Собеседование по Java — типы данных, переменные, операторы, циклы, массивы

1. Сколько ключевых слов зарезервировано языком, что это за слова, какие из них не используются?
2. Из каких символов может состоять имя переменной (корректный идентификатор)?
3. Что значит слово “инициализация”?
4. На какие основные группы можно поделить типы данных?
5. Какие примитивные типы вы знаете?
6. Что вы знаете о преобразовании примитивных типов данных, есть ли потеря данных, можно ли преобразовать логический тип?
7. Какими значениями инициализируются переменные по умолчанию?
8. Как передается значение переменной (по ссылке/значению)?
9. Что вы знаете о функции main, какие обязательные условия ее определения?
10. Какие логические операции и операторы вы знаете?
11. В чем разница краткой и полной схемы записи логических операторов?
12. Что такое таблица истинности?
13. Что такое тернарный оператор выбора?
14. Какие унарные и бинарные арифметические операции вы знаете?
15. Какие побитовые операции вы знаете?
16. Какова роль и правила написания оператора выбора (switch)?
17. Какие циклы вы знаете, в чем их отличия?
18. Что такое “итерация цикла”?
19. Какие параметры имеет цикл for, можно ли их не задать?
20. Какой оператор используется для немедленной остановки цикла?
21. Какой оператор используется для перехода к следующей итерации цикла?
22. Что такое массив?
23. Какие виды массивов вы знаете?
24. Что вы знаете о классах оболочках?
25. Что такое автоупаковка (boxing/unboxing)?

#### **1. Сколько ключевых слов зарезервировано языком, что это за слова, какие из них не используются?**

50, два из них не используются: const, goto;

Для запоминания это:

* Примитивы (byte, short, int, long, char, float, double, boolean)
* Циклы и ветвления (if, else, switch, case, default, while, do, break, continue, for)
* Исключения (try, catch, finally, throw, throws)
* Области видимости (private, protected, public)
* Объявление \ Импорт (import, package, class, interface, extends, implements, static, final, void, abstract, native)
* Создание \ Возврат \ Вызов (new, return, this, super)
* Многопоточность (synchronized, volatile)
* instanceof, enum, assert, transient, strictfp, const, goto

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| abstract | continue | for | new | switch |
| assert\*\*\* | default | goto\* | package | synchronized |
| boolean | do | if | private | this |
| break | double | implements | protected | throw |
| byte | else | import | public | throws |
| case | enum\*\*\*\* | instanceof | return | transient |
| catch | extends | int | short | try |
| char | final | interface | static | void |
| class | finally | long | strictfp\*\* | volatile |
| const\* | float | native | super | while |

\* not used; \*\*added in 1.2, \*\*\* added in 1.4, \*\*\*\* added in 5.0.

##### http://docs.oracle.com/javase/tutorial/java/nutsandbolts/\_keywords.html

#### **2. Из каких символов может состоять имя переменной (корректный идентификатор)?**

Имя или идентификатор переменной — это последовательность из строчных и заглавных латинских букв, цифр, а также символов «$» и «\_». ***ТЕХНИЧЕСКИ****, имя переменной****может начинаться с любого из перечисленных символов,* *кроме цифры***. Но «$» или «\_» - запрещено использовать соглашением по оформлению кода в Java (Java Code Conventions). Кроме того, символ доллара «$», по соглашению, никогда не используется вообще. В соответствии с соглашением имя переменной должно начинаться именно с маленькой буквы (с заглавной буквы начинаются имена классов). **Пробелы** - **не допускаются**.

Желательно: camelCase, без цифр и спец.символов !!!!!!

#### **3. Что значит слово “инициализация”?**

**Инициализация** (от англ. *initialization*, инициирование) для объекта - создание, активация, подготовка к работе, определение параметров. Приведение программы или устройства в состояние готовности к использованию. С точки зрения Java — выделение памяти под объект.

Например, при создании *MyClass myClass = new MyClass().* Таким образом будет выделена память под объект *myClass* (он будет инициализирован).

Без инициализации (*new MyClass()*) запись *MyClass myClass*; просто резервирует имя (объявляется переменная *myClass* типа *MyClass*).

