Реферат на тему “Из C++ в Java”

(Konstantin Burlachenko, with updates from 05-NOV-2019)

"Much of the relative simplicity of Java is - like for most new languages - partly an illusion and partly a function of its incompleteness. Java isn't platform independent - it is a platform." - B.Stroustroup, [2]

1. Введение и цель создания документа 1
2. Требования к читателю 1
3. Процесс построения программы в Java 2
4. Как собрать тестовую программу по шагам из командной строки 3
5. Отличие от C++ с точки зрения JAVA 4
6. Общее для C++ и Java 6
7. Компоненты JAVA платформы 7
8. Код снипеты для базовых вещей 8
9. Ограничения на организацию кода при написании программ на Java 11
10. Диапазон доступных значений некоторых примитивных типов в Java [5] 11
11. Разнообразное в Java 11
Ссылки 12

1. Введение и цель создания документа

Я столкнулся с необходимостью понимать JAVA код. Так как на этом языке я не писал давно, то первым делом я хотел найти возможность, которая конструкциям языка C++ ставит эквивалентные конструкции языка JAVA, однако такой документ я не нашёл в интернете.

И вот этот документ создан, чтобы в будущем можно быстро получить сведения про JAVA, на основе C++. В заголовке статьи указана точка зрения Б.Страуструпа на язык JAVA. Более подробную информацию можно прочитать в [2].

Если у вас будут предложение по улучшению - я рад к взаимодействию по улучшению документа. Моя домашняя почта: burlachenkok@gmail.com

2. Требования к читателю

Знакомство с C++.

3. Процесс построения программы в Java

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**. По сути здесь располагается исходный код.

Эти файлы подаются на вход Java компилятору, который транслирует команды в специальный Java байт-код. Компилятор представлен утилитой **javac (java compiler)**.

Компилятор JAVA (ровно как и например компилятор C/C++) в процессе чтения текста программы разделяет её на:

- Пробелы (white spaces)
- Комментарии (comments)
- Основные лексемы (tokens)
    - Идентификаторы (identifiers)
    - Ключевые слова (key words). Все ключевые слова перечислены здесь [6]
    - Литералы служащие для обозначения конкретных значений для констант (literals)
    - Разделители (separators)
    - Операторы (operators)

Результат работы компилятора сохраняется в бинарных файлах с расширением **.class**.

По сути это бинарный код известный также как файлы с расширением **.obj** в системах сборки проекта(toolchains) для C++.

Java-приложение есть набор таких скомпилированных файлов. Их можно запаковать в архив, или не запаковывать в архив.

Java-приложение состоящее из таких файлов, подается на вход виртуальной машине, которая начинает их исполнять, или интерпретировать, так как сама является программой. Сначала байт-код загружается в систему, как правило, в виде **class** - файлов.

JVM внимательно анализирует, что все они подчиняются общим правилам безопасности Java, а не были созданы злоумышленниками с помощью каких-то других средств. Затем во время исполнения программы, интерпретатор легко может проверить каждое действие на допустимость. Возможности классов, которые были загружены с локального диска или по сети, могут различаться (конечный пользователь легко может назначать или отменять конкретные права). Виртуальная машина реализована программой **java**

Например, апплет по умолчанию никогда не получит доступ к локальной файловой системе. Такие встроенные ограничения есть и во всех стандартных библиотеках *Java*.

Набор инструкций поддерживаемый JVM является неотъемлемой частью платформы *Java*.

Когда загружается очередной **.class**-файл, содержащий описание нового класса, JVM порождает объект класса **java.lang.Class** который будет хранить его структуру.

Зачем нужен *Class*:

1. С помощью *Class* реализована поддержка статических (**static**) полей и методов класса.
2. Наконец, этот класс содержит ряд методов, полезных для разработчиков.

