Пакеты

Программа на Java представляет собой набор пакетов – аналогов директорий в файловых системах. Каждый пакет может включать вложенные пакеты, и в совокупности они образуют иерархическую систему. Кроме этого, пакеты могут содержать классы и интерфейсы, и таким образом группируют типы, что необходимо сразу для нескольких целей. Во-первых, существует специальный уровень доступа, позволяющий типам из одного пакета более тесно взаимодействовать друг с другом, чем с классами из других пакетов. Во-вторых, каждый пакет имеет свое пространство имен, что позволяет создавать одноименные классы в различных пакетах, и разработчикам не приходится тратить время на разрешение конфликта имен. Наконец, с применением пакетов гораздо проще эффективно организовать взаимодействие подсистем друг с другом.

Объявление пакета – первое выражение в модуле компиляции. Оно записывается с помощью ключевого слова package, после которого указывается полное имя пакета. Например, первой строкой (после комментариев) в файле java/lang/Object.java является строка package java.lang; что служит одновременно объявлением пакета lang, вложенного в пакет java, и указанием, что объявляемый ниже класс Object, находится в этом пакете. Так складывается полное имя класса: java.lang.Object.

Если это выражение отсутствует, то такой модуль компиляции принадлежит безымянному пакету. Этот пакет по умолчанию обязательно должен поддерживаться реализацией Java-платформы. Он не может иметь вложенных пакетов, так как составное имя пакета должно обязательно начинаться с имени пакета верхнего уровня. Пакет по умолчанию был введен в Java для облегчения написания небольших или временных приложений.

Область видимости объявления типа – пакет, в котором он располагается. Это означает, что внутри этого пакета допускается обращение к типу по его простому имени. Из всех других пакетов необходимо обращаться по составному имени – это полное имя пакета плюс простое имя типа, разделенные точкой. Поскольку пакеты могут иметь довольно длинные имена, а тип может многократно использоваться в модуле компиляции, такое ограничение может привести к усложнению исходного кода и сложностям в разработке. Для решения этой проблемы вводятся import-выражения, позволяющие импортировать типы в модуль компиляции и далее обращаться к ним по простым именам.

Существует два вида import-выражений:

- импорт одного типа - записываются с помощью ключевого слова import и полного имени типа, например, import java.net.URL; Такое выражение означает, что в дальнейшем в этом модуле компиляции простое имя URL будет обозначать одноименный класс из пакета java.net;

- импорт пакета - включает в себя полное имя пакета, например, import java.awt.*; Это выражение делает доступными все типы, находящиеся в пакете java.awt, по их простому имени.

Модификаторы доступа

Для избавления разработчика от излишней зависимости от деталей внутренней реализации класса используются модификаторы доступа. Уровень доступа элемента языка является статическим свойством, задается на уровне кода и всегда проверяется во время компиляции. В Java модификаторы доступа указываются для: типов (классов и интерфейсов), элементов ссылочных типов (полей, методов, внутренних типов) и конструкторов классов.

В языке Java определены четыре модификатора доступа (от менее открытых - к более открытым):

- private – разрешает доступ только внутри класса;
- default – разрешает доступ из пакета, в котором объявлен класс;
- protected – разрешает доступ из пакета, в котором объявлен класс, а также наследникам классов;
- public – разрешает доступ из любой точки программы.

Уровень доступа по умолчанию (default) определяется автоматически, если не указан ни один из модификаторов доступа.

Классы

Объявление класса состоит из заголовка и тела класса. В заголовке сначала указываются модификаторы класса, затем – ключевое слово class, затем – имя класса. Также класс может быть объявлен как final – в этом случае не допускается создание наследников такого класса, и на своей ветке наследования он является последним. Попытка расширить final-класс приведёт к ошибке компиляции.

Далее заголовок может содержать ключевое слово extends, после которого должно быть указано имя доступного не-final класса. В этом случае объявляемый класс наследуется от указанного класса. Если выражение extends не применяется, то класс наследуется напрямую от базового класса Object, при этом выражение extends Object допускается, но игнорируется. Сам класс Object находится на вершине иерархии наследования и не является чьим-либо наследником.