#### **4. На какие основные группы можно поделить типы данных?**

#### Ссылочные и примитивные.

#### **5. Какие примитивные типы вы знаете?**

***Примитивные***

* byte (целые числа, 1 байт, [-128, 127])
* short (целые числа, 2 байта, [-32768, 32767])
* int (целые числа, 4 байта, [-2147483648, 2147483647])
* long (целые числа, 8 байт, [-9223372036854775808,9223372036854775807])
* float (вещественные числа, 4 байта)
* double (вещественные числа, 8 байт)
* char (символ Unicode, 2 байта, [0, 65536])
* boolean (значение истина/ложь, используется int, зависит от JVM)

***Ссылочные***. В ссылочные типы входят все классы, интерфейсы, массивы.

##### http://stackoverflow.com/questions/383551/what-is-the-size-of-a-boolean-variable-in-java boolean type: http://docs.oracle.com/javase/specs/jvms/se8/html/jvms-2.html#jvms-2.3.4 boolean — боремся за Java память… https://habrahabr.ru/post/76481/

#### **6. Что вы знаете о преобразовании примитивных типов данных, есть ли потеря данных, можно ли преобразовать логический тип?**

Преобразование может быть неявным и явным (приведение типов).

**Неявное** преобразование может выполняться если:

1. типы совместимы (например — оба целочисленные)
2. размер «принимающего» типа больше чем у того, который преобразуется (так называемое «преобразование с расширением»)

|  |  |
| --- | --- |
| 1  2 | int a = 123454;  double b =  a; //неявное преобразование - преобразование с расширением |

**Явное** преобразование имеет вид **переменная\_нового\_типа = (новый\_тип) имя переменной;**

|  |  |
| --- | --- |
| 1  2 | int a;  byte b = (byte) a; //b будет остатком от деления a на диапазон byte, может быть потеря данных |

Примеры:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | public static void typeConverterExample() {          long a = 100L;          double b = 300.0;          Object ab = a + b;          System.out.println(ab.getClass().getName() + " value: " + ab); //java.lang.Double value: 400.0            double c = 1000.05;          long d = 1000;          Object cd = c+d;          System.out.println(cd.getClass().getName() +" value: " + cd);//java.lang.Double value: 2000.05      }        public static void typeNarrowing() {          int a0 = 64;          int a = 257;          int a2 = 126;          int a3 = 129;          byte b0 = (byte) a0;          byte b = (byte) a;          byte b2 = (byte) a2;          byte b3 = (byte) a3;          System.out.println(b0+ " " + b + " " + b2 + " " + b3); //64 1 126 -127            double c = 56.9876;          int d = (int) c;          System.out.println(d); //56            long e = 1000L;          float f = (float) e;          System.out.println(f); //1000.0      } |

При повышении типа byte>short; short>int; int>long; float>double; char>int информация не потеряется. При сужении возможна потеря информации (см. пример выше byte = (byte) int).

При различных операциях может происходить повышение типов в порядке «усиления» к более информативному типу. Например складывая int и double получим тип double. Но есть и особенность, например сложив double (4 байта) и long (8 байт) Java оставит знаки после запятой (float), а не более «длинный» тип. Аналогичный пример с вещественной частью:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | long a = 100L;          double b = a;          Object ab = a + b;          System.out.println(ab.getClass().getName() + " value: " + ab); //java.lang.Double value: 200.0            float c = 100;          long d = 1000;          Object cd = c - d;          System.out.println(cd.getClass().getName() +" value: " + cd);//java.lang.Float value: -900.0 |

Кратко можно записать такие правила:

1. byte, short, char в выражениях всегда повышаются до int
2. если в выражении участвует тип long — то именно к этому типу будет приведён результат
3. если в выражении участвует float — то результат приводится к float
4. если один из операндов имеет тип double — то к этому типу будет приведён весь результат
5. При выборе между длиной и возможностью сохранить дробную часть — будет выбрана дробная часть
6. При выполнении арифм.операций с примитивами в цикле, возможна ситуация переполнения типа(макс.или мин.!!). В таком случае, примитив получит МАКС. или МИН. значение, обратное(с противоположным знаком) текущему.

* ПРИМЕР: byte (целые числа, 1 байт, [-128, 127]) .

