Лекция 25

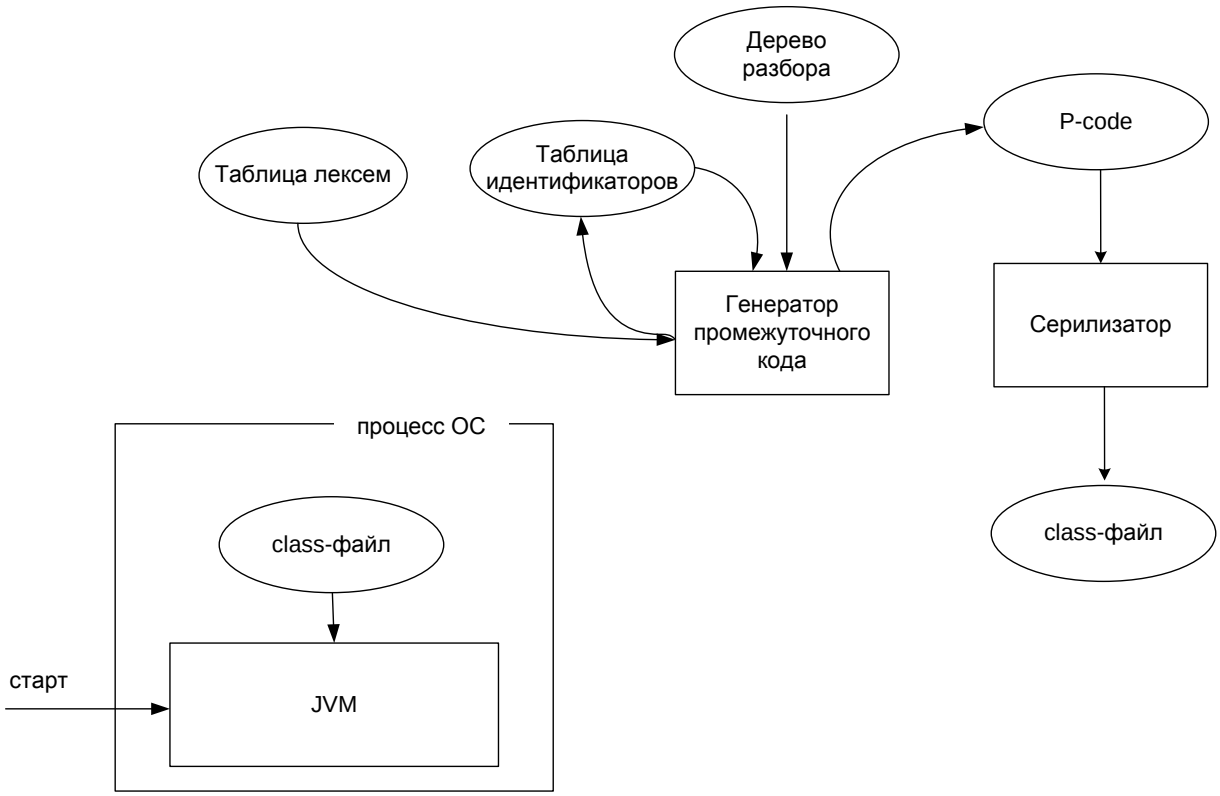
БГТУ, ФИТ, ПОИТ, 3 семестр
Конструирование программного обеспечения

Генерация кода. Общие сведения о виртуальной машине Java

Подходы к разработке трансляторов:

- часть операционной системы;
- для аппаратной платформы (ассемблер);
- реализации для одной программной платформы;
- реализация для одной программой платформы, но для разных процессоров;
- интерпретаторы;
- несколько реализаций для разных платформ;
- кроссплатформенные реализации (Java);
- компиляторы-интерпретаторы (компиляция + интерпретация);
- разработка стандарта и стандартизация (Java, C++,C#)

1. Генерация кода языка Java:



2. **Основные свойства платформы Java** (Sun Microsystems, 1995г.)

Две революционных концепции:

- «Написать один раз и использовать везде» (WORA);
- автоматическое управление памятью.

