будет рассмотрено по ходу изложения курса. Вот их полный список:

( ) [ ] { } ; . ,

##### Операторы

Операторы используются в различных операциях – арифметических, логических, битовых, операциях сравнения и присваивания. Следующие 37 лексем (все состоят только из ASCII -символов) являются операторами языка Java:

= > < ! ~ ? :

== <= >= != && || ++ --

+ - \* / & | ^ % << >> >>>

+= -= \*= /= &= |= ^= %= <<= >>= >>>=

Большинство из них вполне очевидны и хорошо известны из других языков программирования, однако некоторые нюансы в работе с операторами в Java все же присутствуют, поэтому в конце лекции приводятся краткие комментарии к ним.

##### Пример программы

В заключение для примера приведем простейшую программу (традиционное Hello, world!), а затем классифицируем и подсчитаем используемые лексемы:

public class Demo {

/\*\*

\* Основной метод, с которого начинается

\* выполнение любой Java программы.

\*/

public static void main (String args[])

{

System.out.println("Hello, world!");

}

}

Итак, в приведенной программе есть один комментарий разработчика, 7 идентификаторов, 5 ключевых слов, 1 строковый литерал, 13 разделителей и ни одного оператора. Этот текст можно сохранить в файле Demo.java, скомпилировать и запустить. Результатом работы будет, как очевидно:

Hello, world!

### Дополнение. Работа с операторами

Рассмотрим некоторые детали использования операторов в Java. Здесь будут описаны подробности, относящиеся к работе самих операторов. В следующей лекции детально рассматриваются особенности, возникающие при использовании различных типов данных (например, значение операции 1/2 равно 0, а 1/2. равно 0.5 ).

#### Операторы присваивания и сравнения

Во-первых, конечно же, различаются оператор присваивания = и оператор сравнения ==.

x = 1; // присваиваем переменной x значение 1

x == 1 // сравниваем значение переменной x с

// единицей

Оператор сравнения всегда возвращает булевское значение true или false. Оператор присваивания возвращает значение правого операнда. Поэтому обычная опечатка в языке С, когда эти операторы путают:

// пример вызовет ошибку компилятора

if (x=0) { // здесь должен применяться оператор

// сравнения ==

...

}

в Java легко устраняется. Поскольку выражение x=0 имеет числовое значение 0, а не булевское (и тем более не воспринимается как всегда истинное), то компилятор сообщает об ошибке (необходимо писать x==0 ).

Условие "не равно" записывается как !=. Например:

if (x!=0) {

float f = 1./x;

}

Сочетание какого-либо оператора с оператором присваивания = (см. нижнюю строку в полном перечне в разделе "Операторы") используется при изменении значения переменной. Например, следующие две строки эквивалентны:

x = x + 1;

x += 1;

#### Арифметические операции

Наряду с четырьмя обычными арифметическими операциями +, -, \*, /, существует оператор получения остатка от деления %, который может быть применен как к целочисленным аргументам, так и к дробным.

Работа с целочисленными аргументами подчиняется простым правилам. Если делится значение a на значение b, то выражение (a/b)\*b+(a%b) должно в точности равняться a. Здесь, конечно, оператор деления целых чисел / всегда возвращает целое число. Например:

9/5 возвращает 1

9/(-5) возвращает -1

(-9)/5 возвращает -1

(-9)/(-5) возвращает 1

Остаток может быть положительным, только если делимое было положительным. Соответственно, остаток может быть отрицательным только в случае отрицательного делимого.

9%5 возвращает 4

9%(-5) возвращает 4

(-9)%5 возвращает -4

(-9)%(-5) возвращает -4

Попытка получить остаток от деления на 0 приводит к ошибке.

Деление с остатком для дробных чисел может быть произведено по двум различным алгоритмам. Один из них повторяет правила для целых чисел, и именно он представлен оператором %. Если в рассмотренном примере деления 9 на 5 перейти к дробным числам, значение остатка во всех вариантах не изменится (оно будет также дробным, конечно).

