В первых двух главах мы поговорим о том, как можно вырасти профессионально, вооружившись Java 7. В разминочной первой главе мы рассмотрим небольшие синтаксические изменения, которые помогут вам работать более продуктивно. О каждом из этих изменений можно сказать: «мал золотник, да дорог». В ходе работы мы подготовим почву для обсуждения более крупной темы из этой части — нового механизма ввода-вывода в Java, которому посвящена вторая глава.

Основательный Java-разработчик, несомненно, должен знать о новейших функциях, появляющихся в языке. В Java 7 есть несколько таких функций, значительно облегчающих жизнь разработчика-практика. Но понять синтаксис этих изменений, возможно, будет непросто. Чтобы *быстро* писать *эффективный* и *надежный* код, нужно глубоко понимать, как и почему были реализованы эти новые функции. Изменения Java 7 делятся на два больших множества: проект «Монета» (Project Coin) и NIO.2.

Первое множество — проект «Монета» — это несколько небольших изменений на уровне языка. Они призваны повысить продуктивность труда разработчика, не оказывая значительного влияния на базовую платформу. К таким изменениям относятся:

- конструкция `try-with-resources` (автоматически закрывающая ресурсы);
- строки в `switch`;
- усовершенствованные числовые литералы;
- множественный `catch` (объявление нескольких исключений в блоке `catch`);
- ромбовидный синтаксис (требующий меньше шаблонного кода при работе с обобщенными типами).

Все эти изменения могут показаться незначительными. Но если исследовать семантику, которая скрывается за изменениями синтаксиса, можно лучше понять, чем язык Java отличается от платформы Java.

Второе множество изменений — это новый API ввода-вывода (NIO.2). Он кардинально модернизирует поддержку файловой системы Java, а также предоставляет новые мощные возможности асинхронной обработки. К важнейшим изменениям относятся:

- новая конструкция `Path` для ссылки на файлы и файлоподобные сущности;
- вспомогательный класс `Files`, упрощающий создание, копирование, перемещение и удаление файлов;
- встроенная навигация по дереву каталогов;
- асинхронный ввод-вывод, основанный на обратных вызовах и использовании конструкций `future`; позволяет осуществлять в фоновом режиме крупные операции ввода-вывода.

Закончив изучение первой части, вы сможете естественно думать и писать на языке Java 7. Эти новые знания будут закреплены по ходу чтения книги, поскольку все функции Java 7 используются и в последующих главах.

1 Введение в Java 7

В этой главе:

- Java как платформа и как язык;
- небольшие, но мощные синтаксические изменения;
- оператор `try-with-resources`;
- усовершенствованная обработка исключений.

Добро пожаловать в Java 7. Положение дел в этом языке может показаться вам немного непривычным. И это хорошо — так много всего предстоит исследовать теперь, когда шумиха вокруг новой версии утихла и Java 7 раскочегарился на полную мощность. Дочитав эту книгу, вы сделаете только первые шаги в огромный мир — мир новых функций, софтверного искусства, а также в мир других языков, действующих на виртуальной машине Java (JVM).

В качестве разминки мы плавно введем вас в курс Java 7, но уже на этом этапе познакомим вас с некоторыми мощными возможностями. Для начала объясним отличие, которое зачастую неправильно понимается: поговорим об отличиях языка и платформы.

Затем мы рассмотрим проект «Монета» — набор небольших, но эффективных нововведений, появившихся в Java 7. Мы расскажем, как построен процесс одобрения, внедрения и выпуска изменений на платформе Java. Обсудив этот процесс, мы перейдем к рассмотрению шести основных новых функций, появившихся в рамках проекта «Монета».

Вы изучите новый синтаксис, в частности новый способ обработки исключений (множественный `catch`), а также работу с конструкцией `try-with-resources` (использование ресурсов в блоке `try`). Эта конструкция помогает избегать ошибок в коде, работающем с файлами или другими ресурсами. Дочитав эту главу, вы сможете по-новому писать на Java и будете в полной боевой готовности для изучения масштабных тем, ожидающих впереди.

Итак, приступим к делу и поговорим о дуализме языка и платформы — пожалуй, это центральный аспект современного языка Java. Это важнейший вопрос, к которому мы многократно будем возвращаться на протяжении всей книги, поэтому с ним просто необходимо разобраться.

1.1. Язык и платформа

Важнейшая проблема, с которой мы начнем наш разговор, — это разница между языком Java и платформой Java. Удивительно, но разные авторы по-разному определяют и феномен языка, и феномен платформы. Из-за этого может возникать неясность и некоторая путаница и в том, чем отличаются язык и платформа, и в том, к чему относятся те или иные программные функции, используемые в коде приложения.

Четко очертим эти различия прямо сейчас, так как эта разница затрагивает суть самых разных тем из нашей книги. Итак, вот эти определения.