*127 + 1 (переполнение типа!), переменная получает значение -128 !!!! Если на этом примитиве завязано условие выхода из цикла, то возможен бесконечный цикл!!!*

#### **7. Какими значениями инициализируются переменные по умолчанию?**

Числа инициализируются 0 или 0.0. Объекты (в том числе String) — null, char — \u0000; boolean — false;

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | class TestClass {              int a; double b; float c; char d; String s; Object o; boolean e;    *@Override*              public String toString() {                  return "TestClass{" +                          "a=" + a + ", b=" + b + ", c=" + c + ", d=" + d + ", s='" + s + '\'' + ", o=" + o +", e=" + e +'}';              }          }          TestClass testClass = new TestClass();          System.out.println(testClass); //TestClass{a=0, b=0.0, c=0.0, d= , s='null', o=null, e=false}      } |

*Тип char – целочисленный. Если, например,* char d = 65, *то переменная* d получит символьное значение по данному целочисленному коду (по таблице ASCII это значение «А» ).

0 – по умолчанию int.

0.0 – по умолчанию double.

#### **8. Как передается значение переменной (по ссылке/значению)?**

Постоянный холивар по этому вопросу, видимо, связан с переводом на русский язык некоторых терминов, а так же работой Java с примитивами, строками и ссылочными объектами. По английски всё звучит строго — **Java is always pass-by-value**. Обычно спор вызывают особенности языка относящиеся к пониманию примитивов, ссылок и объектов в Java (и то, что некоторые из них неизменяемые). Итак:

* Java передает всё **по значению**. Java никогда не передает ничего по ссылке. Примитивы, ссылки, null — всё передается по значению, не по «ссылке».
* **Передача по значению**: Когда передается параметр в метод, то параметр копируется в другую переменную и она передается в метод. Поэтому это и называется «передача по значению».
* **Передача по ссылке**: Если мы передаем в метод ссылочный тип (объект), то так же происходит передача копии этого объекта (ссылки), которая указывает на какую-либо область памяти (можно назвать ссылку — «указателем»). Изменив что-то в области памяти, вы меняете все объекты, которые ссылаются на эту область памяти. Отсюда идет название **«передача по ссылке»**, хотя передается всё то же значение, просто значением является указатель на область памяти.

Приведу пример с холивара на stackoverflow (http://stackoverflow.com/questions/40480/is-java-pass-by-reference-or-pass-by-value) и переведу на русский:

Пусть мы имеем такую строчку, которая резервирует имя myDog под тип Dog.

|  |  |
| --- | --- |
| 1 | Dog myDog; |

Но это еще не физический объект, а только указатель на тип Dog. Теперь создадим (инициализируем) этот объект.

|  |  |
| --- | --- |
| 1  2 | Dog myDog = new Dog("Rover");  foo(myDog); |

Теперь myDog действительно занимает область в памяти (пусть будет область Dog@42). Вызвав метод foo(myDog) мы передаем **копию** значения указателя на область памяти 42.

|  |  |
| --- | --- |
| 1  2  3  4  5 | public void foo(Dog someDog) {      someDog.setName("Max");     // AAA      someDog = new Dog("Fifi");  // BBB      someDog.setName("Rowlf");   // CCC  } |

В строчке //AAA мы поменяли значение в памяти 42 на «Max».  
В строчке //BBB мы создали **новый** объект, который теперь будет лежать в другой области памяти (пусть будет Dog@74).  
В строке //CCC мы поменяли значение в области памяти 74. Очевидно, что изначальный объект myDog@42 ничего об этом не знает, но он ссылается на уже измененную в строке //AAA область памяти и теперь будет возвращать имя «Max».

Теперь другая засада — примитивы и строки.

* Когда передается примитив в метод, то передается также его копия (копия значения).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | int x = 5;  int y = changeX(x)  //x=5;  //y=25;    private int changeX(int value) {  value = value \* 5;  return value;  } |

Создали примитив x = 5. Передали в метод **копию** значения (т.е. 5). В методе умножили на число и вернули его. Значение примитива y будет равно 25, а значение примитива x = 5 (т.е. не изменится). Всё дело в том, что мы передали **значение** (5), а **не указатель** на область памяти. Соответственно значение x и не должно было поменяться, т.к. мы его не меняли (в методе использовалась копия значения).