4. Как собрать тестовую программу по шагам из командной строки

1. Создать текстовый файл

```
// Main.java
public class Main
{
    public void f(){}
    public static void main(String args [])
    {
        System.out.println("Hello World");
    }
}
```

2. Скомпилировать исходный файл *.java в *.class

javac Main.java

3.а Запустить на выполнение через виртуальную машину java

java Main

3.б Создать JAR архив поместив туда скомпилированный класс, и указав класс с точкой входа

*jar cfe myJar.jar Main *.class*

3.в Запустить на выполнение код находящийся в JAR архиве

java -jar myJar.jar

4. Получить дисассемблерный код

Если у вас нет исходного кода class файла, то через JAVA можно получить описание класса через **javap**.

>> javap Main.class

Compiled from "Main.java"

public class Main {

public Main();

public void f();

public static void main(java.lang.String[]);

}

Дополнительный аргумент –c позволяет получить скомпилированный JAVA байт-код:

javap -c Main.class

5. Отличие от C++ с точки зрения JAVA

1. Создатели отказались от множественного наследования. Было решено, что оно слишком усложняет и запутывает программы.
2. В Java есть всего 8 типов данных, которые не являются объектами. Они были определены с самой первой версии и никогда не менялись. Это 5 целочисленных типов: *byte, short, int, long*, символьный *char*, 2 дробных типа *float* и *double*, и наконец булевский тип *boolean*
3. Авторы *Java* опирались на широко распространенный язык программирования C++. Поэтому многие конструкции схожи.
4. 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.
5. Таким образом, программист хоть и не должен думать явно об удалении, но всё равно должен следить лишь за тем, чтобы не оставалось ссылок на ненужные объекты.
6. Возможность создание многопоточных приложений было внесено с самой первой версии *Java*
7. Резервирование и очищение памяти определяет разработчик *JVM*, а не программистом или компилятором
8. Указатели на физическую память отсутствуют принципиально. В JAVA отсутствуют как таковые указатели
9. Есть встроенное средство документации кода (javadoc).

Его синтаксис поддерживается **doxygen-ом**

10. В C++ char это один байт, а в Java это один Unicode символ
11. Строка в C++ это как правило zero-terminated ASCII string. В Java записанная строка *“string”* немедленно становится экземпляром класса *String*
12. В C++ действия оператора **>>** для знаковых типов не определено. Может быть выполнен как логический так и арифметический сдвиг вправо.
13. В ***Java*** оператор **>>** выполняет арифметический сдвиг. Сдвиг выполняется вправо, а знаковый бит заполняется тем же значением, который у него было до сдвига. А вот оператор **>>>** выполняется просто логический сдвиг, заполняя старший бит нулём.
14. В C++ копирование всегда происходит по значению.

В Java есть **примитивные типы для них происходит копирование по значению**.
А есть **ссылочные типы – это все классы, интерфейсы и массивы**.

15. Все объекты ссылочного типа порождаются с помощью оператора **new, но строковые литералы исключение – их надо создавать с помощью синтаксиса создания строк**
16. В язык встроена Reflection. Например создать объект класса только по имени класса можно через *(Point)Class.forName("Point").newInstance()*

17. Есть базовый класс Object от которого наследуются все классы неявно в цепочке наследования.
18. *Object.getClass(), Object.getName(), Object.equal(…), Object.hashCode(), Object.toString()*
19. Нету множественного наследования, но чтобы иметь возможность реализовывать интерфейсы есть специальный синтаксис для этого.
20. Если две переменные ссылочного типа сравниваются через < == > то выполняется просто проверка ссылок (как адресов) никакие operator == не вызываются. Экивалент operator == есть *Object.equal()*
21. Если одинаковая строковая константа используется для создания нескольких строк, то по факту все эти строки будут по сути одним объектом. Конечно это делается компилятором и уловить что это надо сделать во время выполнения он уже не сможет. Для получения уникальной ссылки на строку среди всех строк в приложении требуется вызвать метода ***String.intent()***
22. Скомпилированные классы можно хранить в архивах формата **ZIP** или **JAR**.
23. Программа, написанная на *Java*, является набором классов. В *C++* это не так.

