Лабораторная работа по программированию 1

1.Язык Java. Особенности языка.

На сегодняшний момент язык Java является одним из самых распространенных и популярных языков программирования. Первая версия языка появилась еще в 1996 году в недрах компании Sun Microsystems, впоследствии поглощенной компанией Oracle. Java задумывался как универсальный язык программирования, который можно применять для различного рода задач.

Текущая версия – 22 которая вышла в марте 2024

Ключевой особенностью языка Java является то, что его код сначала транслируется в специальный байт-код, независимый от платформы. А затем этот байт-код выполняется виртуальной машиной JVM (Java Virtual Machine). В этом плане Java отличается от стандартных интерпретируемых языков как PHP или Perl, код которых сразу же выполняется интерпретатором. В то же время Java не является и чисто компилируемым языком, как С или С++.

Для каждой из платформ может быть своя реализация виртуальной машины JVM, но каждая из них может выполнять один и тот же код.

Java является языком с Си-подобным синтаксисом и близок в этом отношении к C/C++ и C#. Поэтому, если вы знакомы с одним из этих языков, то овладеть Java будет легче.

Еще одной ключевой особенностью Java является то, что она поддерживает автоматическую сборку мусора. А это значит, что вам не надо освобождать вручную память от ранее использовавшихся объектов, как в С++, так как сборщик мусора это сделает автоматически за вас.

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

**JDK (Java Development Kit)** — это бесплатный комплект инструментов для создания и запуска программ при разработке приложений на языке Java.

**JRE (Java Runtime Environment)** — это среда выполнения Java. Программы на Java можно запускать на выполнение, только если установлена JRE. Она действует как посредник между программой и операционной системой, позволяет выполнять программу на разных устройствах и ОС.

JRE можно использовать отдельно от JDK, если нужно не разрабатывать, а только запускать приложения. В то же время JDK не применяется без JRE — программы без запуска и отладки создавать невозможно.

**JAR-файл** — это Java-архив (**J**ava **AR**chive). Это простой архивный файл, сжатый (иногда с нулевой компрессией) по алгоритму **zip**.

Он был создан для удобства распространения программ, написанных на **Java**. Так как обычная программа содержит сотни, тысячи, а иногда и миллионы файлов. Файл может содержать:

* файл манифеста META-INF/MANIFEST.MF
* **java-файлы** (исходный код)
* **class-файлы**
* файлы, необходимые для работы программы: картинки, файлы с настройками и прочее (ресурсы)
* электронные подписи, которые позволяют защитить программу от модификации

**Манифест** - это текстовый файл формата ключ: значение; он содержит описание **jar-файла**.

3.Примитивные типы данных в Java.Приведение типов

 Целочисленные типы: byte, short, int, long.

 Символьный тип: char.

 С плавающей точкой: float, double.

 Логический тип: boolean.

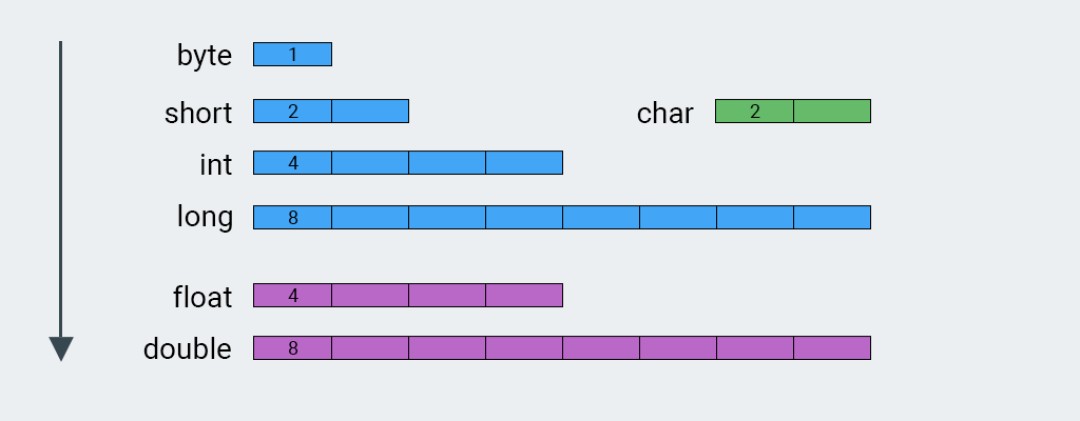
<https://sky.pro/media/kakie-tipy-dannyh-sushhestvuyut-v-java/>

Переменные примитивных типов (за исключением типа boolean) используются для хранения разных типов чисел. И хоть типы переменных всегда неизменны, есть место, где можно проводить преобразование типов. И место это — *присваивание*.

