1. Язык Java. Особенности языка.

Есть какой-то язык, называемый A (язык программирования), на котором программист пишет код, те человек его понимает. Также есть язык B (машинный код), который понимает компьютер, причем языки A и B не имеют между собой ничего общего, те компьютер не понимает язык A, а человеку неудобно писать на языке B.
Компиляция: каждая строчка кода на языке A заменяется эквивалентной на языке B и комп как раз выполняет новую программу уже переведенную на язык B. Описанная технология - трансляция. Определенные трансляторы - компляторы.
Интерпретация: на языке B пишется программа, которая на вход получает код на языке A и каждую команду заменяет эквивалентой на языке B и сразу выполняет ее (те новая программа не создается). Данная технология - интерпретация.
Итак, главная особенность джавы - кроссплатформенность, которая достигается следующим образом:
Есть такая JVM - виртуальная машина, которая на вход получает байт-код (внешне индентичен двоичному) и далее его интерпретирует в машинный. Фишка байт-кода в том, что он никак не привязан к архитекторе машины (это просто промежуточное представление программы), засчет чего и достигается кроссплатформенность. Те железу достаточно просто чтобы поддерживалась JVM и код на джаве запустится.

2. Средства разработки. JDK и JRE. Компиляция и выполнение программы. JAR-архивы.

Компиляция — это процесс преобразования исходного кода программы в машинный код. Результатом компиляции является исполняемый файл.
Основные характеристики компиляции:
Процесс компиляции выполняется до запуска программы. Это значит, что весь исходный код компилируется сразу в машинный код, и только после этого программа запускается.
Результат: После компиляции создается исполняемый файл (например, .exe), который можно запускать без необходимости в исходном коде или компиляторе.
Производительность: Программы, скомпилированные в машинный код, как правило, работают быстрее, так как их инструкции уже преобразованы в формат, который понимает процессор.
Ошибки: Ошибки компиляции (например, синтаксические ошибки) обнаруживаются до запуска программы, что делает процесс отладки более предсказуемым.
Интерпретация — это процесс, при котором программа исполняется построчно во время её запуска. Интерпретатор читает и анализирует исходный код программы и сразу же выполняет его, без создания исполняемого файла.
Основные характеристики интерпретации:
Процесс интерпретации выполняется во время исполнения программы. Интерпретатор последовательно читает исходный код, анализирует и выполняет его команду за командой.
Результат: Интерпретация не создает отдельного исполняемого файла; исходный код программы исполняется напрямую.
Производительность: Интерпретируемые программы обычно работают медленнее, чем скомпилированные, так как каждая строка кода анализируется и исполняется на лету.
Ошибки: Ошибки в коде могут обнаруживаться только в процессе выполнения программы, поскольку интерпретатор анализирует код построчно.
Java совмещает оба подхода: сначала происходит компиляция в байт-код с помощью компилятора (javac), а затем интерпретация (или компиляция JIT) байт-кода с помощью JVM.
Компиляция: Исходный код Java (.java) компилируется в байт-код (.class) — промежуточное представление, которое не зависит от конкретной платформы. Интерпретация: JVM может интерпретировать этот байт-код построчно. JIT-компиляция: Для ускорения выполнения JIT-компилятор компилирует часто исполняемый байт-код в машинный код.
Файл, созданный после JIT-компиляции, не сохраняется как отдельный файл на диске. Вместо этого машинный код, сгенерированный JIT-компилятором, хранится в памяти и используется JVM для ускорения выполнения программы. JIT-компилятор (Just-In-Time) не создает исполняемый файл на диске по нескольким причинам, связанных с его целью и особенностями работы:

Основная цель JIT — динамическая оптимизация во время выполнения JIT-компиляция происходит в процессе выполнения программы, а не до её запуска. Основная задача JIT-компилятора — анализировать и оптимизировать часто исполняемые участки кода (так называемые "горячие точки (hot spot)"), чтобы повысить производительность программы во время её работы. Это означает, что JIT-компиляция адаптируется к конкретному состоянию программы на лету. Создание и сохранение исполняемого файла на диск усложнило бы этот процесс, так как JIT-компилятор должен гибко реагировать на изменения в программе и адаптироваться в реальном времени, изменяя код и оптимизации в зависимости от того, как программа работает на данном устройстве. JIT-компилятор сохраняет сгенерированный машинный код в оперативной памяти, что позволяет немедленно использовать его для выполнения программы. Это обеспечивает большую скорость, так как отсутствует необходимость записи на диск и последующего чтения из него. Если бы JIT компилятор создавал исполняемый файл на диске каждый раз, когда он выполняет оптимизации, это могло бы привести к значительному увеличению использования дискового пространства.
JIT-компилятор ориентирован на динамическую оптимизацию и выполнение программы в режиме реального времени. Его задача — оптимизировать код на основе профилирования программы во время её выполнения, а не создавать постоянные исполняемые файлы на диск. Это делает JIT гибким, быстрым и способным адаптироваться к текущим условиям выполнения программы, что невозможно при создании статических исполняемых файлов. Как работает JIT:
Интерпретация байт-кода: Когда программа запускается, JVM сначала интерпретирует байт-код, исполняя его построчно. Это не очень эффективно, но позволяет запустить программу сразу. Профилирование: Во время выполнения JVM отслеживает часто исполняемые методы и участки кода (горячие точки).
Компиляция горячих точек: Когда JIT обнаруживает горячие точки, он компилирует их в машинный код и кеширует результат. Это позволяет значительно ускорить последующее выполнение этих участков кода, поскольку они исполняются напрямую на процессоре без интерпретации. Также JIT-компилятор также выполняет различные оптимизации на основе анализа кода, что дополнительно ускоряет выполнение программы.
JAR-архивы (Java Archive) — это специальный формат файлов, который используется для упаковки множества файлов в один архив. JAR-файл содержит байт-код Java, который может быть выполнен виртуальной машиной Java (JVM).
Основные характеристики JAR-архивов: Формат ZIP: JAR-файлы основаны на стандартном формате ZIP-архивов, что делает их удобными для сжатия и упаковки множества файлов в один. Это уменьшает размер файлов и упрощает их транспортировку и хранение.
JAR-файл может содержать:
Скомпилированные классы Java (файлы .class).
Метаданные (например, манифест).
Ресурсы (например, изображения, аудиофайлы, текстовые файлы), которые могут быть использованы программой.
Другие JAR-файлы, что удобно для зависимостей.
Манифест (MANIFEST.MF): В JAR-файле может быть специальный файл MANIFEST.MF, который содержит данные о содержимом архива. Например, манифест может указывать, какой класс является точкой входа для программы (основной класс с методом main).
Зачем нужны JAR-архивы:
Упрощение распространения приложений и библиотек: JAR-файлы позволяют упаковать все компоненты программы или библиотеки в один файл. Это упрощает распространение Java-приложений и библиотек, так как пользователю или системе нужно передавать только один файл, а не множество отдельных файлов.
Сжатие и оптимизация: Поскольку JAR-файлы используют формат ZIP, они сжимают содержимое, что уменьшает общий размер файлов и экономит место на диске или при передаче данных.
Модульность и зависимые библиотеки: Программы могут состоять из нескольких JAR-файлов, которые представляют отдельные модули или библиотеки. Например, одно приложение может использовать множество внешних библиотек (JAR-файлов), что делает структуру кода более модульной и управляемой.
Исполняемые JAR-файлы: JAR-файлы могут быть исполняемыми. Это значит, что они содержат метаинформацию (в манифесте), указывающую, какой класс с методом main является точкой входа в программу. Это позволяет запускать программу, используя команду: java -jar имяфайла.jar;
jar -cf name.class - создание архива
jar cfm Labpack.jar MANIFEST.MF Lab.class - собираем jar.
Java Runtime Environment (JRE) — это среда выполнения Java, которая предоставляет все необходимое