- *Язык Java* — это статически типизированный объектно-ориентированный язык, над которым мы немного пошутили в разделе «Об этой книге». Надеемся, что вы уже довольно хорошо знакомы с ним. Одно из самых очевидных качеств языка Java заключается в том, что он пригоден для чтения человеком (или, по крайней мере, должен таким быть!).
- *Платформа Java* — это программное обеспечение, предоставляющее нам среду времени исполнения. Это виртуальная машина Java (JVM), линкующая и выполняющая ваш код в том виде, в каком он ей предоставляется. Код предоставляется в виде файлов классов, непригодных для чтения человеком. Иными словами, машина не интерпретирует непосредственно файлы с исходным кодом на языке Java, а требует предварительного преобразования этого кода в файлы классов.

Одна из основных причин успеха Java как системы ПО заключается в ее стандартизации. Это означает, что Java имеет спецификации, описывающие, как должна работать платформа. Стандартизация позволяет различным производителям и участникам разнообразных проектов создавать реализации, которые теоретически должны работать одинаково. Такие спецификации не гарантируют, насколько высока будет производительность конкретной реализации одной конкретной задачи, но вполне гарантируют правильность результатов.

Существует несколько отдельных спецификаций, управляющих системой Java. Самые важные из них — это спецификация языка Java (JLS) и спецификация виртуальной машины Java (VMSpec). В версии Java 7 это разделение соблюдается очень строго; на самом деле VMSpec уже нигде не ссылается на JLS. Если вы усматриваете в этом признак того, насколько серьезно в Java 7 поставлена работа с исходными языками, не являющимися Java, то нам нравится ход ваших мыслей, продолжайте в том же духе. Ниже мы подробнее обсудим разницу между двумя этими спецификациями.

При внимательном изучении такого дуализма напрашивается вопрос: «А какова же связь между языком и платформой?» Если в Java 7 они настолько разделены, то как они стыкуются и образуют общую систему Java?

Связь между языком и платформой заключается в совместном использовании файлов классов (файлов в формате `.class`). Рекомендуем серьезно изучить определение файлов классов. Эти сведения вам точно не помешают, а знание этой темы — один из способов, позволяющих хорошему Java-программисту стать выдающимся. На рис. 1.1 показан полный процесс создания и использования кода Java.

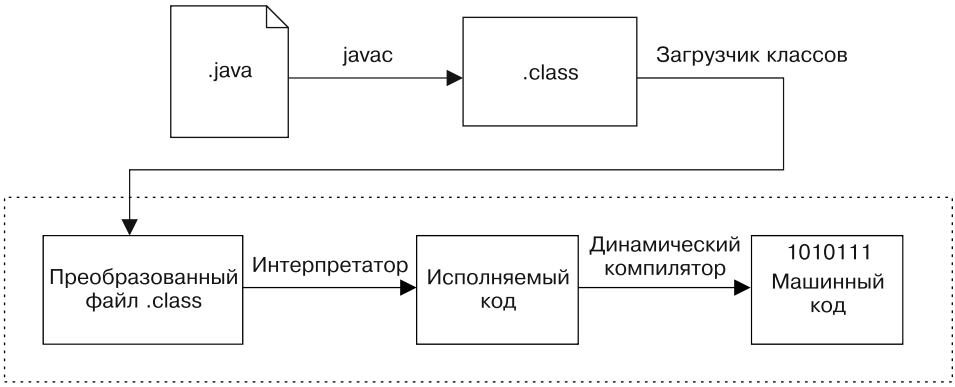

Рис. 1.1. Исходный код Java преобразуется в файлы .class, затем с ним производятся манипуляции в период загрузки, после чего он подвергается динамической компиляции

Как понятно из рисунка, код Java начинается в виде исходного кода, написанного программистом и пригодного для чтения человеком. После этого `javac` компилирует его в файл `.class`. Затем эта информация загружается в виртуальную машину Java. Обратите внимание, что манипуляции с классами и их изменение зачастую осуществляются в ходе процесса загрузки. Многие популярные фреймворки (особенно те, в названии которых присутствует слово Enterprise) преобразуют классы в ходе загрузки.

JAVA — ЭТО КОМПИЛИРУЕМЫЙ ИЛИ ИНТЕРПРЕТИРУЕМЫЙ ЯЗЫК?

Обычно Java видится разработчику как язык, компилируемый в файлы `.class` до того, как код будет использоваться на JVM. Призадумавшись, многие разработчики также объясняют, что работа байт-кода начинается с интерпретации виртуальной машиной JVM, но несколько позже код также проходит динамическую компиляцию (JIT). Но на этом многие специалисты начинают «плыть», выстраивая несколько надуманную концепцию о том, что байт-код, в сущности, является машинным кодом для воображаемого или упрощенного процессора.