Можно присваивать друг другу переменные разных типов. При этом значение, взятое из переменной одного типа, будет преобразовано в значение другого типа и присвоено второй переменной. В связи с этим можно выделить два вида преобразования типов: расширение и сужение.

**Расширение типа** похоже на перекладывание из маленькой корзинки в большую — операция проходит незаметно и безболезненно. **Сужение типа** — это перекладывание из большой корзинки в маленькую: места может не хватить, и что-то придётся выбросить.

Вот типы, отсортированные по размеру «корзинки»:



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

<https://javarush.com/groups/posts/peremennie-v-java>

**Декларация в Java** — это оператор, который используется для объявления переменной, указывая её тип данных и имя.

Вот несколько примеров деклараций:

int number;

Инициализация переменной в Java — это процесс задания ей начального значения перед использованием.

Для инициализации переменной используют оператор присваивания. Слева указывается имя переменной, справа — её значение.

Пример инициализации:

int x = 10; // объявление и инициализация переменной

System.out.println(x); // 10

Оператор присваивания в Java — это знак равенства, который используется для задания переменной нужного значения.

Например, в выражении x = 1 переменной x присваивается значение 1. В любых выражениях, где присутствует x, компьютер будет подставлять на его место единицу, пока разработчик не перепишет значение этой переменной.

С помощью оператора присваивания можно записать в переменную результат вычисления. Например, x = 1 + 2. Теперь компьютер присвоит переменной x значение 3.

В переменную можно даже записать результат другого присваивания. Например, х = у = 1. В переменную x запишется результат выражения y = 1. Если y изменится, вслед за ним изменится и x.

Важно помнить, что в Java данные перетекают справа налево.

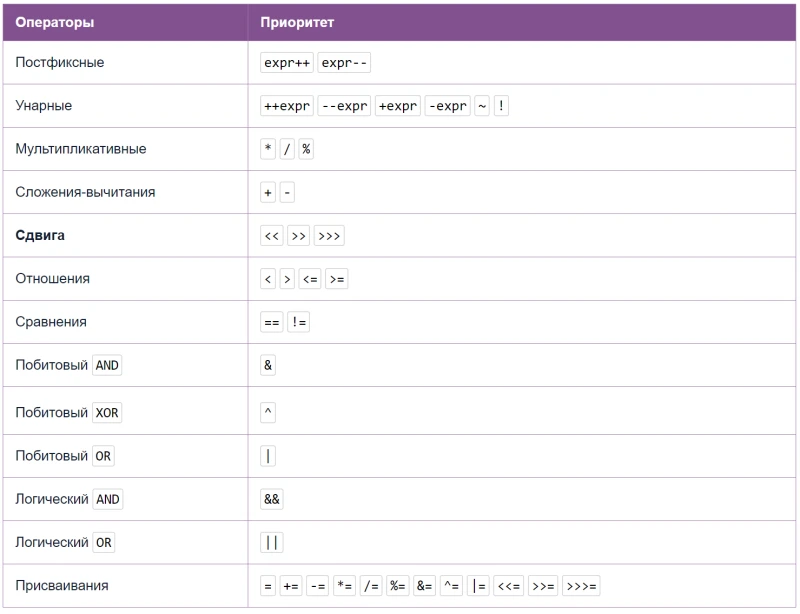
5. Одномерные и двумерные массивы. Декларация и создание массивов. Доступ к элементам массива.

<https://skillbox.ru/media/base/massivy_v_java_sozdayem_zapolnyaem_ispolzuem/>

6. Инструкции ветвления (if-else, switch) и циклов (do, while, for).

<https://java-online.ru/java-if-else.xhtml>

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



<https://practicum.yandex.ru/blog/operatory-java/>

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

<https://metanit.com/java/tutorial/12.1.php>

**Java.lang.Math** — это класс, который предоставляет набор статических методов для выполнения различных математических операций в Java.

**Некоторые основные методы класса Math**:

**abs()**.  Возвращает абсолютное значение для аргумента.

**max()**.  Возвращает большее из двух возможных значений.

**min()**.  Возвращает меньшее из двух значений.

**round()**.  Округляет десятичные числа до ближайшего целого числа.

**sqrt()**.  Вычисляет квадратный корень числа.

**cbrt()**.  Возвращает кубический корень числа.

Класс Math располагается в пакете **java.lang**, что означает, что он автоматически доступен всем программам на Java.  Чтобы использовать его в программе, нужно просто импортировать его в начале кода: **import java.lang.Math**.