* Строки immutable, т.е. неизменяемые. Когда мы присваиваем строке новое значение, то всегда будет создан новый объект в памяти.

Еще один пример, что происходит при передаче параметра в метод:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | class TestClass {              public ArrayList<Object> changeObjectValue(ArrayList<Object> objectValue) {                  objectValue.clear();                  objectValue.add(999);                  return objectValue;              }                public String changeStr(String str) {                  str = "NewString";                  return str;              }                public int changeX(int x) {                  x = x\*5;                  return x;              }          }            TestClass testClass = new TestClass();            ArrayList<Object> value = new ArrayList<Object>();          value.add(23);          String str = "FirstString";          int x = 2;          System.out.println(value + " " + str + " " + x); //[23] FirstString 2            ArrayList<Object> value2 = new ArrayList<Object>();          value2 = value;          value2 = testClass.changeObjectValue(value2);            String str2 = str;          str2 = testClass.changeStr(str2);            int x2 = testClass.changeX(x);            System.out.println(value + " " + str + " " + x); //[999] FirstString 2          System.out.println(value2 + " " + str2 + " " + x2); //[999] NewString 10          System.out.println(value.equals(value2) + " " + str.equals(str2) + " " + (x2 == x)); //true false false |

1. Создаем ссылочный объект value и добавляем в него объект (Integer=23). Так же создаем строку str = «FirstString» и примитивное число x= 2.
2. Отдельно создаем объект value2 и выделяем под него память; Сейчас это выглядит что-то вроде value={ArrayList@450}, value2={ArrayList@453}, т.е. видно, что это разные ссылки на объекты (разные области памяти).
3. value2={ArrayList@450}, value={ArrayList@450} после присвоения оба объекта ссылаются на одну область памяти. Теперь попытаемся изменить их с помощью методов в тестовом классе

* Объекты передаются по копии указателя на область памяти. Т.к. value и value2 указывают на один объект в памяти, то после изменения в методе changeObjectValue одного объекта, поменяется и второй.
* Строка (String) всегда создается новая. Хоть str2 = str и указывали на одну область вначале, но после изменения str2 на новое значение NewString, значение первой строки осталось прежним.
* Т.к. примитивные типы передаются по копии значения (в данном случае 2), то изменение в методе примитивного типа никак не отразилось на значении исходной переменной x (значение в области памяти для x осталось прежним).

Если всё же остались вопросы, то гуглите по **Java pass-by-value.**

#### Переменная всегда передаётся по значению(для примитивных – это непосредственно значение, для ссылочных - ссылка).

#### **9. Что вы знаете о функции main, какие обязательные условия ее определения?**

Обязательная запись  *public static void main(String[] args) {/\*тело метода\*/ }*

или же *public static void main(String… args) {/\*тело метода\*/ }*

Метод main() — точка входа в программу. Может быть несколько методов main. Входные параметры — только массив строк(либо varArg – массив переменной длинны). Если этого метода не будет, то компиляция возможна, но при запуске не в среде разработки(например Idea) будет Error: Main method not found.

#### **10. Какие логические операции и операторы вы знаете? 11. В чем разница краткой и полной схемы записи логических операторов?**

Логические операции: >, <, <=, >=, ==, !=;

|  |  |
| --- | --- |
| Оператор | Описание |
| & | Логическое AND (И) |
| && | Сокращённое AND |
| | | Логическое OR (ИЛИ) |
| || | Сокращённое OR |
| ^ | Логическое XOR (исключающее OR (ИЛИ)) |
| ! | Логическое унарное NOT (НЕ) |
| &= | AND с присваиванием |
| |= | OR с присваиванием |
| ^= | XOR с присваиванием |
| == | Равно |
| != | Не равно |
| ?: | Тернарный (троичный) условный оператор |

Если оба операнда имеют значение true, тогда операторы & и && возвращают true.

Если хотя бы один операнд имеет значение true, тогда операторы | и || возвращают true.

Операторы & и | всегда проверяют значение обоих операндов. && и || носят название операторов короткой схемы, так как если результат булевого выражения может быть определён из левого операнда, правый операнд не вычисляется.

Примечание: || и && могут быть использованы только в логических выражениях.