Типы программ, создаваемые в рамках технологии Java:

- приложения – программы, выполняемые в среде платформы Java;
- апплеты (applet) – программы, написанные на языке Java, откомпилированные в байт-код. Выполняются в браузере с использованием виртуальной Java-машины;
- сервлеты (servlet) и корпоративные бины – Java-программы, серверные компоненты распределенных приложений. Java интерфейс для расширения функциональных возможностей сервера.

Возможности:

- написание программного обеспечения на одной платформе и его запуск на другой платформе;
- создание программ, работающих в веб-браузере;
- разработка приложений на стороне сервера;
- создание высокоспециализированных приложений или служб;
- создание многофункциональных приложений для мобильных устройств, удаленных процессоров, микроконтроллеров, беспроводных модулей, датчиков, шлюзов и других категорий электронных устройств.

Кроссплатформенность:

можно создать Java-приложение на одной платформе, скомпилировать в байт-код и запустить его на другой платформе, поддерживающей виртуальную машину Java (JVM).

JVM служит уровнем абстракции между кодом и оборудованием и состоит:

- спецификации JVM – первая часть виртуальной машины Java (не определяет детали реализации JVM);
- реализации JVM – это готовая JVM (виртуальная машина Java);
- экземпляр JVM – загружается, как приложение, является экземпляром виртуальной машины.

Список языков программирования, использующего в качестве среды выполнения виртуальную машину Java:

- Clojure — функциональный язык, диалект Lisp;
- Groovy — сценарный язык;
- **Kotlin** — объектно-ориентированный язык для индустриальной разработки
- **Scala** — объектно-ориентированный и функциональный язык;
- Ceylon — объектно-ориентированный язык со строгой статической типизацией;
- JRuby — реализация Ruby;
- Jython — реализация Python;
- Nashorn — реализация JavaScript.

Некоторые из этих языков интерпретируются, а некоторые компилируются в байт-код Java и компилируются «на лету» во время исполнения.

Есть возможность реализовать интерпретатор некоторого языка на Java (как, например, сделано для языка JRuby).

3. Структура JVM:

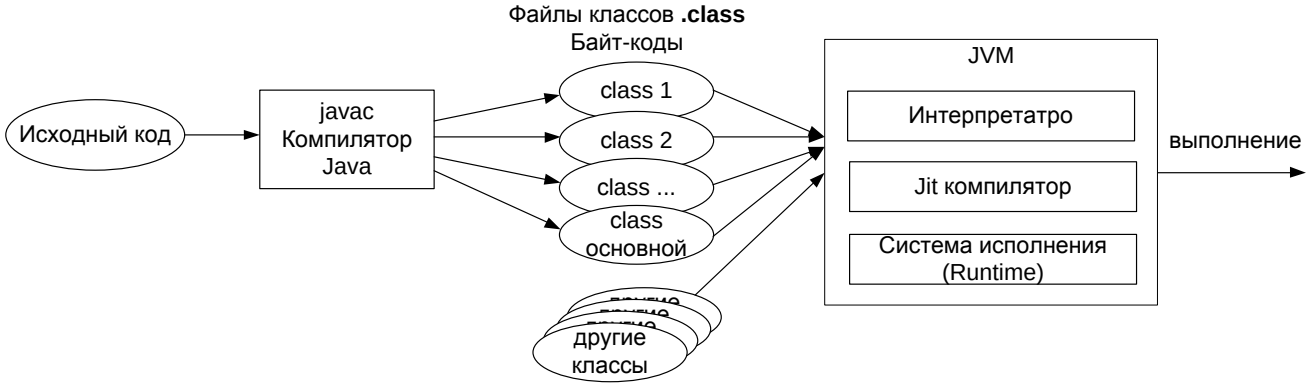
- код приложения выполняется в контейнере;
- защищённая среда выполнения программы;
- возможность автоматического управления памятью;
- кроссплатформенная среда исполнения;
- технология Jast-in-time (JIT).

Платформа Java – языковая система с компиляцией в промежуточное представление (***байт-код, в формате big-endian***), которое представляет собой программу, выполняемую в виртуальной среде.

Среда выполнения Java (JRE – java runtime environment) — это набор программных средств, обеспечивающих выполнение Java-программы на любой аппаратной платформе и в среде любой ОС.

В состав JRE входят: виртуальная машина Java (JVM) и набор стандартных библиотек Java.