В языке C++ на глобальном уровне могут быть объявлены – функции, переменные, операторы, классы, типы.

24. Все примитивные типы дополняются дополнительными классами обёртками. Так как *int* это примитивный тип, и он есть по сути просто целочисленное число, то конструкция

int.parseInt("1") – не имеет смысла *[4]*

С другой стороны есть класс Integer, такой же как и другие классы.

Integer.parseInt("1") – имеет смысл

Классы обёртки наследуются от *Object*, а примитивные типы нет.

итивный тип	обёртка
n	n
	ter

25. С *JAVA-5* появился auto-boxing и это выполняется автоматически, хотя это может приводить к тонким некорректным действиям кода, а не как задумывал программист.
26. Абстрактный класс это отдельная концепция и есть ключевое слово abstract для маркирования своего класса как **abstract**

27. Интерфейс это отдельная концепция, кстати для него **abstract** не требуется. Все методы интерфейса являются по умолчанию **public abstract**. И все данные **public final static**.

Ещё раз это так же, даже без указания этих модификаторов.

28. Интерфейс может наследовать более одного интерфейса, что не разрешено при наследовании классами других классов. Например:
    public interface Drawble extends IColor, Iresize{}
29. Все методы кроме статических и приватных в JAVA являются тем, что в терминологии C++ называют виртуальными.
30. Уровень доступа в переопределенных методах (теоретически) можно назначить любой в C++. В Java нельзя ограничивать уровень доступа сильнее чем в базовом классе.
31. В JAVA массивы имеют встроенное поле *length* для отслеживания длины. Это поле имеется всегда. Массив из дополнительных полей кроме **length** для отслеживания количество элементов в массиве содержит **clone()** для клонирования
32. Базовым типом для массива может быть как примитивный тип, так и абстрактный класс
33. Массив является объектным типом. Массив может инициализироваться с помощью фигурных скобок или инициализрующей конструкции. Массив по сути является не столько низкоуровневой концепцией как в C++
34. Двумерные массивы есть массивы массивов
35. Поскольку массив является объектным типом данных, то можно попытаться представить себе, как бы выглядело объявление класса такого типа. На самом деле объявление вашего массива не хранятся в файлах или еще в каком-нибудь формате.
36. Также никакой класс не может наследоваться от *Массива*.
37. Массив, основанный на типе A, можно привести к массиву, основанному на типе B, если сам тип A приводится к типу B
38. Объект в JAVA всегда "*помнит*", от какого типа он был порожден
39. Оператор **goto** в JAVA запрещён
40. Метод может сгенерировать исключение только ели он задекларировал это в описании метода посредством

    private String doSomething() throws FileNotFoundException,IOExceptin {}

41. В JAVA для работы с коллекциями в java.util имеется: *Vector, Hashtable, Set, List, Map, SortedMap, SortedSet, BitSet*
42. Чтобы получить версии коллекций с поддержкой синхронизацией (для работы из нескольких потоков выполнения)требуется вызывать Collections.synchronizedList, Collections.synchronizedMap
43. Коллекции предназначены для работы с объектами, и не могут содержать элементами простые типы
44. В JAVA есть ключевое слово **synchronized** применяемое для блока и как модификатор метода.
45. В JAVA в базовом классе **Object** есть **wait()** который позволяет вызывающему коду заснуть на объекте. Пробуждение осуществляется другим потоком вызывающие **notifyAll()** или **notify()**
46. Судя по всему в JAVA нет ***typedefs***

47. В JAVA не надо ставить точку с запятой после объявление класса.
48. В JAVA нет заголовочных файлов.

https://stackoverflow.com/questions/40997215/equivalent-to-c-header-files-in-java