9. Подпрограммы, методы, параметры и возвращаемые значения.

<https://silvertests.ru/GuideView.aspx?id=35371>

<https://silvertests.ru/GuideView.aspx?id=35371>

<https://dan-it.gitlab.io/fs-book/java-basic/intro/methods.html>

10. Форматированный вывод числовых данных

<https://study-java.ru/uroki-java/formatirovanie-chisel-i-texta-v-java/>

1. JDK (Java Development Kit): Это набор инструментов, который позволяет разработчикам создавать приложения на Java. JDK включает в себя компилятор (javac), инструменты для отладки, библиотеки классов и другие утилиты, необходимые для разработки программ на языке Java. Все, что касается разработки, находится в JDK.

2. JRE (Java Runtime Environment): Это среда выполнения Java, которая позволяет запускать приложения, написанные на Java. JRE включает в себя JVM, стандартные библиотеки Java и другие компоненты, необходимые для выполнения Java-приложений. JRE не включает инструменты для разработки, такие как компилятор.

3. JVM (Java Virtual Machine): Это виртуальная машина, которая выполняет байт-код Java. JVM отвечает за выполнение Java-программ и обеспечивает выполнение кода на любой платформе, поддерживаемой Java. Это значит, что программы, написанные на Java, могут работать на любых устройствах, где установлена JVM, благодаря механизму "один раз написано — везде запущено".

В резюме:

- JDK — это комплект для разработки, который включает JRE и инструменты для разработки.

- JRE — это среда для выполнения Java, содержащая JVM.

- JVM — это виртуальная машина, которая выполняет байт-код Java.

Стек - это память, выделенная в качестве свободного места для потока выполнения. При вызове функции в верхней части стека резервируется блок для локальных переменных и некоторых бухгалтерских данных. Когда эта функция возвращается, блок становится неиспользуемым и может быть использован при следующем вызове функции. Стек всегда резервируется в порядке LIFO (последний в первом исходящем); следующим освобождаемым блоком всегда является самый последний зарезервированный блок. Это действительно упрощает отслеживание стека; освобождение блока из стека - это не что иное, как настройка одного указателя.

Куча - это память, выделенная для динамического распределения. В отличие от стека, здесь нет принудительного шаблона для выделения и освобождения блоков из кучи; вы можете выделить блок в любое время и освободить его в любое время. Это значительно усложняет отслеживание того, какие части кучи выделены или свободны в любой момент времени; доступно множество пользовательских распределителей кучи для настройки производительности кучи для различных шаблонов использования.

Каждый поток получает стек, в то время как для приложения обычно существует только одна куча

Кучи и стек являются двумя основными структурами данных, используемыми для управления памятью в программировании. Они выполняют разные функции и имеют различные характеристики. Вот их основные особенности:

-- Стек

1. Определение: Стек — это структура данных, работающая по принципу "последний пришёл — первый вышел" (LIFO). Это означает, что последний элемент, добавленный в стек, будет первым, который из него извлечён.

2. Использование: Стек обычно используется для хранения информации о вызовах функций (вызов стека), локальных переменных и для управления потоком выполнения программы. При каждом вызове функции в стек помещаются её параметры, локальные переменные и адрес возврата.

3. Размер: Размер стека ограничен и обычно определяется во время запуска программы. Если стек переполнен (например, из-за глубокой рекурсии), это приводит к ошибке "переполнения стека".

4. Управление памятью: Память в стеке автоматически освобождается, когда функция завершается. При выходе из функции все её локальные переменные и параметры удаляются из стека.

-- Куча

1. Определение: Куча — это структура данных, которая позволяет динамически выделять и освобождать память в процессе выполнения программы. В отличие от стека, куча работает по принципу произвольного доступа.

2. Использование: Куча используется для хранения объектов и данных, размер которых заранее неизвестен, и которые могут меняться во время работы программы (например, массивы, структуры, объекты).

3. Размер: Куча обычно имеет гораздо больший объём памяти, чем стек, и размер может изменяться в зависимости от потребностей программы. Однако, управление памятью на куче может быть более сложным.

4. Управление памятью: Память в куче требует явного управления. Программист должен явно выделять (например, с помощью операторов new в C++ или malloc в C) и освобождать (например, с помощью delete или free) память. Если память не освобождается, это приводит к утечкам памяти.

-- Основные отличия

- Принцип работы: Стек LIFO (последний пришёл — первый вышел), куча — произвольный доступ.