Выполнение приложения в платформе Java



Виртуальная машина Java (JVM) поддерживает внутреннюю среду исполнения байт-кода, которая имеет следующие особенности:

- динамическая объектно-ориентированная модель представления — Java-программа – это набор классов, для каждого класса при компиляции создается двоичный файл с расширением .class, содержащий байт-код методов и описание структуры класса (описание данных и заголовки методов);
- JVM при загрузке создает объект класса Class – описатель класса, в который загружаются данные из каждого файла .class. Описатель класса содержит также ссылки на родительский класс, интерфейсы и т.п.;
- «самоопределение» – элементы внутреннего представления Java- программы;
- динамическое связывание внешних методов;
- контроль за объектами (включая процедуры автоматического сбора мусора);
- многопоточность, реализованная в Java на уровне языка.

4. Виртуальная машина Java

Типы данных, допустимые в JVM, подразделяются на примитивные типы и ссылочные.

4.1.Примитивные типы:

Тип	Размер (биты)	Диапазон	Описание типа
byte	**8**	от -128 до 127	1-байтное целое со знаком
short	**16**	от -32 768 до 32 767	2-байтное целое со знаком
int	**32**	от -2147483648 до 2147483647	4-байтное целое со знаком
long	**64**	от -9223372036854775808 до 9223372036854775807	8-байтное целое со знаком
float	**32**	от $\pm 1{,}5*10^{-45}$ до $\pm 3{,}4*10^{33}$	4-байтное число с плавающей точкой
double	**64**	от $\pm 5*10^{-324}$ до $\pm 1{,}7*10^{306}$	8-байтное число с плавающей точкой
char	**16**	UNICODE	2-байтный символ Unicode

Размеры типов в языке Java и в JVM являются **постоянными**, не зависящими от платформы.

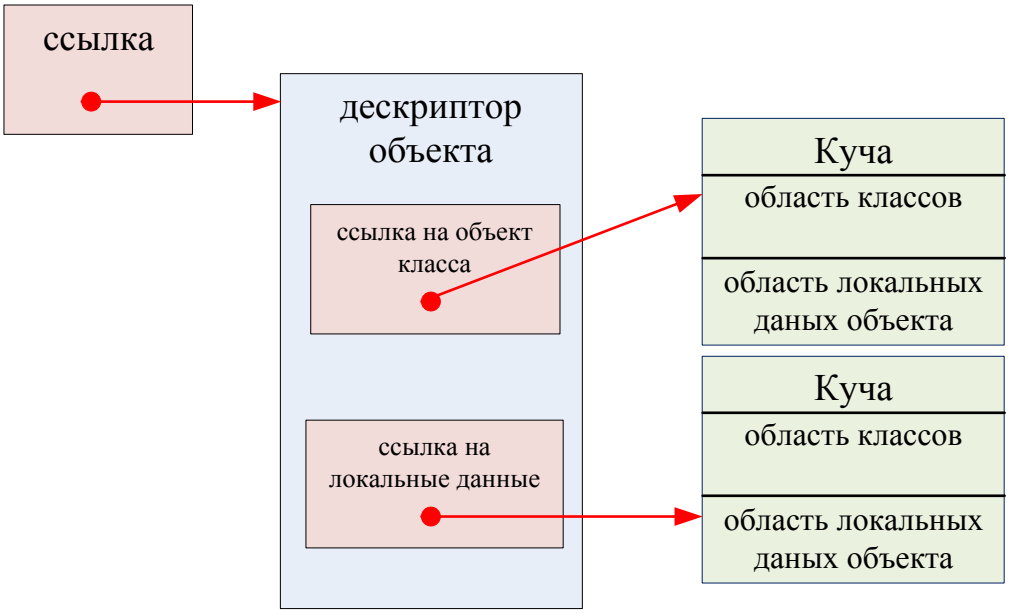
Тип **boolean** (примитивный тип языка Java) представляется в JVM типом **int**.

Примитивный тип **returnAddress** в JVM не имеет соответствия в языке Java и представляет собой указатель на команду байт-кода Java и используется в качестве операнда в командах передачи управления.