для запуска Java-программ, но не включает инструменты для разработки. JRE включает:
JVM
Библиотеки классов Java: Библиотеки стандартного API, которые предоставляет Java (например, java.lang, java.util, и другие).
JRE предназначена для конечных пользователей и систем, которые просто выполняют Java-программы. Например, когда вы скачиваете и запускаете Java-приложение, вам нужен JRE для его выполнения, но нет необходимости в компиляторе, так как код уже был скомпилирован в байт-код. Java Development Kit (JDK) — это полный набор инструментов, необходимый для разработки Java-программ. Он включает в себя:
JRE (Java Runtime Environment)
Компилятор Java (javac): Инструмент, который компилирует исходный код Java (.java файлы) в байт-код (.class файлы).
Средства разработки: Инструменты для отладки, профилирования, создания документации и другие утилиты.
Примеры и документация: Примеры кода и документация для разработчиков.

3. Примитивные типы данных в Java. Приведение типов.

Виртуальная машина Java поддерживает следующие примитивные типы: числовые типы, boolean тип и типы с плавающей точкой.
Целые типы
byte, содержит 8-битовые знаковые целые.
Значение по умолчанию - ноль.
short, содержит 16-битовые знаковые целые.
Значение по умолчанию - ноль.
int, содержит 32-битовые знаковые целые.
Значение по умолчанию - ноль.
long, содержит 64-битовые знаковые целые.
Значение по умолчанию - ноль.
char, содержит 16-битовые беззнаковые целые, представляющие собой кодовые точки таблицы символов Unicode в базовой странице UTF-16.
Значение по умолчанию - нулевая кодовая точка ('\u0000')
Типы с плавающей точкой
float, содержит числа с плавающей точкой одинарной точности. Значение по умолчанию - положительный ноль. double, содержит числа с плавающей точкой двойной точности.
Значение по умолчанию - положительный ноль.
Значение boolean типа может быть true или false, значение по умолчанию false.
Стандарт IEEE 754 включает в себя не только положительные и отрицательные значения мантиссы, но также и положительные и отрицательные нули, положительные и отрицательные бесконечности, и специальное не числовое значение NaN (Not-a-Number). NaN используется в качестве результата некоторых неверных операций, таких как деление нуля на нуль.
Приведение типов в Java — это процесс преобразования значения одного типа данных в другой тип. Java строго типизированный язык, что означает, что каждая переменная и объект имеют определённый тип, который нельзя просто так изменить. Однако иногда требуется явным образом преобразовать один тип в другой. Приведение типов бывает двух видов: неявное (автоматическое) и явное (ручное).
Неявное приведение типов (Upcasting) -Это приведение типов, которое происходит автоматически и не требует вмешательства программиста. Java автоматически преобразует тип данных, если это преобразование безопасно, т.е. не вызывает потери данных. Примитивные типы данных: Java автоматически преобразует меньшие примитивные типы данных в большие (например, int в long, float в double).
Явное приведение требуется, когда существует возможность потери данных или когда типы данных не совместимы по умолчанию. Для этого нужно явно указать новый тип данных в скобках перед значением или переменной.
В Java для создания констант используется ключевое слово final. Переменная, объявленная как final, не

может быть изменена после инициализации, то есть её значение устанавливается один раз и остаётся неизменным в течение всей программы.
Преобразование типов подразумевает изменение значения одной переменной одного типа данных на другой тип. Оно может быть явным или неявным, и происходит либо автоматически, либо вручную. Например:
double a = 10.5;
int b = (int) a; // Явное преобразование double в int
Приведение типов — это процесс, при котором одно значение интерпретируется как другой тип. В Java приведение используется для работы с совместимыми типами (например, в иерархии наследования классов или между примитивными типами), но не изменяет данные. Например:
int a = 10;
long b = a; // Неявное приведение int к long
Примитивные типы данных в Java являются основными типами и представляют собой простые значения. Они не являются объектами и хранятся непосредственно в памяти. Примитивные типы не имеют методов. Их поведение определяется только их значением.
Ссылочные типы данных представляют собой объекты и массивы. Они хранят ссылки на объекты или массивы в памяти, а не сами объекты. Переменная сама по себе не хранит объект, а лишь ссылку на него. Ссылочные типы имеют методы и свойства, которые можно использовать для манипуляции данными. Все ссылочные переменные по умолчанию инициализируются значением null, что означает отсутствие ссылки на объект.

