Программирование на языке Java. Отличия Java от C++

Картузов А.В.

В большинстве книг по C++ вы найдете такое же описание достоинств объектно-ориентированного программирования и доказательства того, что это—очередная ступень в развитии индустрии программирования. В чем же беда C++ и почему была создана Java?

Фактически, большинство архитектурных решений, принятых при создании Java, было продиктовано желанием предоставить синтаксис, сходный с C и C++. В Java используются практически идентичные соглашения для объявления переменных, передачи параметров, операторов и для управления потоком выполнением кода. В Java добавлены все хорошие черты C++, но исключены недостатки последнего.

1. Глобальные переменные

В старые добрые времена Фортрана, когда "настоящие мужчины" писали на ассемблере, а программы хранились на перфокартах, главным инструментом в программировании были глобальные переменные. C перенял эту особенность и несколько ее усовершенствовал—программисту приходилось по крайней мере объявлять тип глобальной переменной. Конечно же, при использовании глобальных переменных проблема состояла в том, что любая функция могла привести к широкомасштабным побочным эффектам, изменив глобальное состояние системы.

В Java единственным глобальным пространством имен является классовая иерархия. В этом языке просто невозможно создать глобальную переменную, не принадлежащую ни одному из классов.

2. Goto

Другая не слишком хорошая конструкция традиционных языков программирования—оператор goto, предназначенный для передачи управления. Всем нам на первых занятиях по программированию говорили, что можно обойтись без этого оператора, ухудшающего структуру программы и делающего ее неудобочитаемой. До того, как в C++ появился механизм работы с исключениями, goto активно использовался для выхода из циклов в исключительных ситуациях.

В Java оператора goto нет. В ней есть зарезервированное ключевое слово goto, но это сделано лишь во избежание возможной путаницы—для того, чтобы удержать программистов от его использования. Зато в Java есть операторы continue и break с меткой, восполняющие отсутствие goto в тех единственных случаях, когда использование последнего было бы оправдано. А мощный хорошо определенный встроенный в Java механизм исключений делает ненужным использование goto во всех остальных ситуациях.

3. Указатели

Указатели или адреса в памяти—наиболее мощная и наиболее опасная черта C++. Причиной большинства ошибок в сегодняшнем коде является именно неправильная работа с указателями. Например, одна из типичных ошибок—просчитаться на единицу в размере массива и испортить содержимое ячейки памяти, расположенной вслед за ним.

Хотя в Java дескрипторы объектов и реализованы в виде указателей, в ней отсутствуют возможности работать непосредственно с указателями. Вы не можете преобразовать целое число в указатель, а также обратиться к произвольному адресу памяти.

4. Распределение памяти

В строю опасных качеств C++ рука об руку с указателями идет распределение памяти. Распределение памяти в C, а значит и в C++, опирается на инь и янь ненадежного кода—на вызовы библиотечных функций malloc() и free().Если вы вызовете free() с указателем на блок памяти, который вы уже освободили ранее, или с указателем, память для которого никогда не выделялась—готовьтесь к худшему. Обратная проблема, когда вы просто забываете вызвать free(), чтобы освободить ненужный

больше блок памяти, гораздо более коварна. "Утечка памяти" (memory leak) приводит к постепенному замедлению работы программы по мере того, как системе виртуальной памяти приходится сбрасывать на диск неиспользуемые страницы с мусором. И, наконец, когда все системные ресурсы исчерпаны, программа неожиданно аварийно завершается, а вы начинаете ломать голову над этой проблемой. В C++ добавлены два оператора—new и delete, которые используются во многом аналогично функциям malloc() и free(). Программист по-прежнему отвечает за то, чтобы каждый неиспользуемый объект, созданный с помощью оператора new, был уничтожен оператором delete.

в Java нет функций ьфддщс() , free(). Поскольку в ней каждая сложная структура данных—это объект, память под такие структуры резервируется в куче (heap) с помощью оператора new. Реальные адреса памяти, выделенные этому объекту, могут изменяться во время работы программы, но вам не нужно об этом беспокоиться. Вам даже не придется вызывать free () или delete, поскольку Java—система с так называемым сборщиком мусора. Сборщик мусора запускается каждый раз, когда система простаивает, либо когда Java не может удовлетворить запрос на выделение памяти.

