**ПРОГРАММА**

Вся информация хранится в файлах. По сути **файл** — **это некоторая область на жестком диске, которая является логическим целым.** Именно логическим — ваш файл может быть разбросан по нескольким секциям жесткого диска (или иного носителя, например флешки или компакт-диска). Специальная программа — операционная система — управляет файлами на диске. А человек (и другие программы) видят файл как одно целое. И могут его копировать, переносить и даже удалять.

Так вот программа — **это просто файл с набором команд**, которые могут быть выполнены процессором. Запускает программу обычно операционная система — она считывает программы из файла, загружает в память и запускает.

Каждая программа при своем создании проходит простой цикл — **написание текста, компиляция, запуск и проверка работоспособности**. Большинство программ — это обычный текст, который можно набирать в том же приложении «Блокнот» в Windows (или еще каком-либо простом текстовом редакторе). В большие программы конечно же подключаются картинки, видео, звук и прочие ресурсы, но тем не менее основным элементом программ является обычный текст, который описывает порядок выполнения вашей программы. Шаг за шагом.

Компиляция — это перевод текста программы в набор кодов, которые понятны процессору. Вы наверно не будете удивлены тем, что программы, которые исполняются на CPU (central processing unit, CPU, дословно — центральное обрабатывающее устройство, часто просто процессор) явно не текстовые. Они представляю из себя набор байтов (думаю, что о байтах вы что-то слышали) состоящие из нулей и единиц (и о двоичном коде возможно тоже что-то слышали). Каждая комбинация является командой или данными, которые обрабатывает CPU. Так вот много лет назад программы для компьютеров писались именно так — в память забивали набор байтов, состоящих из нулей и единиц и стартовали программу. Внешний вид такого чуда мысли программиста был мягко говоря слабо читаем. Хотя люди с опытом могли творить просто чудеса. Тем не менее такое кодирование имело еще одну проблему — надо было все делать самому. Где-то в памяти хранить данные и где-то хранить команды. Их надо было разделять, и ими надо было управлять И все это делалось вручную — т.е. программист был с одной стороны почти богом (мог делать все, что хочет), с другой стороны он был вынужден делать титаническую работу по разбиению программы на мелкие-мелкие детали. Как частенько говорят — «из-за деревьев леса не видно». Написать программу обработки текста — это была проблема. В итоге инженеры пришли к нормальной идее, что можно создать так называемые алгоритмические языки программирования. Идея достаточно простая — некоторые конструкции при написании алгоритмов являются стандартными и их на самом деле не так уж и много — мы это еще увидим. Конструкции стандартные — значит и перевод их с понятного представления в машинное тоже будет стандартным. Т.е. можно написать программу, которая будет «переводить» текст на алгоритмическом языке в машинный. И уже полученный результат можно исполнять на CPU. Правда тут возникла еще одна особенность — процесс компиляции может выдавать ошибки, если программа содержит некорректные с точки зрения компилятора конструкции. Но алгоритмические языки резко увеличили скорость программирования. Компилятор сам распределяет память для данных, команд — это огромное подспорье.

С этого и началась эра алгоритмических языков — люди писали программы в более-менее понятной форме, в виде текста. Компилятор их преобразовывал в машинный код, который можно было исполнять.

Стоит отметить. что также существуют программы-интерпретаторы. В отличии от компилируемых языков, которые на выходе получают сразу готовый машинный код, интерпретаторы получают на вход текст программы и пытаются ее выполнить «на лету». Т.е. каждая строка или команда сразу преобразуется в машинный код и выполняется. Это удобно — не надо компилировать — но с другой стороны это замедляет работу программы. Для каждой команды надо выполнить преобразование в машинный код, а потом выполнить этот машинный код. Долго, но зато удобно.