49. При доступе out-of-bound генерируется Exception. C++ массивы ближе к реальному железу и там это и есть по сути **undefined behavior**.
50. В Java если класс поддерживает интерфейс java.io.Serializable то он может быть сериализован. Пример
    a. *ByteArrayOutputStream os = new ByteArrayOutputStream();*
    b. *Object objSave = new Integer(10);*
    c. *ObjectOutputStream oos = new ObjectOutputStream(os);*
    d. *oos.writeObject(objSave);*

6. Общее для C++ и Java

1. Строгая типизация
2. Значение операторов <;> <,>
3. Оформление комментариев и правила работы с комментариями. Например запрещены вложенных комментариев
4. Правила именование идентификаторов
5. Правила написания и представления для целых типов
6. **int эквивалентны int32_t, long аналогичен C++, т.е. это всегда 4 байта.**
7. Суффиксы около целочисленный констант (L)
8. Правила написания чисел с плавающей точкой и суффиксы **f**, **d**
9. Почти все операторы как в C++ и в Java одинаковы. Кажется есть только несколько операторов которых нет с идентичным синтаксисом в C++. А именно это:

< **>>>** >

<**instanceof**>

<**syncronized**>

10. Объявление переменной
11. Обычные неявные унарные, бинарные преобразования
12. Преобразования типов
13. Оператор “;” есть отсутствие оператора как в C++ так и в JAVA
14. *If,switch,break,continue,return* имеют одинаковый смысл. Кроме одного нюанса в JAVA после break может быть указана метка. Это приведёт к немедленному переходу на конец(фактически выход) из именованного блока этой меткой.
15. Switch может иметь в качестве выражение только целочисленные значения
16. try/catch keywords имеют одинаковый смысл.

(Правда для throw в Java нужно кидать объект приводимый к типу Thowable, а в C++ ограничения на тип нет)

17. В C++11 и JAVA есть встроенная поддержка потоков. (класс Thread или интерфейс Runable передаваемый объекту класса Thread).
18. Есть модификатор volatile. Но как мне известно volatile не гарантирует, что будет выполнен сброс кеша после выполнения записи write на переменную.
19. И там и там есть тип перечислимый тип *enum*

7. Компоненты JAVA платформы

Name of component	Value for Java eco-system
Java Language Specification, JLS	Спецификация языка программирования Java
Java Development Kit, JDK	Утилиты для разработки, стандартных библиотек классов и демонстрационных примеров. Оно не содержит текстовых редакторов, но содержит **javac (java compiler)** Типовой путь под Windows **"C:/Program Files/Java/jdk-12.0.2/bin"**
JVM specification	Спецификация для создателей виртуальных машин. Виртуальная машина реализована программой **java**: https://docs.oracle.com/javase/8/docs/technotes/tools/windows/java.html Эта программа запускает java приложение. Есть небольшая вариация программы java – это программа **javaw**. Она не создаёт консольное окно. Входным параметров этой программы должно быть **имя типа** содержащего статическую публичную функцию, которая является точкой входа. *public class MyClass { public static void main(String[] args){} }* Если класс лежит в директории a/b/c/MyClass.java, то после компиляции надо запустить команду: **java a.b.c.MyClass**

Name of platform	Value
Java 2 Platform, Standard Edition (J2SE)	Предназначается для использования на рабочих станциях и персональных компьютерах
Java 2 Platform, Enterprise Edition (J2EE)	Содержит все необходимое для создание сложных, высоконадежных, распределенных серверных приложений.
Java 2 Platform, Micro Edition (J2ME)	Усечение Standard Edition для того, чтобы удовлетворять жестким аппаратным требованием небольших устройств

