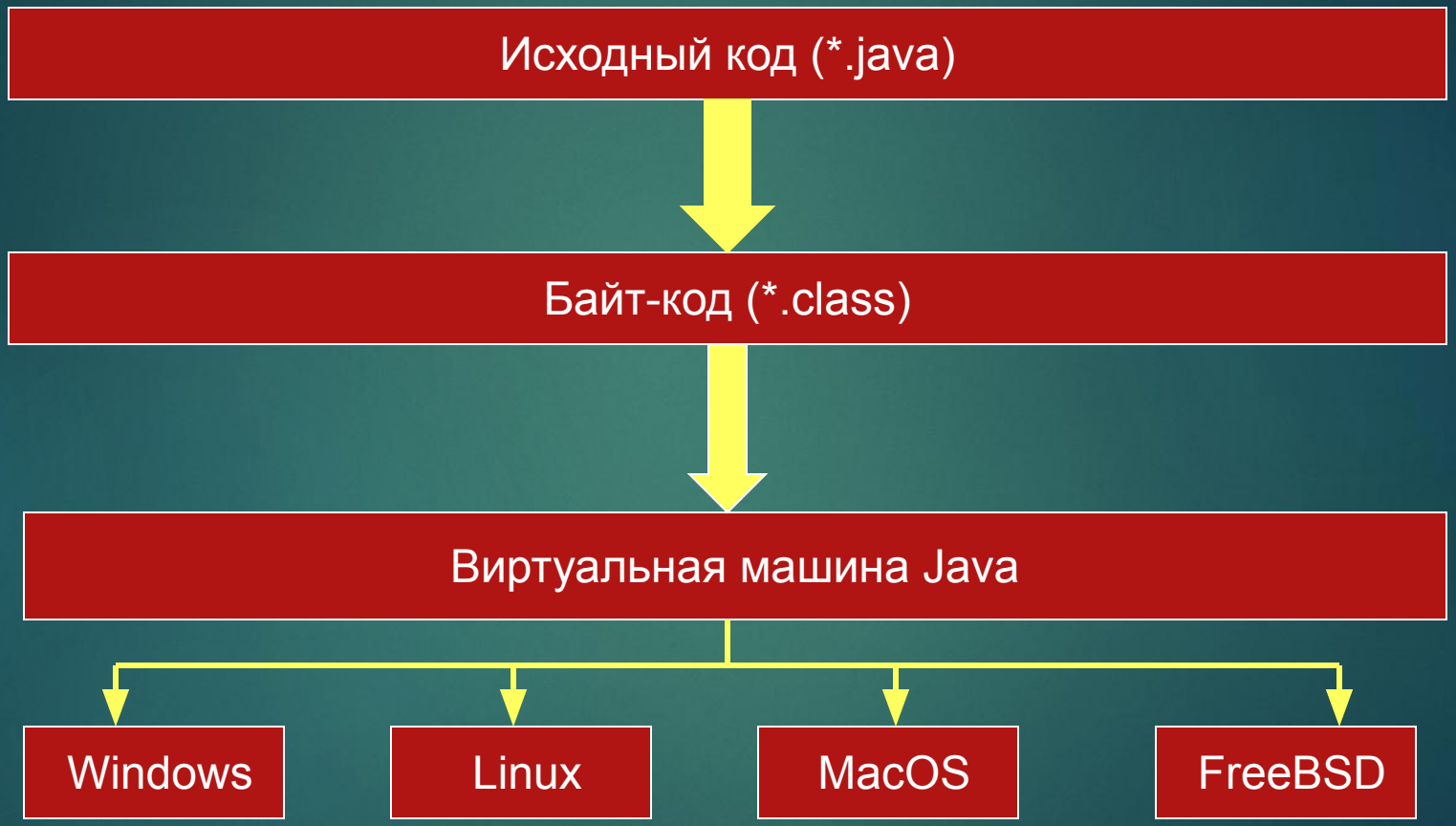
Характеристики языка Java

- ***Простой*** (нет указателей, нет освобождения памяти, нет перегрузки операций, нет шаблонов, нет множественного наследования).
- ***Объектно-ориентированный***, в Java даже нет глобальных переменных или функций, есть только поля и методы классов.
- ***Платформо-независимый*** т.е. не ориентирован на какую-то одну аппаратную или программную архитектуру.
- ***Устойчивый*** (проверяет выход за границу массива, не только предоставляет аппарат исключений, но и требует от программиста их обработки).
- ***Многопоточный***, средства работы с потоками встроены в язык.
- ***Интерпретируемый***, выполнение программы происходит путем интерпретации частично откомпилированного кода.
- ***Распределенный*** (позволяет выполнять удаленные вызовы методов).
- ***Динамический*** (использует информацию о типах и отражение).