4. Работа с переменными. Декларация. Инициализация. Присваивание.

Декларация переменной в Java — это процесс объявления переменной с указанием её типа. На этом этапе переменная создаётся, но её значение ещё не определено.
Инициализация переменной — это процесс присвоения ей значения при или после её объявления. После инициализации переменная содержит определённое значение, которое можно использовать в дальнейшем.
int age = 25; // Инициализация
age = 30; // Присваивание нового значения переменной age

5. Одномерные и двумерные массивы. Декларация и создание массивов. Доступ к элементам массива.

Одномерные массивы представляют собой набор элементов одного типа, доступных по индексу. Индексы в массиве начинаются с 0. После объявления массива, нужно выделить память для него с помощью оператора new, указав количество элементов.
Двумерные массивы представляют собой массивы массивов и могут быть визуализированы как таблицы или матрицы, где у каждого элемента есть два индекса: строка и столбец.
Массивы в Java индексируются с использованием типа int, а это значит, что максимальный индекс элемента массива не может превышать максимально допустимое значение для типа int.
java.util.Arrays — это класс в Java, который предоставляет полезные методы для работы с массивами. Этот класс содержит статические методы для манипуляции массивами любого типа, включая сортировку, сравнение, поиск, копирование и преобразование массивов в строку. Некоторые методы: .fill (заполнить весь массив значением аргумента), .equals (проверка на р-во массивов), .binarySearch, .sort, .toString (вывод элементов массива переводом в строку)

6. Операторы и выражения в Java. Особенности вычисления, приоритеты операций.

В Java операторы используются для выполнения различных операций над переменными и значениями. Выражение состоит из операндов и операторов, и при его вычислении получается некоторое значение.
Когда в выражениях участвуют операнды разных типов, Java автоматически преобразует их к более широкому типу (например, int к double), чтобы избежать потери данных. Это называется автоматическим приведением типов.
В логических выражениях с операторами && (логическое И) и || (логическое ИЛИ) Java использует короткое замыкание. Это означает, что если результат всего выражения уже известен после вычисления первой части, то вторая часть не вычисляется.
Операторы инкремента (++) и декремента (–) могут быть в двух формах — префиксной и постфиксной. Разница заключается в том, когда происходит изменение значения.
Префиксная форма: Сначала изменяется значение, а потом возвращается результат.
int a = 5;
int b = ++a; // сначала a увеличивается до 6, затем b получает значение 6
Постфиксная форма: Сначала возвращается значение, а затем оно изменяется.
int a = 5;
int b = a++; // сначала b получает значение 5, затем a увеличивается до 6
Приоритет операций:
1. () (скобки), [] (индексация массива)
2. ++, –, ! (не), (поразрядное отрицание)
3. *, /, %
4. +, -
5. «, », »> (беззнаковый сдвиг)
6. <, <=, >, >=
7. ==, !=
8. & (поразрядное и)
9. ˆ (xor)
10. | (поразрядное или)
11. &&
12. ?: (тернарный оператор: переменная = выражение: true ? false)
13. операторы присваивания
Оператор << сдвигает биты числа влево на указанное количество позиций. При сдвиге влево освободившиеся справа биты заполняются нулями. Сдвиг влево эквивалентен умножению на степень двойки.
Оператор >> сдвигает биты числа вправо на указанное количество позиций. При сдвиге вправо, старшие биты заполняются значением знакового бита (если число отрицательное, то они будут единицами). Это приводит к делению на степень двойки с округлением в меньшую сторону.
Оператор >>> сдвигает биты числа вправо на указанное количество позиций, заполняя освободившиеся старшие биты нулями, независимо от знака. Этот оператор используется для работы с беззнаковыми целыми числами (в Java это значения типа int и long).