8. Код снипеты для базовых вещей

№	JAVA вариант	Решаемая задача или C++ аналог

1	**\u1B05**	Вставить Unicode символ в текст программы.
2	**final int a=1;**	Аналог **const int a = 1;**
3	System.out.println	**printf(…)**
4	new Point(1,2) создание динамически объекта и заведение подсчёта ссылок на него.	new Point(1,2) выделение памяти и конструирование объекта
5	class Parent { } class Child extends Parent { }	class Parent { }; class Child : public Parent { };
6	bool test = p instanceof Child;	bool test = (dynamic_cast<Child>(p) != nullptr);
7	Object.finalize() Будет вызван когда заблогоросудиться Garbage Collector-у	Desctructor Правила вызова чётко определены
8	Обращение к имени класса: Reflect Полное имя типа java.lang.reflect	Обращение к имени класса: Reflect Полное имя типа(Fully qualified name) java::lang::reflect
9	env.variable **classpath** служит как доп.пути по которым будут искаться пакеты при их импорте. https://en.wikipedia.org/wiki/Classpath_(Java) Это могут быть или конкретные JAR файлы, или пути к директориям.	В C++ это понятие отсутствует. Может быть похожая концепция это дополнительный INCLUDEPATH который передаётся **компилятору**. Это понятие эквивалентно PYTHONPATH для питона.
10	Пакет является концепцией схожей с **namespace**. В Java объявление пакета может располагаться только на верхнем уровне *package mainPack.subPack;* Более того, судя по всему директории на файловой системе должны соответствовать структуре пакетов.	В C++ может располагаться где угодно namespace mainPack{namespace subPack {}}
11	**import std.*;** **import std.cout;** список операторов **import** может стоять после оператора package, но до любого определения классов в исходном Java-файле.	**using namespace std;** **using std::cout;** using может располгаться где угодно.

	Конфлиткт неиспользуемых имён не является ошибкой.	Конфлиткт неиспользуемых имён не является ошибкой.
12	class A{}	class A{};
13	Возможно импортировать пакет и использовать только один тип (или даже ни одного) из него	Возможно импортировать пакет и использовать только один тип (или даже ни одного) из него
14	package first; public class FirstClass { } public interface MyInterface { }	namespace first { class FirstClass { } ; class MyInterface { }; }

15	package first;
	class FirstClass { }
	interface MyInterface { }
	
	// Объявление класса и интерфейса в пакете.
	// В JAVA класс и интерфейс по умолчанию доступен только классам из этого же пакета только.
	// Эта видимость известна как **default**
		namespace first
		{
		class FirstClass { };
		class MyInterface { };
		}

16	Область видимости доступного пакета - вся программа, то есть любой класс может использовать доступный пакет	Чтобы достучаться до класса в пространстве имён сначала требуется выполнить команду препроцессора C - **#include**, чтобы объявления стали доступны.
17	Область видимости локальных переменных начинается с момента их инициализации и до конца блока, в котором они объявлены	Область видимости локальных переменных начинается с момента их инициализации и до конца блока, в котором они объявлены
18	this Возможно вызывать другой конструктор.	this В C++11 появилась так же как в JAVA вызывать другой конструтор (delegate ctor)
19	Уровни доступа к полям, методам, ctor: *private* *protected* *public* *(none) default* - доступ только изнутри пакета	Уровни доступа к полям, методам, ctor: *private* *protected* *public* *(none) default – отсутствует такое понятие в C++*

20	Уровни доступа к пакетам отсутствуют	Уровни доступа к пространством имён так же отсутствуют
21	Уровни доступа к классам и интерфейсам: (none) default – доступ только изнутри пакета public – публичный доступ	Такое понятие отсутствует или же можно сказать, что все классы имеют доступ типа **public** в C++
22	public static void main(String[] args) {System.exit(-1);}	int main(int argc, char**argv) {return -1;}
23	Примитивные типы передаются по значению как аргументы	Примитивные типы передаются по значению как аргументы
24	Ссылочные типы передаются по ссылке	Ссылочные типы передаются по ссылке, но ссылки в C++ и указатели имеют специальный синтаксис для этого
25	Перегруженные методы существуют	Перегруженные методы существуют в языке C++ с 1998-го года
26	null	Nullptr, 0, (void*)0

27	abstract class Operation {
	public abstract int calc(int a);
	}
	
	Дополнительные комментарии про **Abstract**
	
	abstract class Operation
	Запрещает создание объектов этого класса
	
	public abstract int calc(int a);
	Указывает, что метод должен быть реализован в производном (по уровню насоедования) классе.
	