Если в объявлении класса A указано выражение extends B, то класс A называют прямым наследником класса B. Класс A считается наследником класса B если либо A является прямым наследником B, либо существует класс C, который является наследником B, а A является наследником C (это правило применяется рекурсивно). Если компилятор обнаруживает, что класс является своим наследником, то возникает ошибка компиляции. Наследование более чем от одного класса (множественное наследование) в языке Java запрещено.

Далее в заголовке может быть указано ключевое слово implements, за которым должно следовать перечисление через запятую имен доступных интерфейсов. В этом случае говорят, что класс реализует перечисленные интерфейсы. Класс может реализовывать любое количество интерфейсов. Если выражение implements отсутствует, то класс действительно не реализует никаких интерфейсов.

За заголовком следует пара фигурных скобок, которые могут быть пустыми или содержать описание тела класса. Тело класса может содержать объявление элементов класса (полей, методов и внутренних типов) и остальных допустимых конструкций (конструкторов, инициализаторов, статических инициализаторов).

Элементами класса являются элементы, описанные в объявлении тела класса и переданные по наследству от класса-родителя и всех реализуемых интерфейсов при условии достаточного уровня доступа. Элементы класса имеют имена и передаются по наследству, областью видимости элементов является все объявление тела класса. Для элементов простые имена указываются при объявлении, составные формируются из имени класса или имени переменной объектного типа и простого имени элемента. Допускается применение к ним любого из модификаторов доступа.

Поля и методы могут иметь одинаковые имена, поскольку обращение к полям всегда записывается без скобок, а к методам - всегда со скобками.

Конструкторы, инициализаторы и статические инициализаторы не обладают именами, а потому не могут быть вызваны явно – их вызывает сама виртуальная машина. По той же причине они не обладают и модификаторами доступа.

Поля

Объявление полей начинается с перечисления модификаторов. Возможно применение любого из модификаторов доступа, либо ни одного из них, что означает уровень доступа по умолчанию. Поле также может быть объявлено final, что означает, что оно инициализируется ровно один раз, и больше не будет менять своего значения.

После списка модификаторов указывается тип поля. Затем идет перечисление одного или нескольких имен полей с возможными инициализаторами. Запрещается использовать поле в инициализации других полей до его объявления, но в остальных случаях поля можно объявлять и ниже их использования.

Методы

Объявление метода состоит из заголовка и тела метода. Заголовок состоит из следующих элементов:

- модификаторов, в том числе и модификаторов доступа;
- типа возвращаемого значения или ключевого слова void;
- имени метода;
- списка аргументов (параметров) в круглых скобках;
- специального throws-выражения.

Заголовок начинается с перечисления модификаторов. Для методов доступен любой возможных модификаторов доступа или использование доступа по умолчанию. Кроме этого, существует модификатор final, который говорит о том, что такой метод нельзя переопределять в наследниках. Можно считать, что все методы final-класса, а также все private-методы любого класса являются final. Также поддерживается модификатор native. Объявленный с таким модификатором метод не имеет реализации на Java – он должен быть написан на другом языке программирования и добавлен в систему в виде загружаемой динамической библиотеки. Существует специальная спецификация JNI (Java Native Interface), описывающая правила создания и использования native-методов.

После перечисления модификаторов указывается тип возвращаемого значения. Это может быть как примитивный, так и объектный тип. Если метод не возвращает никакого значения, указывается ключевое слово void.

Затем определяется имя метода. Указанный идентификатор при объявлении становится простым именем метода. Составное имя формируется из имени класса или имени переменной объектного типа и простого имени метода. Областью видимости метода является все объявление тела класса.

Аргументы метода перечисляются через запятую. Для каждого указывается сначала тип, затем имя параметра. Если аргументы отсутствуют, указываются пустые круглые скобки. Использование одноименных аргументов, а также создание в методе локальных переменных с именами, совпадающими с именами параметров, запрещено. Для каждого аргумента можно указать ключевое слово final перед указанием его типа. В этом случае такой параметр не может менять своего значения в теле метода.