4.2.Ссылочные типы

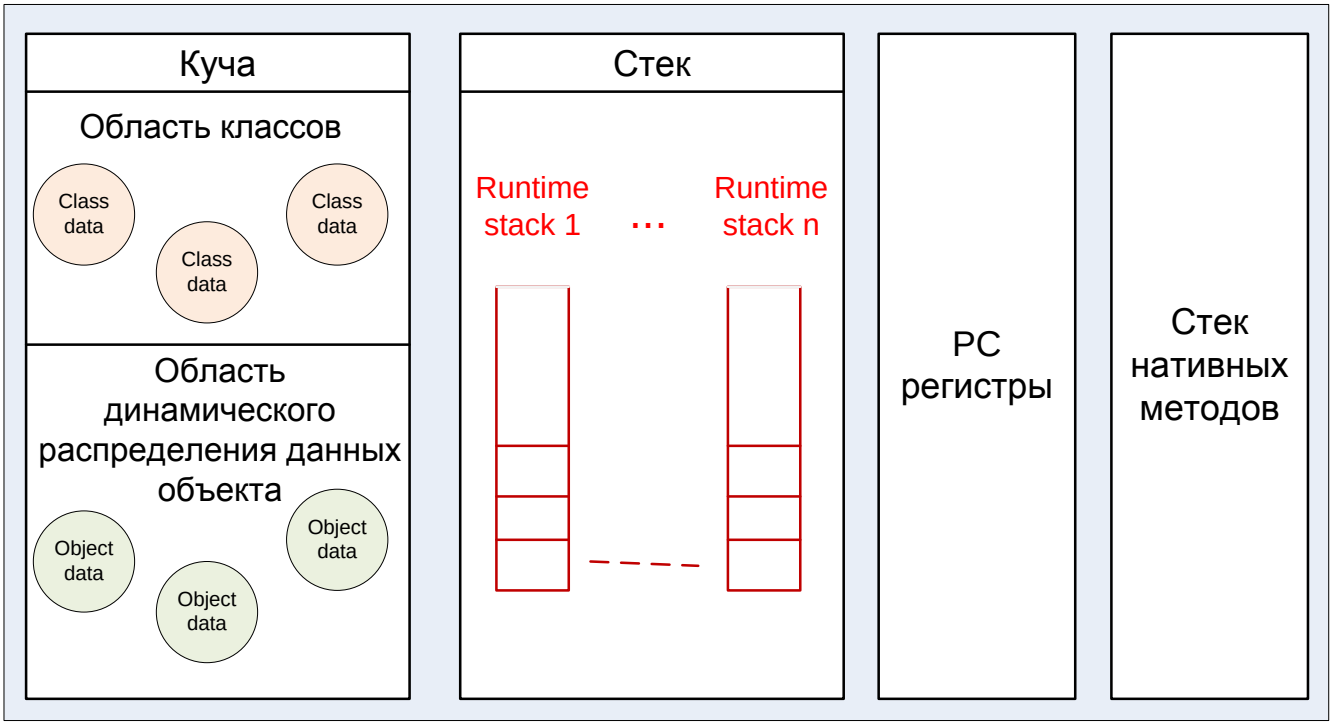
Ссылочные типы в JVM — ссылки на объекты — экземпляры классов, массивы и интерфейсы.

В JVM ссылка на объект является указателем на дескриптор объекта:



Все ссылки в JVM являются ссылками в плоском 32-разрядном адресном пространстве.

Основные области памяти, с которыми работает JVM:



Класс является основной программной единицей платформы Java, объединяющей в себе данные и методы их обработки.

Динамическая область памяти (куча) разделяется на две части: область классов и область динамически распределяемой памяти. Куча создается при запуске JVM.

Область классов:	
Область методов	Пул констант

Каждый класс представляется двумя структурами памяти: областью методов и пулом констант:

- область методов содержит исполняемую часть класса – байт-коды методов класса и таблицу символических ссылок на внешние методы и переменные;
- пул констант содержит литералы класса.