5. Хрупкие типы данных

C++ получил в наследство от C все обычные типы данных последнего. Эти типы служат для представления целых и вещественных чисел различных размеров и точности. К несчастью, реальный диапазон и точность этих типов колеблется в зависимости от конкретной реализации транслятора. Поведение кода, который прекрасно транслируется и выполняется на одной машине, может радикально отличаться при смене платформы. Различные трансляторы C++ могут резервировать под целый тип 16, 32 или 64 бита в зависимости от разрядности машинного слова.

В Java эта проблема решена, поскольку, как вы увидите в главе 4, в ней для всех базовых числовых типов независимо используются определенные соглашения, не зависящие от конкретной реализации среды. Не исключено, что на некоторых архитектурах реализовать работу с числами таких размеров в интерпретаторе Java окажется трудно, либо реализация будет неэффективна, однако это единственный способ гарантировать воспроизводимые результаты на широком спектре платформ.

6. Ненадежное приведение типов

Приведение типов в C и C++—мощный механизм, который позволяет произвольным образом изменять тип указателей. Такой техникой надо пользоваться с крайней осторожностью, поскольку в C и C++ не предусмотрено средств, позволяющих обнаруживать неправильное использование приведения типов. Поскольку объекты в C++—это просто указатели на адреса памяти, в этом языке во время исполнения программы нет способа обнаруживать случаи приведения к несовместимым типам.

Дескрипторы объектов в Java включают в себя полную информацию о классе, представителем которого является объект, так что Java может выполнять проверку совместимости типов на фазе исполнения кода, возбуждая исключение в случае ошибки.

7. Ненадежные списки аргументов

C++ гордится своей возможностью передавать указатели на произвольные типы в списках аргументов переменной длины, известных под названием varargs. Интерфейс varargs—простое расширение, основанное на возможности приведения любого адреса к произвольному типу, при этом заботы о проверке допустимости типов ложатся на плечи программиста.

Было бы прекрасно, если бы в Java существовала безопасная возможность объявлять и передавать списки аргументов переменной длины, но в Java 1. 0 такие средства отсутствуют.

8. Раздельные файлы заголовков

Когда-то великим достижением считались файлы заголовков, в которые можно было поместить прототипы классов и распространять их вместе с оттранслированными двоичными файлами, содержащими реальные реализации этих классов. Поддержка этих файлов заголовков (ведь они должны соответствовать реализации, их версия должна совпадать с версией классов, хранящихся в оттранслированных двоичных файлах) становилась непосильной задачей по мере роста размеров библиотек классов.

В Java такое невозможно, поскольку в ней отсутствуют файлы заголовков. Тип и видимость членов класса при трансляции встраиваются внутрь файла *.class (файла с байт-кодом). Интерпретатор Java пользуется этой информацией в процессе выполнения кода, так что не существует способа получить доступ к закрытым переменным класса извне.

9. Ненадежные структуры

C++ пытается предоставить программисту возможность инкапсуляции данных посредством объявления структур (struct) и полиморфизм с помощью объединений (union). Эти две конструкции прикрывают критические и катастрофические машинно-зависимые ограничения по размеру и выравниванию данных.

В Java нет конструкций struct и union, все это объединено в концепции классов.

10. Препроцессорная обработка

Работа препроцессора C++ которого заключается в поиске специальных команд, начинающихся с символа #. Эти команды позволяют выполнять простую условную трансляцию и расширение макроопределений.

Java управляется со своими задачами без помощи препроцессора, вместо принятого в C стиля определения констант с помощью директивы #define в ней используется ключевое слово final.

11. QED

Quod erat demonstrandum—латинское "что и требовалось доказать". Просто прочитав обо всех этих проблемах, даже если вам еще не приходилось иметь с ними дела, вы должны быть готовы погрузиться в материал следующей главы.