Важным понятием является сигнатура метода. Сигнатура определяется именем метода и его аргументами (количеством, типом, порядком следования). Если для полей запрещается совпадение имен, то для методов в классе запрещено создание двух методов с одинаковыми сигнатурами.

Завершает заголовок метода throws-выражение, которое применяется в Java для корректной обработки ошибок.

Конструкторы

Формат объявления конструкторов похож на упрощенное объявление методов. Также выделяют заголовок и тело конструктора. Заголовок состоит, во-первых, из модификаторов доступа (никакие другие модификаторы не допустимы). Затем указывается имя класса, которое можно расценивать двояко. Можно считать, что имя конструктора совпадает с именем класса. А можно рассматривать конструктор как безымянный, а имя класса – как тип возвращаемого значения, так как конструктор может породить только объект класса, в котором он объявлен.

Затем следует перечисление входных аргументов по тем же правилам, что и для обычных методов. В связи с отсутствием имени (или из-за того, что у всех

конструкторов одинаковое имя, совпадающее с именем класса) сигнатура конструктора определяется только набором входных параметров по тем же правилам, что и для методов. Аналогично, в одном классе допускается любое количество конструкторов, если у них различные сигнатуры.

Также завершает заголовок конструктора throws-выражение. Оно имеет особую важность для конструкторов, поскольку создать ошибку – это единственный способ для конструктора не создавать объект. Если конструктор выполнился без ошибок, то объект гарантированно создается.

Тело конструктора пустым быть не может и поэтому всегда описывается в фигурных скобках (скобки при необходимости могут быть пустыми). Также оно может содержать любое количество return-выражений без аргументов. Если процесс исполнения дойдет до такого выражения, то на этом месте выполнение конструктора будет завершено.

Однако логика работы конструкторов имеет и некоторые важные особенности. Поскольку при их вызове осуществляется создание и инициализация объекта, то становится понятно, что такой процесс не может происходить без обращения к конструкторам всех родительских классов. Поэтому вводится обязательное правило - первой строкой в конструкторе должно быть обращение к родительскому классу, которое записывается с помощью ключевого слова super, за которым идет перечисление аргументов.

Создание объекта начинается при исполнении выражения с ключевым словом new, за которым следует имя класса, от которого будет порождаться объект, и набор аргументов для его конструктора. По этому набору определяется, какой именно конструктор будет использован, и происходит его вызов. Первая строка его тела содержит вызов родительского конструктора. В свою очередь, первая строка тела конструктора родителя будет содержать вызов далее к его родителю, и так далее. Восхождение по дереву наследования заканчивается, очевидно, на классе Object, у которого есть единственный конструктор без параметров. Именно в этот момент JVM порождает объект, и далее начинается процесс его инициализации. Выполнение начинает обратный путь вниз по дереву наследования. У самого верхнего родителя, прямого наследника от Object, происходит продолжение исполнения конструктора со второй строки. Когда он будет полностью выполнен, необходимо перейти к следующему родителю на один уровень наследования вниз и завершить выполнение его конструктора, и так далее. Наконец, можно будет вернуться к конструктору исходного класса, который был вызван с помощью new, и также продолжить его выполнение со второй строки. По его завершению объект считается полностью созданным, исполнение выражения new будет закончено, а в качестве результата будет возвращена ссылка на порожденный объект.

Инициализаторы

Наконец, последней допустимой конструкцией в теле класса является объявление инициализаторов. Записываются объектные инициализаторы очень просто – внутри фигурных скобок. Инициализаторы не имеют имен, исполняются при создании объектов, не могут быть вызваны явно и не передаются по наследству (хотя инициализаторы в родительском классе продолжают исполняться при создании объекта класса-наследника).

Статические элементы

В ряде случаев необходимо описать поле, значение которого не является характеристикой экземпляра данного класса, а относится ко всему классу в целом. В отличие от полей объекта, такие поля называются полями класса и объявляются такие поля с помощью модификатора static. Чтобы обратиться к такому полю, ссылка на объект не требуется – вполне достаточно имени класса. Для удобства позволяется обращаться к статическим полям и через ссылки на экземпляры класса, однако такое обращение неявно конвертируется компилятором в обращение через имя класса. Обращение к статическому полю является корректным независимо от того, были ли порождены объекты от этого класса и в каком количестве.