#### **12. Что такое таблица истинности?**

Таблица истинности — это таблица, описывающая логическую функцию.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | B | A | B | A & B | A ^ B | !A |
| false | false | false | false | false | true |
| true | false | true | false | true | false |
| false | true | true | false | true | true |
| true | true | true | true | false | false |

#### **13. Что такое тернарный оператор выбора?**

В языке Java есть также специальный тернарный условный оператор, которым можно заменить определённые типы операторов if-then-else — это оператор ? :

Тернарный оператор использует три операнда. Выражение записывается в следующей форме:  
логическоеУсловие ? выражение1 : выражение2  
Если логическоеУсловие равно true, то вычисляется выражение1 и его результат становится результатом выполнения всего оператора. Если же логическоеУсловие равно false, то вычисляется выражение2, и его значение становится результатом работы оператора. Оба операнда выражение1 и выражение2 должны возвращать значение одинакового (или совместимого) типа.

#### **14. Какие унарные и бинарные арифметические операции вы знаете?**

**Унарные** операции выполняются над одним операндом, **бинарные** — над двумя операндами.

Операндом является переменная или значение (например, число), участвующее в операции.

Пример *унарных* арифметических операций:

* *++* — постфиксный/префиксный инкремент, увеличивает значение целочисленной переменной на 1;
* *—* (двойной минус) — постфиксный/префиксный декремент, уменьшает значение целочисленной переменной на 1;
* *+* — оставляет знак числа;
* *—* — изменяет знак числа.

Слово постфиксный означает, что операция применится к операнду после вычисления всего выражения, в которое операнд входит. Аналогично префиксный означает, что операция применится до вычисления выражения.

Пример *бинарных* арифметических операций:

* *+* — сложение чисел или строк;
* *—*  — вычитание чисел;
* *\** — умножения чисел;
* */* — деления чисел;
* *%* — вычисление остатка от деления чисел.

Операция вычисления остатка от деления применима как к целым числам, так и к вещественным.

#### **15. Какие побитовые операции вы знаете?**

|  |  |
| --- | --- |
| ~ | Побитовый унарный оператор NOT |
| & | Побитовый AND |
| &= | Побитовый AND с присваиванием |
| | | Побитовый OR |
| |= | Побитовый OR с присваиванием |
| ^ | Побитовый исключающее XOR |
| ^= | Побитовый исключающее XOR с присваиванием |
| >> | Сдвиг вправо (деление на 2 в степени сдвига) |
| >>= | Сдвиг вправо с присваиванием |
| >>> | Сдвиг вправо с заполнением нулями |
| << | Сдвиг влево (умножение на 2 в степени сдвига) |
| <<= | Сдвиг влево с присваиванием |
| >>>= | Сдвиг вправо с заполнением нулями с присваиванием |

##### **Подробнее http://developer.alexanderklimov.ru/android/java/bitwise.php**

#### **16. Какова роль и правила написания оператора выбора (switch)?**

Оператор **switch** сравнивает аргумент на равенство с предложенным значением.

**switch** – расширение if(когда в if более 3х условий).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | switch(ВыражениеДляСравнения) {       case Совпадение1:      команда;  break;  case Совпадение2:      команда;  break;  case Совпадение3:      команда;  break;  default:      оператор;      break;  } |

Обязательно необходимо ставить break; после завершения тело команды, иначе будет продолжаться выполнение ниже по строчкам.

Если убрать break, то одна и та же операция будет применяться к любому из кейсов.

##### Подробнее http://developer.alexanderklimov.ru/android/java/switch.php

#### **17. Какие циклы вы знаете, в чем их отличия?**

В Java используются циклы for, foreach, while, do-while.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | while(условие) {      // тело цикла  }    do {      // тело цикла  }  while(условие-логическое выражение)    for(инициализация; логическое выражение (условие выхода); шаг (итерация)) {      // тело цикла  }    for(Object object : objects) {      // тело цикла  }  foreach – построен на основе Итератора (интерфейс Iterable). В данном примере: objects.iterator(). Чтобы зайти в foreach , у Итератора вызывается hasNext(). Если true, тогда object = Iterator.next(). Если false – не входит в цикл. |

do-while всегда выполнится хотя бы один раз, т.к. проверка идет после тела цикла.