Этапы программирования на языке Java



Соглашения об именовании

Имена классов должны всегда начинаться с большой буквы (например, **ConnectionFactory**)

Имена пакетов должны состоять только из букв в нижнем регистре и цифр. При этом каждая составляющая имени должна начинаться с буквы (например, **org.apache.log4j**). Обычно имена пакетов начинаются с инвертированного имени домена компании-разработчика. Так из приведенного выше примера видно, что разработчиком проекта log4j является Apache Software Foundation (www.apache.org).

Названия методов и переменных должны начинаться с маленькой буквы и быть осмысленными. Каждое новое слово должно начинаться с большой буквы. Подчеркивания отсутствуют. (например, **insertToDatabase()**, **value**, **personName**).

Названия констант состоят из больших букв, цифр и знаков подчеркивания в качестве разделителей между словами (**MAX_INTEGER**).

Классы, объекты и объектные ссылки

Класс в Java – это некоторое описание типа

Объект представляет собой экземпляр класса

Доступ к объектам и вызов их методов осуществляется посредством объектных ссылок.

Ссылка может не ссылаться ни на какой объект — тогда это пустая (**null**) ссылка.

Все ссылки строго типизированы

Данные простых типов ссылками не являются

Примитивные типы java

Целые типы переменных

Тип	Размер (бит) битах	Минимальное значение	Максимальное значение
`byte`	8	-128	127
`short`	16	-32768	32767
`int`	32	-2147483648	2147483647
`long`	64	-9223720368547 75808	9223720368547 75807
`char`	16	0	65536

```
int x = 0;
long i = 122737;
byte a1 = 12;
int a2 = 0x07;
short r1 = 017;
char ch = ‘w’;
```

Примитивные типы java

Вещественные типы переменных

Тип	Разрядность (бит)	Диапазон	Точность
`float`	32	`3,4e-38 < |x| < 3,4e38`	7-8 цифр
`double`	64	`1,7e-308 < |x| < 1,7e308`	17 цифр

```
double b1 = 4.12;
float pi = 3.14f;
//При использование типа float требуется указывать f в конце числа
double d = 27;
double c = pi *d;
```

Примитивные типы java

Булевский тип переменных

Переменные булевского типа (логические переменные) могут принимать одно из двух значений: «истина» или «ложь» и используются в языках программирования в операциях отношения (сравнения) и логических операциях. Так, результатом сравнения

5 > 3

будет «истина», а результатом сравнения

8 < 1

будет «ложь».

```
boolean switch = true;
```

Значения NaN и Infinity

- **Inifinity** – бесконечность
  - Может получаться, например, в результате деления на ноль.
  - Может быть как положительным, так и отрицательным (-Infinity).
  - Константы определены в классах Double и Float
    - Double.NEGATIVE_INFINITY
    - Double.POSITIVE_INFINITY
- **NaN** (Not-a-Number) –значение, используемое для представления результата некоторой некорректной операции, такой как деление 0 на 0.
  - Константы определены в классах Double и Float
    - Double.NaN

```
System.out.println(1d/0d); // вернет Infinity
System.out.println(0d/0d); // вернет NaN
```

Приведение типов

- Ручное приведение типов – с использованием конструкции (<тип>)
  - byte b = (byte)intVal;
- Автоматическое преобразование типов производится виртуальной машиной при выполнении математических операций:
  - над переменными разного типа – результат приводится к старшему типу.
  - когда результирующее значение должно получиться большего размера – к типу, старшему чем тип операндов.
  - Например, выражение

    byte b = 10;

    b = b*(**byte**)2;