9.0%5.0 возвращает 4.0

9.0%(-5.0) возвращает 4.0

(-9.0)%5.0 возвращает -4.0

(-9.0)%(-5.0) возвращает -4.0

Однако стандарт IEEE 754 определяет другие правила. Такой способ представлен методом стандартного класса Math.IEEEremainder(double f1, double f2). Результат этого метода – значение, которое равно f1-f2\*n, где n – целое число, ближайшее к значению f1/f2, а если два целых числа одинаково близки к этому отношению, то выбирается четное. По этому правилу значение остатка будет другим:

Math.IEEEremainder(9.0, 5.0) возвращает -1.0

Math.IEEEremainder(9.0, -5.0) возвращает -1.0

Math.IEEEremainder(-9.0, 5.0) возвращает 1.0

Math.IEEEremainder(-9.0, -5.0) возвращает 1.0

Унарные операторы инкрементации ++ и декрементации --, как обычно, можно использовать как справа, так и слева.

int x=1;

int y=++x;

В этом примере оператор ++ стоит перед переменной x, это означает, что сначала произойдет инкрементация, а затем значение x будет использовано для инициализации y. В результате после выполнения этих строк значения x и y будут равны 2.

int x=1;

int y=x++;

А в этом примере сначала значение x будет использовано для инициализации y, и лишь затем произойдет инкрементация. В результате значение x будет равно 2, а y будет равно 1.

#### Логические операторы

Логические операторы "и" и "или" ( & и | ) можно использовать в двух вариантах. Это связано с тем, что, как легко убедиться, для каждого оператора возможны случаи, когда значение первого операнда сразу определяет значение всего логического выражения. Если вторым операндом является значение некоторой функции, то появляется выбор – вызывать ее или нет, причем это решение может сказаться как на скорости, так и на функциональности программы.

Первый вариант операторов ( &, | ) всегда вычисляет оба операнда, второй же – ( &&, || ) не будет продолжать вычисления, если значение выражения уже очевидно. Например:

int x=1;

(x>0) | calculate(x) // в таком выражении

// произойдет вызов

// calculate

(x>0) || calculate(x) // а в этом - нет

Логический оператор отрицания "не" записывается как ! и, конечно, имеет только один вариант использования. Этот оператор меняет булевское значение на противоположное.

int x=1;

x>0 // выражение истинно

!(x>0) // выражение ложно

Оператор с условием ?: состоит из трех частей – условия и двух выражений. Сначала вычисляется условие (булевское выражение), а на основании результата значение всего оператора определяется первым выражением в случае получения истины и вторым – если условие ложно. Например, так можно вычислить модуль числа x:

x>0 ? x : -x

#### Битовые операции

Прежде чем переходить к битовым операциям, необходимо уточнить, каким именно образом целые числа представляются в двоичном виде. Конечно, для неотрицательных величин это практически очевидно:

0 0

1 1

2 10

3 11

4 100

5 101

и так далее. Однако как представляются отрицательные числа? Во-первых, вводят понятие знакового бита. Первый бит начинает отвечать за знак, а именно 0 означает положительное число, 1 – отрицательное. Но не следует думать, что остальные биты остаются неизменными. Например, если рассмотреть 8-битовое представление:

-1 10000001 // это НЕВЕРНО!

-2 10000010 // это НЕВЕРНО!

-3 10000011 // это НЕВЕРНО!

Такой подход неверен! В частности, мы получаем сразу два представления нуля – 00000000 и 100000000, что нерационально. Правильный алгоритм можно представить себе так. Чтобы получить значение -1, надо из 0 вычесть 1:

00000000

- 00000001

------------

- 11111111

Итак, -1 в двоичном виде представляется как 11111111. Продолжаем применять тот же алгоритм (вычитаем 1):

0 00000000

-1 11111111

-2 11111110

-3 11111101

и так далее до значения 10000000, которое представляет собой наибольшее по модулю отрицательное число. Для 8-битового представления наибольшее положительное число 01111111 (=127), а наименьшее отрицательное 10000000 (=-128). Поскольку всего 8 бит определяет 28=256 значений, причем одно из них отводится для нуля, то становится ясно, почему наибольшие по модулю положительные и отрицательные значения различаются на единицу, а не совпадают.