- Размер: Размер стека обычно ограничен, в то время как куча может быть значительно больше и изменяемой.

- Управление памятью: Память в стеке управляется автоматически, а в куче — вручную.

- Использование: Стек используется для хранения локальных переменных и информации о вызовах функций, а куча — для динамически созданных объектов и структур данных.

Динамическое выделение и освобождение памяти — это процесс управления памятью в программировании, который позволяет создавать и уничтожать объекты и структуры данных во время выполнения программы, а не на этапе компиляции. Вот более подробное объяснение:-

Байт-код Java и машинный код отличаются по своей природе, способу исполнения и платформенной независимости. Вот основные отличия между ними:

1. Определение:

- Байт-код Java: Это промежуточный код, который генерируется компилятором Java (javac) из исходного кода Java. Байт-код не является нативным кодом для конкретного процессора, а предназначен для выполнения на платформе Java Virtual Machine (JVM).

- Машинный код: Это нативный код, который непосредственно понимается и исполняется процессором. Каждая архитектура процессора имеет свой набор команд и форматов машинного кода.

2. Платформенная зависимость:

- Байт-код Java: Он является платформонезависимым. Программы на Java могут быть скомпилированы в байт-код и выполняться на любой платформе, где установлена соответствующая JVM. Это достигается благодаря принципу "один раз написано — везде запущено".

- Машинный код: Он зависит от конкретной архитектуры процессора. Машинный код, сгенерированный для одной архитектуры (например, x86), не будет работать на другой (например, ARM).

3. Исполнение:

- Байт-код Java: Выполняется интерпретатором или компилятором Just-In-Time (JIT) в JVM, которая преобразует байт-код в машинный код на лету перед выполнением.

- Машинный код: Выполняется напрямую процессором без каких-либо дополнительных преобразований.

4. Производительность:

- Байт-код Java: Может иметь более низкую производительность по сравнению с машинным кодом из-за необходимости интерпретации или компиляции JIT. Однако JIT-компиляция может значительно улучшить производительность выполнения.

- Машинный код: Обычно выполняется быстрее, так как процессор работает с кодом напрямую, без дополнительных преобразований.

В общем, байт-код Java служит средним уровнем, обеспечивая платформенную независимость, в то время как машинный код предназначен для конкретных аппаратных условий и обеспечивает высокую производительность.

1. что делает javac

javac — это компилятор языка программирования Java, который преобразует исходный код, написанный в файлах с расширением .java, в байт-код. Байт-код — это промежуточный код, который может быть выполнен Java Virtual Machine (JVM).

Основные функции javac включают:

1. Компиляция исходного кода: javac принимает файлы .java в качестве входных данных и создает соответствующие файлы .class, содержащие байт-код.

2. Проверка синтаксиса: Во время компиляции javac проверяет синтаксис и семантику кода. Если есть ошибки, такие как неверные конструкции, отсутствующие методы или классы, невызываемые методы и т.д., компилятор выдаст сообщения об ошибках.

3. Поддержка нескольких файлов: javac может компилировать несколько файлов одновременно. Например, вы можете передать в качестве аргументов несколько файлов .java.

4. Опции компиляции: javac предоставляет различные параметры командной строки для настроек компиляции, таких как указание целевой версии Java, использование библиотек и директорий, управление предупреждениями и так далее.

Пример использования:

javac MyClass.java

После выполнения этой команды, если нет ошибок, будет создан файл MyClass.class, который содержит байт-код, готовый для выполнения на JVM.

2. f ключ в jar

В команде jar cfm MyArchive.jar Manifest.txt MyClass.class флаги c, f и m имеют следующие значения:

- c — создать новый архив (create).

- f — указать имя файла архива (file).

- m — использовать указанный файл манифеста (manifest).

Таким образом, эта команда создает новый JAR-файл с именем MyArchive.jar, используя манифест из файла Manifest.txt и добавляя в архив класс MyClass.class.

- String.format

String.format в Java — это метод, позволяющий форматировать строки, используя заданные шаблоны. Он обеспечивает гибкий и удобный способ создания строк с подстановкой значений переменных, а также контроля их отображения.

Вот основные моменты о String.format:

1. Синтаксис:

Метод принимает два основных параметра: строку формата и значения, которые будут подставлены в строку.

Пример:

String formattedString = String.format("Hello, %s! You have %d new messages.", name, messageCount);

2. Спецификаторы формата:

В строке формата используются спецификаторы %-символы, которые начинаются с процента (%) и определяют, как будет отображаться соответствующее значение.