**Java Virtual Machine**

Java — это тоже компилируемый язык программирования. И как у других подобных языков, у него тоже есть стадии написания текста, компиляции и запуска. Но в отличии от языков типа C, Pascal и прочих, программа на Java не сразу компилируется в машинный код. После компиляции создается так называемый байт-код. И этот байт-код выполняется специальной программой — Java Virtual Machine, JVM (виртуальная машина Java). Зачем это надо ? Идея очень проста и логична. Каждый тип процессора имеет свой набор команд, что означает, что один и тот же текст на алгоритмическом языке будет преобразовываться в разные машинные коды в зависимости от того, под какой процессор вы компилируете вашу программу. Да и сам компилятор тоже должен существовать под конкретный процессор. И даже если вы сами написали нужный компилятор — надо иметь под рукой исходный текст. А его вам могут и не дать. Он денег стоит.

Из всего этого следует, что компилировать программу под каждый процессор, и даже под каждую операционную систему отдельно требует больших усилий. В компании Sun Microsystems пошли по следующему пути: решили, что программы на Java будут выполняться под управлением специальной программы, **по сути на виртуальном процессоре**, который имеет совершенно определенный набор команд. И каждая программа на Java может быть выполнена по сути на любом процессоре, на любой операционной системе под которую написана JVM. Таким образом один раз скомпилированная программа на Java будет по идее выполняться где угодно. Девиз Sun был именно таким: Write once run anywhere — написано однажды, запускается везде.

В итоге программа на Java тоже проходит стадию написания текста, компиляции и запуска. Но запускается она под управлением JVM, которая интерпретирует байт-код. вы вполне законно можете подумать, что раз «интерпретирует» — значит работает медленнее. Да, в некоторых случаях это так. Но и тут разработчики Sun смогли найти решение. Был изобретен JIT-компилятор (JIT — Just In Time). Этот механизм позволяет преобразовать байт-код в машинный код при первом проходе программы и в дальнейшем выполнять просто готовый машинный код. Т.е. появилась еще одна стадия — компиляция байт-кода в машинный код. Это позволяет существенно ускорить работу программы. Скорость работы программы на Java вполне сопоставима со скоростью работы программы на том же C или Pascal. Так что страшные истории про жуткие тормоза программ на Java — это только истории и больше ничего. Хотя относительно графических программ можно сказать, что Java уступает в быстродействии.

**Программа на Java — шаги по запуску программы**

Кроме того, что программу надо написать, программу надо еще суметь запустить. Так что программисты постоянно делают один и тот же цикл из четырех шагов:

1)-Редактировать код программы

2)-Собрать/скомпилировать программу из текста в код, понятный JVM — байт-код. Опять же — надо исправлять ошибки компилятора. Тогда возвращаемся к первому шагу.

3)-Если все скомпилировалось — запустить программу под управлением JVM

Посмотреть корректность работы программы. Если даже программа запустилась — не факт, что вы совершенно верно реализовали алгоритм. Нередко надо опять идти к первому шагу.

Если на каком-то шаге произошла ошибка, то повторяем все с самого начала. Если все хорошо, то цикл можно завершать.

**Установка Java Development Kit (JDK)**

JDK — это набор программ для разработки и запуска программ на Java. Для нас самыми главными будут две программы:

**javac** — компилятор файлов с программами, написанными на java. У таких файлов обычно расширение .java. Например First.java. После успешной компиляции появляется файл с расширением .class. Т.е. если мы скомпилируем First.java, то должен появиться файл First.class. По большому секрету файлов .class может быть несколько, но мы этот вопрос разберем, .когда придет время.

**java** — по сути это программа, которая и является JVM. Именно она исполняет **байт-код. который содержится в файле .class**

В нашем курсе мы будем рассматривать работу под операционной системой Windows. Если вы предпочтете использовать иную систему, то Вам придется потрудиться — мы не рассматриваем как работать на Java под другими ОС. Установка JDK не является сложной задачей — если у вас будет несколько иная версия, чем мы рассматриваем, то вряд ли вы увидите очень большую разницу.