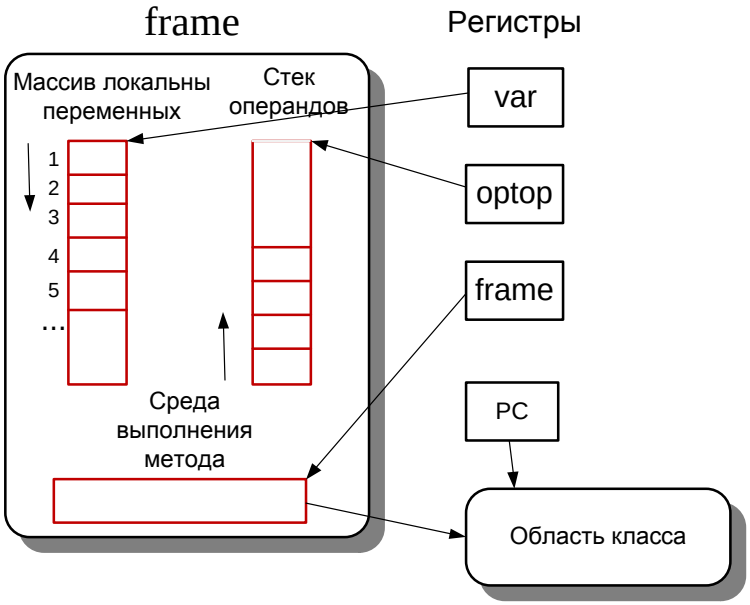
В области динамического распределения выделяется память для размещения объектов.

JVM поддерживает параллельное выполнение нескольких потоков. Для каждого потока JVM создает набор регистров и стек потока.

Набор регистров включает в себя четыре 32-разрядных регистра:

Регистр	**Назначение**
pc	регистр-указатель на команду
optop	регистр-указатель на вершину стека операндов текущего кадра
var	регистр-указатель на массив локальных переменных текущего кадра
frame	регистр-указатель на среду выполнения текущего метода

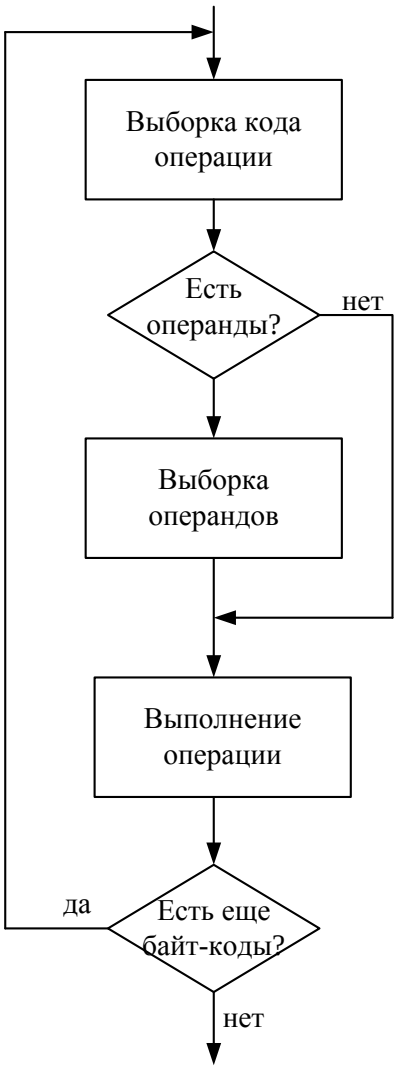
Элементами стека являются кадры (frame) методов:



JVM – стековая машина, которая выполняет байт-код в режиме интерпретации.

Команды JVM состоят из 1-байтного кода операции, могут содержать операнды. Число и размер операндов определяются кодом операции.

Основной алгоритм работы JVM:



Виртуальная машина Java – программа-интерпретатор.

Инструкция байт-кода обрабатывается, создается нативный код, который передается на исполнение процессору.

Основные типы команд JVM:

- ***команды*** загрузки и сохранения данных:
  - загрузка в стек локальной переменной;
  - сохранение значения из стека в локальной переменной;
  - загрузка в стек константы (из пула констант).
- ***команды*** манипулирования значениями:
  - арифметические операции;
  - побитовые логические операции;
  - сдвиг;
  - инкремент (операция работает с операндом - локальной переменной).
- ***команды*** преобразования типов;
- ***команды*** создания ссылочных данных и доступа к ним:
  - создания экземпляров класса;
  - доступа к полям класса;
  - создания массивов;
  - чтения в стек и сохранения элементов массивов;
  - получения свойств массивов и объектов.
- ***команды*** работы со стеком;
- ***команды*** передачи управления:
  - безусловный переход;
  - условный переход;
  - переход по множественному выбору.
- ***команды*** вызова методов и возврата;
- ***команды*** генерации и обработки исключений.