На самом деле байт-код виртуальной машины Java можно считать переходной формой между исходным кодом, пригодным для чтения человеком, и машинным кодом. В техническом отношении с точки зрения теории компиляции байт-код — это действительно своеобразный промежуточный язык (intermediate language), а не настоящий машинный код. Это означает, что процесс преобразования исходного кода Java в байт-код не является компиляцией в том смысле, в каком она понимается в языках C и C++. В свою очередь, `javac` не назовешь таким же компилятором, как `gcc`. В сущности, это генератор файлов классов для обработки исходного кода Java. Настоящим компилятором в экосистеме Java является динамический компилятор (JIT) (см. рис. 1.1).

Некоторые специалисты характеризуют систему Java как «динамически компилируемую». При этом акцентируется тот факт, что истинная компиляция — это динамическая компиляция во время исполнения, а не создание файла класса во время исполнения.

Поэтому верным ответом на вопрос «Java — это компилируемый или интерпретируемый язык?» будет: «И такой и такой».

Итак, когда мы немного разъяснили разницу между языком и платформой, поговорим о некоторых заметных изменениях синтаксиса языка, появившихся в Java 7. Начнем с небольших синтаксических перемен, объединенных в рамках проекта «Монета».

1.2. Малое прекрасно — расширения языка Java, или Проект «Монета»

Проект «Монета» (Project Coin) — это свободный проект, разрабатываемый в рамках подготовки Java 7 (и 8) с 2009 года. В этом разделе мы объясним, как подбирались функции и как процесс эволюции языка происходит на уровне небольших изменений, объединенных проектом «Монета». Этот раздел будет построен в форме ситуационного исследования (case study).

ПРИЧЕМ ТУТ «МОНЕТА»

Цель проекта «Монета» — постепенно вносить в язык Java небольшие изменения. Это название в английском языке связано с игрой слов, так как существительное coin означает «монета», а глагол to coin — «чеканить». Это же слово может означать «выдумать», например «выдумать фразу». Именно в таком значении «Монета» понимается здесь — проект «чеканит» новые выражения, добавляя их в язык.

Подобные языковые игры, фантазии и вездесущие несносные каламбуры просто наводнили техническую культуру. Хотя вы, вероятно, к этому уже привыкли.

Мы считаем, что важно рассказать не только о том, *что* изменилось в языке, но и о том, *почему* появилось такое изменение. В ходе разработки Java 7 вокруг новой версии языка творилось множество всего интересного. Но члены сообщества не всегда и не вполне представляли, сколько работы потребуется на полную разработку изменений — до такой степени готовности, чтобы их можно было показать людям. Мы надеемся, что сможем пролить немного света на эту проблему, а также развеять некоторые мифы. Но если вы не очень интересуетесь тем, как развивается язык Java, можете переходить к разделу 1.3 — там мы непосредственно обсуждаем конкретные изменения в языке.

В изменении языка Java прослеживается определенная кривая усилий. Реализация некоторых языковых деталей требует значительно меньшего количества разработок, чем реализация других. На рис. 1.2 мы попытались представить различные линии разработки и относительное количество усилий, требуемое в каждом конкретном случае. Мы расположили эти линии в порядке возрастания необходимых усилий.

В принципе, лучше идти по пути наименьшего сопротивления. Это означает, что если существует возможность реализовать новую функцию как библиотеку, то так и стоит поступить. Но не все функции настолько просты, чтобы их можно было реализовывать в виде библиотек либо новых возможностей интегрированной среды разработки. При реализации некоторые функции должны быть глубже укоренены в платформе.

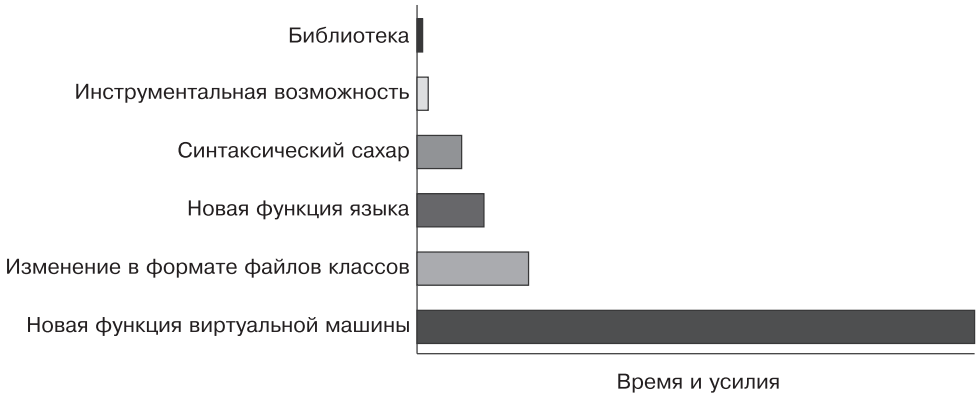

Рис. 1.2. Относительная сложность усилий при реализации новой функциональности различными способами

Ниже перечислены функции (в основном из Java 7), которые иллюстрируют элементы приведенной выше шкалы:

- *синтаксический сахар* — символы подчеркивания в числах (Java 7);
- *небольшая новая языковая функция* — конструкция `try-with-resources` (Java 7);
- *изменение формата файлов классов* — аннотации (Java 5);
- *новая функция виртуальной машины Java* — `invokedynamic` (Java 7).