	Абстрактный метод не может быть private, native, static.
		class Operation {
		public:
		virtual int calc(int a) = 0;
		}

28	Для связки с языками которые поддерживают указатели есть специальное ключевое слово **native** Пример как реализовать связку JAVA с C кодом представлено здесь: https://stackoverflow.com/questions/6101311/what-is-the-native-keyword-in-java-for	Такое понятие отсутсвует
29	У классов есть один базовый класс и много интерфейсов	Нет как таковых интерфейсов выделенных в синтаксическую сущность, но

		есть множественное наследование.
30	Объвление массива int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7,8} };	Объвление массива int myNumbers[][4] = { {1, 2, 3, 4}, {5, 6, 7, 8} };
31	double result = (double) 19 / 5; // C-style cast	double result = (double) 19 / 5; // C-style cast
32	String concatentation is build-in String g = “1”+”2”;	std::string g = std::string(“1”) + std::string(”2”);

33	public class Secretary extends Employee
	{
	public void f()
	{
	super.f();
	}
	}
		class Secretary: public Employee
		{
		public:
		virtual void f()
		{
		Employee::f();
		}
		}

34	throw new ExceptionType();	throw ExceptionType();

9. Ограничения на организацию кода при написании программ на Java

В модуле компиляции может быть максимум один **public** тип, и **его имя и имя файла должны совпадать**. Не **public** типов может быть на самом деле несколько. Не **public** типы, имена которых не совпадают с именем файла должны использоваться только внутри этого модуля компиляции.

На практике программисты зачастую помещают в один модуль компиляции ровно один тип, независимо от того, **public** он или нет. Это существенно упрощает работу с ними.

10. Диапазон доступных значений некоторых примитивных типов в Java [5]

Data Type	Size	Description
Byte	1 byte	-128 to 127
Short	2 bytes	-32,768 to 32,767
Int	4 bytes	-2,147,483,648 to 2,147,483,647
Long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
Float	4 bytes	3.4e−038 to 3.4e+038. Sufficient for storing 6 to 7 decimal digits
Double	8 bytes	1.7e−308 to 1.7e+038. Sufficient for storing 15 decimal digits
boolean	1 bit	True/false
Char	2 bytes	Sufficient to store ASCII

11. Разнообразное в Java

Статический контекст выполнения

К статическому контексту относят

- статические методы

- статические инициализаторы

- инициализаторы статических полей

Ссылочные типы:

- Объекты

- Массивы

- Интерфейсы

Потоки данных и работы с ними

- FilterInputStream, FilterOutputStream

- ByteArrayInputStream, ByteArrayOutputStream

- BufferedInputStream, BufferedOutputStream

- DataInputStream и DataOutputStream

На количество вызовов `finalize()` нет никакого подсчёта

```
// Main.java

class Blah
{
  @Override
  public void finalize()
  {
     System.out.println("finalizing!");
  }
}

public class Main
{
  private static void f()
  {
    Blah blah = new Blah();
    blah.finalize();
  }

  public static void main(String[] args)
  {
    System.out.println("start");
    f();
    System.out.println("before call gc");
```

```
    System.gc(); // suggests that the Java Virtual Machine expend effort toward
recycling unused objects
    System.out.println("after call gc");
    System.out.println("end");
  }
}
```

Ссылки

[1] Sun Java Course, 2003 - Sun Microsystems из Московского физико-технического института(МФТИ)

[2] http://www.stroustrup.com/bs_faq.html

[3] https://stackoverflow.com/

[4] https://stackoverflow.com/questions/8660691/what-is-the-difference-between-integer-and-int-in-java

[5] https://www.w3schools.com/java/java_data_types.asp

[6] https://www.w3schools.com/java/java_ref_keywords.asp