Каждый из типов данных JVM обрабатывается своими командами (используются префиксы и суффиксы, зависящие от типа данных)

Пример команд загрузки и сохранения целочисленных данных:

Команда	Opcode (hex)	Количество; операнды	Стек, до и после	Описание
iload	0x15	1: index	→ value	загрузить целое из локальной переменной с индексом *#index* в стек (проверки: тип int, диапазон)
istore	0x36	1: index	value →	сохранить целое в локальной переменной с индексом *#index* (проверки: тип int, диапазон)

Имеются аналогичные команды для загрузки

двойных целых	(0x16 - lload);
вещественных	(0x17 - fload) и т.д.;
загрузки ссылки на целочисленный объект	(0x19 - aload).

Спецификация JVM SE.

4.3.Структура файла описания класса

Каждый файл класса описывает один класс или интерфейс. Файл класса содержит поток байт, структурированный определенным образом.

Основные компоненты файла класса:

- ***верификационная информация***: «число» – сигнатура файла класса, номер версии;
- ***флаг доступа***, отображающий модификаторы, заданные в определении класса (public, final, abstract и т.д.), а также признак класса или интерфейса;
- ***пул констант*** – таблица структур, представляющих строковые константы — имена классов и интерфейсов, полей, методов, а также другие константы, на которые есть ссылки в файле класса;
- ***ссылки*** на имена this-класса и суперкласса в пуле констант;
- ***перечень интерфейсов***, реализуемых классом (в виде ссылок в пул констант);
- ***описание полей класса*** с указанием их имен, типов, модификаторов и т.д.;
- ***методы класса*** – каждый метод представляется в виде определенной структуры, в которой содержится описание метода (имя, модификаторы, и т.д.), одним из атрибутов этой структуры является массив байт-кодов метода.

Многие компоненты файла класса (пул констант, перечень интерфейсов и др.) имеют нефиксированную длину, такие компоненты начинаются 2-байтным полем, содержащим их длину.

4.4.Многопоточность и синхронизация

В Java средства создания потоков системно-независимые (поддерживаются встроенными средствами языка).

JVM обеспечивает для каждого потока собственную среду вычисления — собственный набор регистров и стек, выполняет действий по планированию потоков на выполнение. Для этого библиотечные методы Java обращаются к ОС, используя API той ОС, в среде которой работает JVM.

Эти средства позволяют синхронизировать работу потоков, в новых версиях был введен более стройный аппарат синхронизации и взаимного исключения.

Класс **Object** имеет три метода:

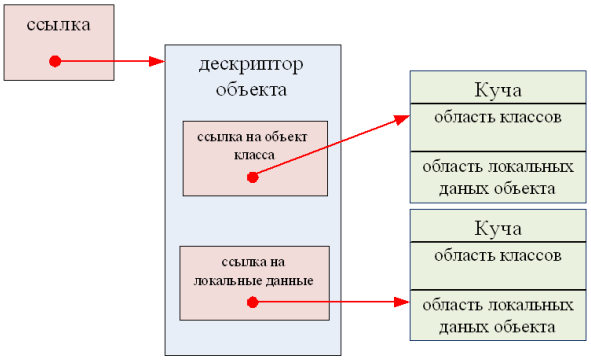
- wait() — ожидать уведомления об этом объекте;
- notify() — послать уведомление одному из потоков, ждущих уведомления об этом объекте;
- notifyAll() — послать уведомление всем из потоков, ждущим уведомления об этом объекте.

4.5. Управление памятью в куче

Память для данных ***примитивных типов*** выделяется в области локальных переменных кадра.

Стек для ***метода*** выделяется только на время выполнения метода, при завершении выполнения память стека освобождается, а, следовательно, все локальные переменные освобождаются.

Ссылочные типы состоят из двух частей: ***ссылки*** на объект и собственно ***тела объекта***.



Ссылка представляет собой указатель на объект (адрес памяти) в терминах языка C/C++, но адресная арифметика в Java не разрешена. Объект в программе доступен только через переменную, являющуюся ссылкой на него.