СИНТАКСИЧЕСКИЙ САХАР

Некоторые функции языка называются выражением «синтаксический сахар» (syntactic sugar). Это означает, что такая синтаксическая форма избыточна — она уже есть в языке, — но синтаксический сахар существует потому, что человеку с ним легче работать.

Действует общее правило, согласно которому функции, являющиеся синтаксическим сахаром, удаляются из представления программы для компилятора на ранней стадии процесса компиляции. Принято говорить об «обессахаривании» кода для получения базового представления конкретной функции.

Внедрять в язык изменения на уровне синтаксического сахара сравнительно легко, так как для этого требуется выполнить небольшую работу. Изменения вносятся только на уровне компилятора (в случае с Java это `javac`).

Проект «Монета» (а также оставшаяся часть этой главы) посвящен переменам, расположенным в диапазоне от синтаксического сахара до небольших изменений языка.

Первая серия предложений по изменениям в рамках «Монеты» высказывалась на рассылке разработчиков «Монеты» в период с февраля по март 2009 года. Было внесено почти 70 предложений, демонстрирующих, насколько широк диапазон возможных оптимизаций. Было даже сделано шутливое предложение добавить многострочные строки в стиле «лолкотов» — юмористических надписей на фотографиях с котами, которые бывают смешными или немного издевательскими, — см. http://icanhascheezburger.com/.

Предложения «Монеты» оценивались в соответствии с совершенно простым набором правил. Автор предложения должен был сделать три вещи:

- подать подробное правильно оформленное предложение, описывающее предлагаемое изменение (принципиально это должно быть изменение языка Java, а не изменение виртуальной машины Java);
- открыто обсуждать сделанное предложение в рассылке и учитывать конструктивную критику от других участников;
- быть готовым предоставить прототипный набор заплаток (патчей), которые позволили бы реализовать предложенное изменение.

На примере проекта «Монета» хорошо видно, как язык и платформа могут развиваться в будущем: изменения обсуждаются открыто, происходит раннее прототипирование функций, работа ведется в коллективном режиме.

На данном этапе вы можете задать вопрос: «А как выглядит такое маленькое изменение в спецификации?» Одно из таких изменений, которое мы рассмотрим ниже, связано с добавлением единственного слова — `String` — в раздел 14.11 спецификации JLS. Сложно представить себе более мелкое изменение, но даже оно затрагивает сразу несколько частей спецификации.

JAVA 7 — ПЕРВАЯ ВЕРСИЯ, КОТОРАЯ РАЗРАБАТЫВАЛАСЬ В СТИЛЕ СВОБОДНОГО ПО

Java не всегда был «свободным» языком программирования. Но после соответствующего объявления, сделанного на конференции JavaOne в 2006 году, исходный код языка Java был выпущен по лицензии GPLv2 (за исключением отдельных фрагментов, исходного кода которых у Sun не было). Это случилось примерно на этапе выхода Java 6, поэтому Java 7 — первая версия языка, которая разрабатывалась по лицензии свободного программного обеспечения (OSS). При создании платформы Java в свободном режиме основное внимание было приковано к проекту OpenJDK.

В рассылках, таких как coin-dev, lambda-dev и mlvm-dev, велись развернутые дискуссии о функциях, которые могут быть добавлены в будущем. Таким образом, широкое сообщество разработчиков могло совместно заниматься созданием Java 7. На самом деле мы помогаем вести программу Adopt OpenJDK, чтобы сориентировать других разработчиков, пока мало знакомых с OpenJDK, — так мы помогаем улучшать и сам язык Java. Посетите страницу http://java.net/projects/jugs/pages/AdoptOpenJDK, может быть, вы захотите к нам присоединиться.

Изменения всегда приводят к последствиям, которые потом сказываются на всей структуре языка.

Вот действия, которые следует осуществить (или как минимум проверить их необходимость) при *любом* изменении:

- обновить JLS;
- реализовать прототип в компиляторе исходного кода;
- добавить библиотечную поддержку, необходимую для данного изменения;
- написать тесты и примеры;
- обновить документацию.