    Не будет работать, т.к. необходимо ручное преобразование к byte
- При делении целых чисел друг на друга преобразование к double или float не производится – и дробная часть отбрасывается. Поэтому, если она необходима – нужно преобразовать один и операндов в double вручную
  - **double** d = 3 / 2; // d будет иметь значение 1.0 !!!
  - d = (**double**)3 / 2; // d будет иметь значение 1.5

Комментарии

Java допускает комментарии в исходном коде программы:

многострочные в стиле языка C:

```
/* Любое количество любых строк
   лишь бы там не было сочетания
   звездочки и косой черты  */
```

однострочные в стиле языка C++:

```
// все написанное до конца строки - комментарий
```

комментарии документатора :

```
/** Многострочный комментарий,
 * который войдет в
 * программный документ */
```

Арифметические операторы

+ Сложение
- Вычитание
* Умножение
/ Деление
% Вычисление остатка
++ Инкремент
-- Декремент
+= Присваивание со сложением
-= Присваивание с вычитанием
*= Присваивание с умножением
/= Присваивание с делением
%= Присваивание с вычислением остатка

Логические операторы

Оператор Описание
= = Равно
!= Не равно
< Меньше
< = Меньше или равно
> Больше
> = Больше или равно
& Логическое и
| Логическое или
! Отрицание
&& Условное и
| | Условное или

Управляющие конструкции

Отметим, что управляющие конструкции Java схожи с операторами языка C, но есть некоторые отличия.

Оператор if / else

Схема условного оператора такова:

```
if (условие)
   оператор 1
[else
   оператор 2]
```

Смысл условного оператора такой же как и в других языках программирования. В отличие от языка C условие всегда должно иметь логическое значение, поэтому пишут

```
if (i == 0) …
```

а не

```
if (!i) …
```

как в C

Операторы цикла while и do / while

Схемы для оператора цикла следующие:

```
while (условие)
    оператор;
```

и

```
do
   оператор
while (условие);
```

Условием служит такое же логическое выражение, как в операторе **if**

В первом цикле условие проверяется до выполнения вложенного оператора (цикл с предусловием), во втором - после (цикл с постусловием). Это ведет к тому, что вложенный оператор первого цикла может не выполниться ни разу, а оператор второго цикла обязательно будет выполнен хотя бы раз

Примеры циклов:

```
// Цикл с предусловием
float sum = 0, x = 1;
while(x < 100) {
    sum += 1 / x;
    x++;
}
// То же, но с постусловием
sum = 0; x = 1;
do {
    sum += 1 / x;
    x++;
while(x < 100)
// Идиома бесконечного цикла
while (true) {
    // оператор
}
```

Оператор for

Оператор for имитирует такой же оператор языка C.

for (выражение_инициализации; [выражение1] ; [выражение2])
оператор

Смысл оператора for передается следующим псевдокодом, который делает то же, что оператор for.

```
выражение_инициализации;
while (выражение1) {
    оператор
    выражение 2;
}
```

Операторы break и continue

Чтобы прервать последовательность повторений любого цикла в языке Java используется оператор break

Пример. Цикл прекратит повторяться, когда x превысит 99.

⇓

```
for (float x = 1; ; x++ ) {
    if (x > 99)
        break;
    sum += 1 / x;
}
```

Чтобы перейти к следующей итерации цикла, не завершая текущую, в языке C используется оператор continue

Пример. Второй десяток слагаемых в сумму не войдет.

⇓

```
for (float x = 1; x < 100; x++ ) {
    if ( 10 < x && x < 21)
        continue;
    sum += 1 / x;
}
```

Оба эти оператора есть в языке Java, но, кроме того, для работы с вложенными циклами имеется вариант этих операторов с меткой. Метка L помечает блок кода, внутри которого располагаются операторы "break L;" или "continue L;"

Оператор "break L;" - передает управление оператору, следующему за помеченным блоком

Оператор "continue L;" - продолжает помеченный цикл (инструкция continue используется только в циклах);

Пример. Применение оператора break для досрочного выхода из вложенных циклов.

⇓

```
for (int i1 = 0; i1 < 10; i1++) {
L: for (int i2 = 0; i2 < 10; i2++) {
      for (int i3 = 0; i3 < 10; i3++) {
         if (i3 == 6)
            break L;
      }
   }
   // Сюда попадет управление после выполнения оператора "break L;"
}
```

Оператор return

Обеспечивает возврат из вызываемой функции в вызывающую