Память, выделяемая для ***ссылок***, управляется автоматически, как и память для примитивных типов.

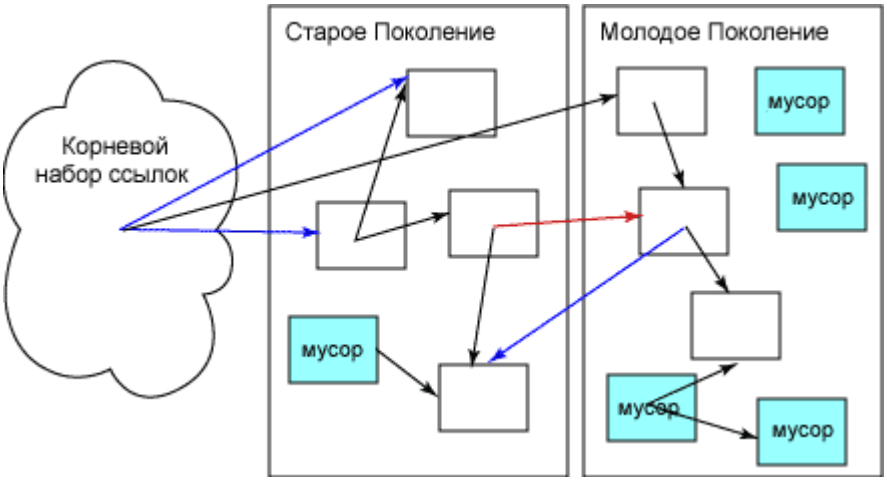
Выделение динамической памяти	инструкция **new** (явно): - выделяет память для тела объекта - возвращает ссылку на созданный объект
Освобождение динамической памяти	освобождает выделенную память **JVM**

Сборщик мусора автоматически освобождает память.

Сборщик мусора считает ***неиспользуемыми*** те объекты, на которые нет ссылок (т.к. память, занимаемая ссылкой на объект, освобождается при выходе из блока, в котором был создан объект).

Когда накопившийся мусор приводит к нехватке памяти, выполняется сборка мусора. При создании объекта устанавливается «признак мусора» (в дескрипторе каждого объекта имеется «признак мусора»).

Схема работы сборщика мусора (один из вариантов):



4.6.Защита ресурсов

Доступ к объектам в Java осуществляется по ссылкам.

Физический смысл ссылки и указателя C/C++ одинаков – это адрес памяти.

Ссылка в Java:

- ссылка не может быть преобразована в число или какое-либо иное представление физического адреса,
- над ссылками недопустимы арифметические операции (в C/C++ адресная арифметика может привести к выходу за пределы той области памяти, которая выделена под процесс).