Как известно, битовые операции "и", "или", "исключающее или" принимают два аргумента и выполняют логическое действие попарно над соответствующими битами аргументов. При этом используются те же обозначения, что и для логических операторов, но, конечно, только в первом (одиночном) варианте. Например, вычислим выражение 5&6:

00000101

& 00000110

-------------

00000100

// число 5 в двоичном виде

// число 6 в двоичном виде

//проделали операцию "и" попарно над битами

// в каждой позиции

То есть выражение 5&6 равно 4.

Исключение составляет лишь оператор "не" или "NOT", который для побитовых операций записывается как ~ (для логических было !). Этот оператор меняет каждый бит в числе на противоположный. Например, ~(-1)=0. Можно легко установить общее правило для получения битового представления отрицательных чисел:

Если n – целое положительное число, то -n в битовом представлении равняется ~(n-1).

Наконец, осталось рассмотреть лишь операторы побитового сдвига. В Java есть один оператор сдвига влево и два варианта сдвига вправо. Такое различие связано с наличием знакового бита.

При сдвиге влево оператором << все биты числа смещаются на указанное количество позиций влево, причем освободившиеся справа позиции заполняются нулями. Эта операция аналогична умножению на 2n и действует вполне предсказуемо, как при положительных, так и при отрицательных аргументах.

Рассмотрим примеры применения операторов сдвига для значений типа int, т.е. 32-битных чисел. Пусть положительным аргументом будет число 20, а отрицательным -21.

// Сдвиг влево для положительного числа 20

20 << 00 = 00000000000000000000000000010100 = 20

20 << 01 = 00000000000000000000000000101000 = 40

20 << 02 = 00000000000000000000000001010000 = 80

20 << 03 = 00000000000000000000000010100000 = 160

20 << 04 = 00000000000000000000000101000000 = 320

...

20 << 25 = 00101000000000000000000000000000 = 671088640

20 << 26 = 01010000000000000000000000000000 = 1342177280

20 << 27 = 10100000000000000000000000000000 = -1610612736

20 << 28 = 01000000000000000000000000000000 = 1073741824

20 << 29 = 10000000000000000000000000000000 = -2147483648

20 << 30 = 00000000000000000000000000000000 = 0

20 << 31 = 00000000000000000000000000000000 = 0

// Сдвиг влево для отрицательного числа -21

-21 << 00 = 11111111111111111111111111101011 = -21

-21 << 01 = 11111111111111111111111111010110 = -42

-21 << 02 = 11111111111111111111111110101100 = -84

-21 << 03 = 11111111111111111111111101011000 = -168

-21 << 04 = 11111111111111111111111010110000 = -336

-21 << 05 = 11111111111111111111110101100000 = -672

...

-21 << 25 = 11010110000000000000000000000000 = -704643072

-21 << 26 = 10101100000000000000000000000000 = -1409286144

-21 << 27 = 01011000000000000000000000000000 = 1476395008

-21 << 28 = 10110000000000000000000000000000 = -1342177280

-21 << 29 = 01100000000000000000000000000000 = 1610612736

-21 << 30 = 11000000000000000000000000000000 = -1073741824

-21 << 31 = 10000000000000000000000000000000 = -2147483648

Как видно из примера, неожиданности возникают тогда, когда значащие биты начинают занимать первую позицию и влиять на знак результата.

При сдвиге вправо все биты аргумента смещаются на указанное количество позиций, соответственно, вправо. Однако встает вопрос – каким значением заполнять освобождающиеся позиции слева, в том числе и отвечающую за знак. Есть два варианта. Оператор >> использует для заполнения этих позиций значение знакового бита, то есть результат всегда имеет тот же знак, что и начальное значение. Второй оператор >>> заполняет их нулями, то есть результат всегда положительный.