Если объявление статического поля совмещается с его инициализацией, то поле инициализируется также однократно при загрузке класса. На объявление и применение статических полей накладываются аналогичные ограничения, что и для динамических - нельзя использовать поле в инициализаторах других полей или в инициализаторах класса до того, как это поле объявлено.

По аналогии со статическими полями существуют и статические методы. Например, метод main() запускается до того, как программа создаст хотя бы один объект. Статическими также могут быть и инициализаторы. В отличие от инициализаторов объекта, рассматриваемых ранее, они называются инициализаторами класса, и их код выполняется один раз во время загрузки класса в память виртуальной машины.

Статические поля также могут быть объявлены как final, что означает, что они должны быть проинициализированы строго один раз, и затем уже больше не менять своего значения. Аналогично, статические методы могут быть объявлены как final, что означает, что их нельзя переопределять в классах-наследниках.

Для инициализации статических полей можно пользоваться статическими методами и нельзя обращаться к динамическим методам. Вводят специальные понятия - статический и динамический контексты. К статическому контексту относят статические методы, статические инициализаторы, инициализаторы статических полей. Все остальные части кода имеют динамический контекст.

Абстрактные классы и методы

Иногда бывает удобным описать только заголовок метода, без его тела, и таким образом объявить, что такой метод будет существовать в этом классе и его потомках. Реализацию этого метода, то есть его тело, можно описать позже в наследниках данного класса. В таком случае при описании метода используется модификатор abstract.

Поскольку абстрактный метод не имеет тела, после описания его заголовка ставится точка с запятой. А раз у него нет тела, то к нему нельзя обращаться, пока его наследники не опишут реализацию. Это означает, что нельзя создавать экземпляры класса, у которого есть абстрактные методы. Такой класс сам объявляется абстрактным.

Класс может быть абстрактным и в случае, если у него нет абстрактных методов, но должен быть абстрактным, если такие методы есть. Разработчик может указать ключевое слово abstract в списке модификаторов класса, если хочет запретить создание экземпляров этого класса. Классы-наследники должны реализовать все абстрактные методы своего абстрактного родителя (если они есть), чтобы их можно было объявлять не абстрактными и порождать от них экземпляры. Конечно, класс не может быть одновременно abstract и final. Это утверждение верно и для методов. Кроме того, абстрактный метод не может быть private, native, static.

Сам класс может без ограничений пользоваться своими абстрактными методами. Это корректно, поскольку метод класса может быть вызван только у объекта, а объект при этом может быть порожден только от неабстрактного класса, который является его наследником и должен реализовать все абстрактные методы. По этой же причине можно объявлять переменные типа абстрактный класс. Они могут иметь значение null или ссылаться на объект, порожденный от неабстрактного наследника этого класса.

Вложенные классы

Синтаксис языка Java позволяет нам объявлять классы внутри другого класса, такие классы называются внутренними или вложенными. Причины использования вложенных классов:

- это хороший способ группировки классов, которые используются только в одном месте: если класс полезен только для одного другого класса, то логично будет держать их вместе. Вложение таких вспомогательных классов делает код более удобным;
- инкапсуляция: допустим, есть два класса – A и B, и классу B требуется доступ к свойству класса A, которое может быть закрытым. Вложение класса B в класс A решит эту проблему, и, более того, сам класс B можно скрыть от внешнего использования;
- улучшение читаемости и обслуживаемости кода: вложение классов позволяет хранить код там, где он используется.

Вложенные классы делятся на два вида: статические и не статические. Вложенные классы, объявленные с модификатором static, называются вложенными статическими классами, объявленные без модификатора static – внутренними классами. Внутренние классы имеют доступ к полям содержащего их внешнего класса, даже если они объявлены как private, статические – не имеют. Как и другие поля класса, вложенные классы могут быть объявлены с любым модификатором доступа.