Некоторые распространенные спецификаторы:

- %s — строка

- %d — целое число

- %f — число с плавающей запятой

- %x — шестнадцатеричное число

3. Указание параметров:

Вы можете добавлять дополнительные параметры, такие как ширина поля и точность.

Пример:

String formattedString = String.format("%.2f", pi); // Оставит только 2 знака после запятой для числа

4. Работа с массивами:

Можно также форматировать элементы массивов.

Пример:

String names = {"Alice", "Bob", "Charlie"};

String formatted = String.format("Names: %s, %s, %s", names0, names1, names2);

5. Локализация:

Метод также поддерживает локализацию, позволяя форматировать числа и даты в зависимости от языка и региона.

String.format полезен для создания хорошо отформатированных строк и особенно эффективен, когда необходимо комбинировать текст и переменные для создания динамических сообщений.

Интерпретация и компиляция — это два различных подхода к выполнению программного кода, каждый из которых имеет свои особенности и преимущества.

1. Компиляция:

- Процесс компиляции включает преобразование исходного кода, написанного на языке программирования (например, Java, C++), в машинный код или байт-код, который может быть непосредственно выполнен процессором или виртуальной машиной.

- Компилятор анализирует весь код перед его выполнением и создает исполняемый файл. Этот файл можно запускать независимоя от исходного кода.

- Преимущества:

- Быстрая производительность, так как код уже скомпилирован в машинный формат.

- Проверка ошибок на этапе компиляции, что помогает избежать проблем во время выполнения.

- Недостатки:

- Время на компиляцию кода может быть значительным, особенно для больших программ.

- Необходимость повторной компиляции после внесения изменений в исходный код.

2. Интерпретация:

- Интерпретатор выполняет исходный код построчно или блочно, переводя каждую строку в машинный код на лету.

- Нет необходимости создавать отдельный исполняемый файл; код выполняется программой-интерпретатором (например, Python, JavaScript).

- Преимущества:

- Удобство разработки, так как не требуется компиляция после каждого изменения; можно сразу выполнять код.

- Более высокая гибкость, так как код может быть изменен и выполнен динамически.

- Недостатки:

- Обычно медленнее, чем скомпилированный код, так как интерпретатор тратит время на анализ каждой строки во время выполнения.

- Ошибки могут возникать в процессе выполнения, что может затруднить отладку.

В некоторых случаях используются смешанные подходы, такие как языки с компиляцией в промежуточный байт-код, который затем интерпретируется (например, Java).

jit компеляция

https://ru.wikipedia.org/wiki/JIT-компиляция

<https://wiki.merionet.ru/articles/chto-takoe-jit-kompilyaciya>

https://fullstackguy.ru/blog/2022/08/10/how-jit-compilation-works/

JIT-компиляция (Just-In-Time Compilation) — это процесс, применяемый в виртуальных машинах, таких как Java Virtual Machine (JVM) и .NET Common Language Runtime (CLR), для динамической компиляции промежуточного кода в машинный код во время выполнения программы. Основные аспекты JIT-компиляции включают:

1. Динамическая компиляция:

- JIT-компиляция происходит во время выполнения программы, что позволяет компилировать только те части кода, которые действительно используются, вместо компиляции всего кода заранее.

2. Улучшение производительности:

- JIT-компилятор анализирует код и оптимизирует его, что может привести к значительному увеличению производительности по сравнению с интерпретацией. Скомпилированный код выполняется быстрее, так как он уже преобразован в машинный код.

3. Гибкость:

- JIT-компиляция позволяет перекомпилировать код на лету, если анализирует, что определенные функции часто вызываются, что повышает эффективность выполнения.

4. Кэширование:

- После того как метод был скомпилирован JIT-компилятором, он может сохраняться в кэше, чтобы его не пришлось компилировать повторно при следующих вызовах, что дополнительно ускоряет выполнение программы.

5. Комбинация с интерпретацией:

- Во многих системах JIT-компиляция работает в паре с интерпретацией. Сначала код может интерпретироваться для быстрого старта, а в процессе выполнения некоторые части могут быть JIT-компилированы для повышения производительности.

6. Оптимизации:

- JIT-компиляторы могут применять различные оптимизации, такие как удаление неиспользуемого кода, оптимизация циклов и инлайн-замена методов, что улучшает производительность скомпилированного кода.

Таким образом, JIT-компиляция является мощным инструментом, который сочетает преимущества компиляции и интерпретации, улучшая производительность программ, написанных на языках с виртуальными машинами, таких как Java