// Сдвиг вправо для положительного числа 20

// Оператор >>

20 >> 00 = 00000000000000000000000000010100 = 20

20 >> 01 = 00000000000000000000000000001010 = 10

20 >> 02 = 00000000000000000000000000000101 = 5

20 >> 03 = 00000000000000000000000000000010 = 2

20 >> 04 = 00000000000000000000000000000001 = 1

20 >> 05 = 00000000000000000000000000000000 = 0

// Оператор >>>

20 >>> 00 = 00000000000000000000000000010100 = 20

20 >>> 01 = 00000000000000000000000000001010 = 10

20 >>> 02 = 00000000000000000000000000000101 = 5

20 >>> 03 = 00000000000000000000000000000010 = 2

20 >>> 04 = 00000000000000000000000000000001 = 1

20 >>> 05 = 00000000000000000000000000000000 = 0

Очевидно, что для положительного аргумента операторы >> и >>> работают совершенно одинаково. Дальнейший сдвиг на большее количество позиций будет также давать нулевой результат.

// Сдвиг вправо для отрицательного числа -21

// Оператор >>

-21 >> 00 = 11111111111111111111111111101011 = -21

-21 >> 01 = 11111111111111111111111111110101 = -11

-21 >> 02 = 11111111111111111111111111111010 = -6

-21 >> 03 = 11111111111111111111111111111101 = -3

-21 >> 04 = 11111111111111111111111111111110 = -2

-21 >> 05 = 11111111111111111111111111111111 = -1

// Оператор >>>

-21 >>> 00 = 11111111111111111111111111101011 = -21

-21 >>> 01 = 01111111111111111111111111110101 = 2147483637

-21 >>> 02 = 00111111111111111111111111111010 = 1073741818

-21 >>> 03 = 00011111111111111111111111111101 = 536870909

-21 >>> 04 = 00001111111111111111111111111110 = 268435454

-21 >>> 05 = 00000111111111111111111111111111 = 134217727

...

-21 >>> 24 = 00000000000000000000000011111111 = 255

-21 >>> 25 = 00000000000000000000000001111111 = 127

-21 >>> 26 = 00000000000000000000000000111111 = 63

-21 >>> 27 = 00000000000000000000000000011111 = 31

-21 >>> 28 = 00000000000000000000000000001111 = 15

-21 >>> 29 = 00000000000000000000000000000111 = 7

-21 >>> 30 = 00000000000000000000000000000011 = 3

-21 >>> 31 = 00000000000000000000000000000001 = 1

Как видно из примеров, эти операции аналогичны делению на 2n. Причем, если для положительных аргументов с ростом n результат закономерно стремится к 0, то для отрицательных предельным значением является -1.

### Заключение

В этой лекции были рассмотрены основы лексического анализа программ Java. Для их записи применяется универсальная кодировка Unicode, позволяющая использовать любой язык помимо традиционного английского. Еще раз напомним, что использование Unicode возможно и необходимо в следующих конструкциях:

* комментарии;
* идентификаторы ;
* символьные и строковые литералы.

Остальные же ( пробелы, ключевые слова, числовые, булевские и null- литералы, разделители и операторы) легко записываются с применением лишь ASCII -символов. В то же время любой Unicode -символ также можно задать в виде специальной последовательности ASCII -символов.

Во время анализа компилятор выделяет из текста программы < пробелы > (были рассмотрены все символы, которые рассматриваются как пробелы ) и комментарии, которые полностью удаляются из кода (были рассмотрены все виды комментариев, в частности комментарий разработчика). Пробелы и все виды комментариев служат для разбиения текста программы на лексемы. Были рассмотрены все виды лексем, в том числе все виды литералов.

В дополнении были рассмотрены особенности применения различных операторов.