Локальные классы – это классы, объявленные в блоке операторов между фигурными скобками, например, в теле метода, цикла или оператора ветвления. Локальный класс имеет доступ к членам класса, в котором он объявлен, а также к локальным переменным, объявленным как final. Локальный класс, объявленный в теле метода, имеет доступ к параметрам этого метода. Объявление переменных в локальном классе затеняет переменные, ранее объявленные с тем же именем в содержащей этот класс области. Локальные классы, как и внутренние, не могут содержать статические методы и статические поля, не объявленные как final, и могут обращаться только к статическим полям внешнего класса.

Анонимные классы позволяют сделать код более кратким, одновременно объявляя класс и создавая его экземпляр. В отличие от локальных классов, анонимные классы не имеют имени и не могут содержать конструкторы, а их объявление происходит внутри выражения и напоминает вызов конструктора класса. Оно состоит из оператора new, имени наследуемого класса или реализуемого интерфейса, списка аргументов в круглых скобках и тела класса в фигурных скобках. Поскольку анонимный класс является частью выражения, после закрывающейся фигурной скобки ставится точка с запятой.

Ограничения и правила доступа к локальным переменным для анонимных классов те же, что и для локальных классов. Анонимные классы также могут содержать в себе локальные классы, и идеальны для реализации интерфейсов, содержащих два и более метода.

Интерфейсы

Концепция абстрактных методов позволяет предложить альтернативу множественному наследованию. В Java класс может иметь только одного родителя, поскольку при множественном наследовании могут возникать конфликты, которые серьезно запутывают объектную модель. Например, если у класса есть два родителя, которые имеют одинаковый метод с различной реализацией, то какой из них унаследует новый класс? И как будет работать функциональность родительского класса, который лишился своего метода?

Все эти проблемы не возникают в случае, если наследуются только абстрактные методы от нескольких родителей. Даже если будет унаследовано несколько одинаковых методов, все равно у них нет реализации, и можно один раз описать тело

метода, которое будет использовано при вызове любого из этих методов. Именно так устроены интерфейсы в Java. Они во многом напоминают классы, но могут содержать только константы, вложенные типы и сигнатуры методов без их описания. Нельзя создать объект типа интерфейса – интерфейсы могут только реализовываться некоторым классом или наследоваться другим интерфейсом. Объявление интерфейсов очень похоже на упрощенное объявление классов и начинается с заголовка.

Сначала указываются модификаторы доступа. Интерфейс может быть объявлен как public, и тогда он будет доступен для всеобщего использования, либо модификатор доступа может не указываться, в этом случае интерфейс доступен только для типов своего пакета. Модификатор abstract для интерфейса не требуется, поскольку все интерфейсы являются абстрактными.

Далее записывается ключевое слово interface и имя интерфейса, после этого может следовать ключевое слово extends и список интерфейсов, от которых будет наследоваться объявляемый интерфейс. Родительских типов может быть много - главное, чтобы не было повторений, и чтобы отношение наследования не образовывало циклической зависимости.

Затем в фигурных скобках записывается тело интерфейса. Оно состоит из объявления элементов, то есть полей-констант и абстрактных методов. Все поля интерфейса должны быть public static final, поэтому эти модификаторы указывать необязательно и даже нежелательно, чтобы не загромождать код. Поскольку поля объявляются финальными, необходимо их сразу инициализировать. Все методы интерфейса являются public abstract, и эти модификаторы также являются необязательными и нежелательными.

При объявлении интерфейса объявляется новый ссылочный тип данных. Следовательно, название интерфейса можно использовать в качестве типа данных, как и любые другие типы. Если объявлена переменная типа интерфейс, то ей можно присвоить объект любого класса, который реализует этот интерфейс.

Начиная с Java версии 8, интерфейс может содержать статические методы, а также методы с реализацией по умолчанию. Поскольку статические методы не принадлежат конкретному объекту, они не являются частью API классов, реализующих интерфейс, и их необходимо вызывать с использованием имени интерфейса, предшествующего имени метода. Методы с реализацией по умолчанию, в отличие от обычных методов интерфейса, объявляются с ключевым словом default в начале сигнатуры метода и содержат тело метода, чтобы обеспечить его реализацию.