7. Математические функции в составе стандартной библиотеки Java. Класс java.lang.Math.

Класс Math в Java содержит множество статических методов для выполнения стандартных математических операций.
Java предоставляет множество пакетов для решения различных задач, что упрощает разработку программ. Вот несколько ключевых вспомогательных пакетов (Пакетом (пространством имен) в Java называется структура вложенных по какому-то признаку папок с размещенными в них классами):
java.lang — содержит основные классы, такие как String, Math, Integer, System, и другие, которые часто используются в Java-программах. Этот пакет автоматически импортируется во все программы.
java.util — один из самых важных вспомогательных пакетов, который содержит коллекции, такие как списки (List), множества (Set), очереди (Queue), карты (Map), а также классы для работы с датами и

временем, случайными числами и утилиты для работы с массивами.
Пакеты в Java хранятся как структура каталогов в файловой системе. Каждый пакет соответствует директории (папке) на диске, а каждый класс внутри пакета — это .class файл внутри этой директории.
В Java для использования классов и интерфейсов из других пакетов нужно их импортировать. Импорт пакетов позволяет вам использовать классы, которые находятся вне текущего пакета, без необходимости указывать полный путь (имя пакета) каждый раз, когда вы обращаетесь к этим классам.
Импорт конкретного класса: можно импортировать конкретный класс из пакета. Например, если хотите использовать класс ArrayList из пакета java.util, то пишите:
import java.util.ArrayList; Теперь вы можете использовать ArrayList в своем коде, не указывая полный путь java.util.ArrayList.
Импорт всего пакета: Чтобы импортировать все классы и интерфейсы из пакета, можно использовать символ подстановки *. Например: import java.util.*;
Это позволит использовать любой класс из пакета java.util, такой как ArrayList, HashMap, Date и другие, без необходимости импортировать их по отдельности.
Импорт статических методов и полей: Если нужно импортировать статические методы или поля какого-либо класса, можно использовать import static. Например, если вы хотите использовать статические методы класса Math, такие как Math.sqrt() или Math.PI, без необходимости указывать Math каждый раз, вы можете сделать это так:
import static java.lang.Math.*;

8. Подпрограммы, методы, параметры и возвращаемые значения.

Подпрограмма — это общее название для именованного блока кода, который выполняет определенную задачу и может быть вызван из другой части программы. В Java подпрограммы называются методами.
Методы — это подпрограммы в Java, которые принадлежат классу или объекту и выполняют конкретные действия. Каждый метод имеет свое имя, может принимать параметры и возвращать значения.
Синтаксис метода в Java:
<модификаторы доступа> <тип возвращаемого значения> <имя метода>(<список параметров>)
// Тело метода
return <значение>; // Если метод возвращает значение
Модификаторы доступа (public, private, protected) определяют, кто может вызывать метод.
Тип возвращаемого значения указывает, что метод должен вернуть (например, int, String, void).
Имя метода описывает, что делает метод, например, calculateArea, printMessage.
Параметры — это данные, которые передаются методу для его работы. Параметры указываются в круглых скобках при объявлении метода.

9. Форматированный вывод числовых данных.