#### **18. Что такое “итерация цикла”?**

Это одноразовое выполнение кода, размещенного в теле цикла.

#### **19. Какие параметры имеет цикл for, можно ли их не задать?**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | for(инициализация; логическое выражение (условие ); шаг (итерация)) {      // тело цикла  }  // можно записать вот так  for(;;) {  }  //такая запись эквивалентна while(true) {} |

#### **20. Какой оператор используется для немедленной остановки цикла?**

break; — выход из цикла (не затрагивает внешний цикл).**21. Какой оператор используется для перехода к следующей итерации цикла?**

continue; — заканчивает выполнение кода в этом теле цикла. Переход к следующей итерации.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | for (int i=0; i < 3; i++) {              for (int j = 0; j < 4; j++) {                  if (j == 2) {                      System.out.println("#A# i: " + i + " j: " + j + " break (end j loop). Will not see #C#");                      break;                  }                  if (j == 1) {                      System.out.println("#B# i: " + i + " j: " + j + " continue (j++).Will not see #C#");                      continue;                  }                  System.out.println("#C# i: " + i + " j: " + j);              }          }  #C# i: 0 j: 0  #B# i: 0 j: 1 continue (j++).Will not see #C#  #A# i: 0 j: 2 break (end j loop). Will not see #C#  #C# i: 1 j: 0  #B# i: 1 j: 1 continue (j++).Will not see #C#  #A# i: 1 j: 2 break (end j loop). Will not see #C#  #C# i: 2 j: 0  #B# i: 2 j: 1 continue (j++).Will not see #C#  #A# i: 2 j: 2 break (end j loop). Will not see #C# |

#### **22. Что такое массив?**

Массив (в некоторых языках программирования также таблица, ряд, матрица) — структура данных в виде набора компонентов одного типа (элементов массива), расположенных в памяти непосредственно друг за другом. При этом доступ к отдельным элементам массива осуществляется с помощью индексации, то есть ссылки на массив с указанием номера (индекса) нужного элемента. За счёт этого, массив является структурой с произвольным доступом.

Размерность массива — т.к. массив может содержать другие массивы, для доступа к конкретному элементу необходимо указать индексы на каждом из уровней => количество указанных индексов = размерности массива. Например:

int[][][] ints = new int[][][] {}; <=> int[][][] ints = new int[5][5][5] ;

int i = ints[0][0][0];

В Java массивы являются объектами. Это значит, что имя, которое даётся каждому массиву, лишь указывает на адрес какого-то фрагмента данных в памяти*. Кроме адреса в этой переменной ничего не хранится. Индекс массива, фактически, указывает на то, насколько надо отступить от начального элемента массива в памяти, чтоб добраться до нужного элемента.*

|  |  |
| --- | --- |
| 1  2  3  4  5 | тип[] имя;  тип  имя[];    тип[] имя  = new тип[размер];  тип[] имя = {эл0, эл1, …, элN}; |

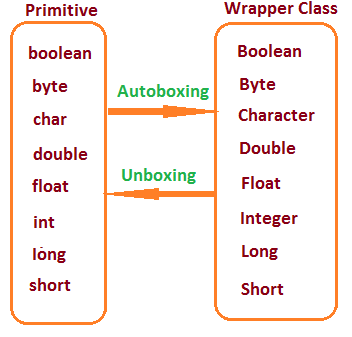
#### **23. Какие виды массивов вы знаете?**

В Java — одномерные и многомерные массивы.

#### **24. Что вы знаете о классах-оболочках?**

Для каждого примитивного типа есть соответствующий класс (Byte, Double, Float, Integer, Long, Short). Числовые классы имеют общего предка — абстрактный класс Number, в котором описаны шесть методов, возвращающих числовое значение, содержащееся в классе, приведенное к соответствующему примитивному типу: byteValue(), doubleValue(), floatValue(), intValue(), longValue(), shortValue(). Эти методы переопределены в каждом из шести числовых классов-оболочек.

Классы-оболочки реализуют Comparable => имеют естественный порядок сортировки.



#### **Что такое автоупаковка/автораспаковка (boxing/unboxing)?**

Это автоматическое преобразование из примитивных типов данных к ссылочным и наоборот.