Пример. ***Простейшее приложение на Java.***

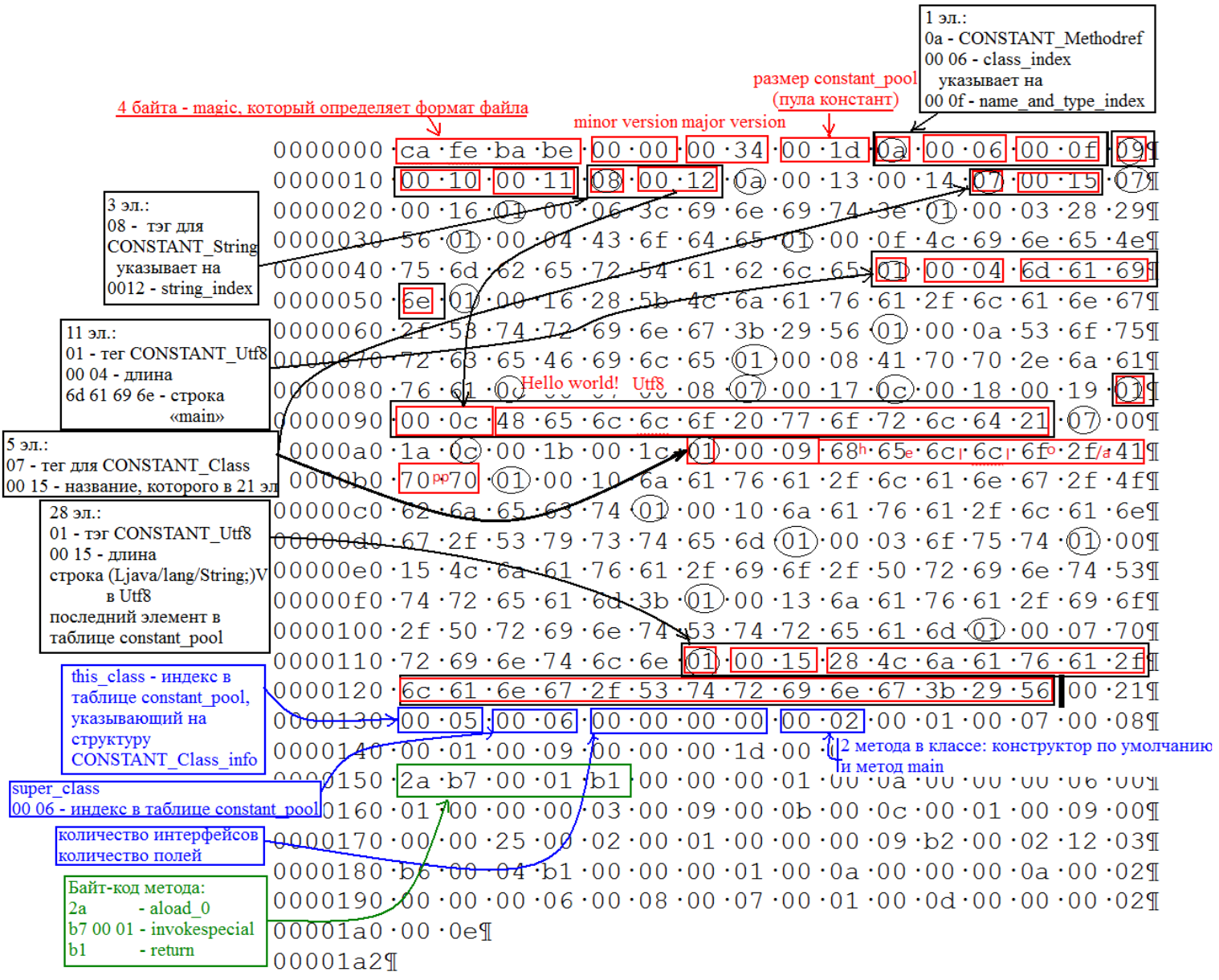
файл App.java:

```
package hello;

public class App {

    public static void main(String[] args) {
        System.out.println("Hello world!");
    }

}
```

Скомпилируем файл командой:

javac src/hello/App.java -d classes/

Содержимое файла App.class**:**



В классе 2 метода: конструктор по умолчанию и метод main:

`00 02` – methods_count

	Method 1	**Method 2**
Constructor/main	00 01 - access_flags 00 07 - name_index 00 08 - descriptor_index 00 01 - attributes_count	00 09 // access_flags 00 0b // name_index 00 0c // descriptor_index 00 01 // atributes_count
Attribute 1	00 09 - name_index (Code) 00 00 00 1d - attribute_length 00 01 - max_stack 00 01 - max_locals 00 00 00 05 - code_length	00 09 - name_index (Code) 00 00 00 25 - attribute_length 00 02 - max_stack 00 01 - max_locals 00 00 00 09 - code_length
code[code_length]	2a - aload_0 b7 00 01 - invokespecial b1 - return	b2 00 02 - getstatic 2 (java.lang.System) 12 03 - ldc 3 b6 00 04 - invokevirtual 4 b1 - return
Описание метода	00 00 - exception_table_length 00 01 - attributes_count 00 0a - attribute_name_index 00 00 00 06 - attribute_length 00 01 - line_number_table_length 00 00 - start_pc 00 03 - line_number	00 00 - exception_table_length 00 01 - attributes_count 00 0a - attribute_name_index 00 00 00 0a - attribute_length 00 02 - line_nuber_table_length 00 00 - start_pc 00 06 - line_number 00 08 - start_pc 00 07 - line_number

Далее следует описание атрибутов класса:

```
00 01       - attributes_count
00 0d       - name_index (SourceFile)
00 00 00 02 - attributes_length
00 0e       - sourcefile_index(App.java)
```