Метод printf() позволяет выводить строки с форматированием. Этот метод использует форматные спецификаторы, чтобы указать, как выводить числа или строки.
System.out.printf(<строка формата>, <аргументы>);
Строка формата: специальная строка, которая включает в себя текст и форматные спецификаторы (например, %d, %f и т.д.).
%d — для целых чисел (int).
%f — для чисел с плавающей точкой (float, double).
%s — для строк.
%n — перенос строки.
Аргументы: значения, которые будут подставлены на место форматных спецификаторов.

```
1 public class Main {
2     public static void main(String[] args) {
3         int number = 123;
4         double pi = 3.14159;
```

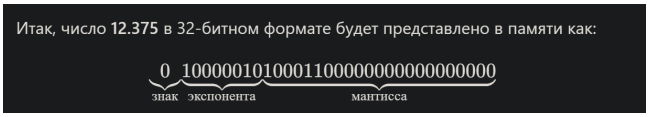
```
5
6           System.out.printf("Number: %d%n", number);
7           // Number: 123
8
9           System.out.printf("PI: %.2f%n", pi);
10          // PI: 3.14
11
12          System.out.printf("Number with leading zeros: %05d%n", number);
13          // Number with leading zeros: 00123
14
15          System.out.printf("Right-aligned: %10d%n", number)
16          //Right-aligned:        123
17
18          System.out.printf("Left-aligned: %-10d%n", number);
19          // Left-aligned: 123
20      }
21  }
```

%d — для целых чисел. Можно указать ширину, выравнивание, нули.
%5d — целое число с шириной 5, выравнено вправо.
%05d — целое число с шириной 5, дополненное ведущими нулями.
%f — для чисел с плавающей точкой (float, double).
%10.2f — число с шириной 10 и двумя знаками после запятой.
%+.2f — число с двумя знаками после запятой и обязательным знаком (например, +3.14).
print() — вывод без переноса строки.
println() — вывод с переносом строки.
printf() — вывод форматированной строки.

10. Дополнительно.

Класс создает новый тип данных, который можно применять для создания объектов. То есть класс создает логическую инфраструктуру, определяющую отношения между его членами. При объявлении объекта класса создается экземпляр этого класса. Таким образом, класс является логической конструкцией, а объект имеет физическую реальность (занимает место в памяти).
Создание класса означает создание нового типа данных, который можно применять для объявления объектов этого типа. Однако получение объектов класса представляет собой двухэтапный роцесс. Во-первых, потребуется объявить переменную типа класса. Такая переменная не определяет объект, а просто может ссылаться на объект. Во-вторых, необходимо получить физическую копию объекта и присвоить ее этой переменной, для чего служит операция new. Операция new динамически (т.е. во время выполнения) выделяет память для объекта и возвращает ссылку на нее, которая по существу является адресом в памяти объекта, выделенной new. Затем ссылка сохраняется в переменной. Таким образом, в Java все объекты класса должны размещаться динамически.
Box mybox; // объявить ссылку на объект
mybox = new Box() ; // разместить в памяти объект Box
NaN возникает в результате операций, которые по своей природе не могут быть определены в области реальных чисел. Вот несколько распространенных случаев:
Деление 0 на 0
Корень квадратный из отрицательного числа
Операции с NaN: Любые арифметические операции с NaN приводят к NaN
При некорректных операциях с бесконечностями: Например, если вы пытаетесь вычислить разность двух бесконечностей
NaN не равно ни себе, ни другим значениям: В Java и других языках программирования NaN считается "неопределенным поэтому сравнение NaN с любым другим числом, включая само NaN, всегда возвращает false.
* ... *\ - многострочные комментарии
\\ - однострочные
** ... *\ - документация

Числа с плавающей точкой, такие как дробные числа, хранятся в памяти компьютера в соответствии с стандартом IEEE 754. Этот стандарт определяет, как представлять вещественные числа (дробные) в виде бинарного числа с плавающей точкой. В Java для представления дробных чисел используются два типа: float (32-битное представление) и double (64-битное представление).
Число с плавающей точкой представляется как:
$(-1)^{\text{знак}} \times (1 + \text{мантисса}) \times 2^{\text{экспонента - смещение}}$
Это означает, что число кодируется тремя компонентами:
Знак (sign) — указывает, положительное это число или отрицательное.
Экспонента (exponent) — определяет масштаб числа (насколько большое или маленькое число).
Мантисса (mantissa) — хранит значащие цифры числа, отвечающие за его точность.
Тип float (32 бита)
Число float в Java занимает 32 бита, которые делятся на следующие части:
1 бит для знака: 0 для положительного числа, 1 для отрицательного.
8 бит для экспоненты: используется для хранения степени двойки.
23 бита для мантиссы: хранит значащие цифры числа, с ведущей единицей (нормализованное число).
Тип double (64 бита)
Число double занимает 64 бита:
1 бит для знака.
11 бит для экспоненты: позволяет хранить больший диапазон значений.
52 бита для мантиссы: обеспечивает более высокую точность по сравнению с float.
Допустим, у нас есть число 12.375 в десятичной системе. В памяти оно будет храниться следующим образом:
Перевод числа в двоичную систему:
Целая часть 12 — это 1100 в двоичной системе. Число 12.375 в двоичной системе выглядит как: 1100.011.
Запись в формате IEEE 754: 1.100011×2^3
Мантисса: Хранит дробную часть числа без ведущей единицы (всегда предполагается единица перед дробной частью в нормализованном числе). Мантисса будет 100011, а оставшиеся биты заполняются нулями.
Экспонента: Степень 3 (так как число умножается на 3), и в формате float экспонента сдвигается на 127 (смещение для 32-битного числа). Значит, экспонента будет 130, что в двоичном виде — 10000010.



Особенности:
Потеря точности: Не все числа могут быть точно представлены в двоичной системе, особенно такие, как 0.1 или 0.3, которые не имеют точного двоичного представления. Это приводит к погрешностям при вычислениях с числами с плавающей точкой.
Переполнение и денормализованные числа: Когда число слишком большое для хранения в заданном формате, происходит переполнение (в результате получается бесконечность, Infinity). Когда число слишком маленькое, оно может стать денормализованным, теряя точность, или стать нулем.
Экспонента хранится не напрямую, а в смещенном виде — это значит, что к фактическому значению экспоненты добавляется специальная константа, называемая смещением (bias). Это сделано для того, чтобы экспонента всегда была положительным числом, что упрощает хранение и сравнение чисел. Смещение различается в зависимости от типа числа:
Для типа float (32 бита) смещение = 127.
Для типа double (64 бита) смещение = 1023.
В Java модификаторы доступа (access modifiers) определяют уровень видимости и доступности классов, методов, полей и конструкторов для других классов и пакетов. Всего существует четыре основных модификатора доступа:
Public
Поля, методы, конструкторы и классы, отмеченные как public, доступны из любого места в программе, то есть из любого класса и любого пакета.
Private
Элементы, отмеченные как private, видны и доступны только внутри самого класса (вне класса к ним можно обратиться через методы). Они не видны ни для каких других классов, даже если они находятся в том же пакете.

Protected
Поля, методы и конструкторы, помеченные как protected, доступны внутри самого класса, внутри классов того же пакета, в подклассах (наследниках) в других пакетах.
Package-Private или Default
Если не указать явный модификатор, элемент класса (или сам класс) будет доступен только для классов, находящихся в том же пакете. Этот модификатор называют package-private или по умолчанию (default).
Static и final
Ключевое слово static используется для обозначения элементов класса, которые принадлежат классу, а не конкретному экземпляру объекта. Это значит, что static члены класса (поля, методы) существуют в единственном экземпляре для всего класса, независимо от того, сколько объектов этого класса создано. Те можно юзать методы класса без создания конкретных экземпляров класса (static method), определять поля (переменные) общие для всех объектов класса (static <type> <NameOfVariable>). Однако статические методы могут напрямую вызывать только другие статические методы своего класса и могут напрямую получать доступ только к статическим переменным своего класса, к тому же их нельзя использовать вместе с методом this по понятным причинам (this связан с объектом).
Поле может быть объявлено как final (финальное), что предотвращает изменение его содержимого, делая его по существу константой. Это означает, что поле final должно быть инициализировано при его объявлении. Объявление параметра как final предотвращает его зменение внутри метода. Объявление локальной переменной как final предотвращает присваивание ей значения более одного раза.