Контрольные вопросы и ответы:

Отличие компилятора и интерпретатора:

Интерпретатор и компилятор – это инструменты позволяющие преобразовать исходный код программы в код понятный машине.

Схема работы компилятора:

берёт исходный код программы и анализирует его;

преобразует программный код в машинный код;

создаёт файл с байт кодом (зачастую это exe файл);

передаёт запускной файл ОС.

Схема работы интерпретатора:

берёт исходный код программы и анализирует его;

интерпретирует программный код в машинный код и сразу его выполняет.

Основная разница между ними состоит в следующем:

Компилятор обрабатывает сразу весь программный код после чего выдаёт результат, интерпретатор же берет одну инструкцию, преобразует и выполняет ее, а затем берет следующую инструкцию.

Компилятор генерирует отчет об ошибках после преобразования, а интерпретатор при нахождении ошибки прекращает свою работу.

Компилятор по сравнению с интерпретатором требует больше времени для анализа и обработки программного кода.

Созданный компилятор запускной файл программы будет работать быстрее чем запуск той же программы через интерпретатор.

Особенности Java:

**Простота**

Синтаксис языка был унаследован от C++. Сегодня на фоне Python, Groove или Go его трудно назвать простым, однако тогда эволюционный вид позволил привлечь внимание Си-разработчиков.

**Надёжность**

Надёжность обеспечивается двумя принципами:

ООП. Иерархия наследования увеличивает читаемость кода и снижает количество невынужденных ошибок.

Строгая типизация. Разработчику приходится выполнять больший объём работы, но данные интерпретируется однозначно.

Кроме того, первоначально в Java предполагался запрет прямого доступа к памяти, что также повышало бы надёжность. Но разработчики оставили несколько лазеек, например бэкдор sun.misc.Unsafe, которые этот запрет обходят.

**Безопасность**

Кроме сохранения общей формы конструкций, Java по сравнению с C++ формально лишился двух потенциальных опасностей: указателей и множественного наследования. На деле обе функции сохранены, но представлены в ином виде: вместо указателей используются значения, а в множественном наследовании участвуют не классы, а интерфейсы. Тем не менее, такая особенность java программирования почти исключает возможный урон от невнимательности разработчика.

**Удобство**

Дословно концепция Java звучит как: «Write once, run anywhere». То есть исполняемость кода не зависит от используемой операционной системы или установленного ПО. Достигается это благодаря транслированию в байт-код виртуальной машиной JVM.

Как нельзя кстати пригодилась эта особенность java на android. Разнообразие производителей, моделей телефонов, характеристик — всё это могло бы негативно сказаться на работе приложений, если бы не существование такого универсального инструмента.

**Производительность**

Особенность Java, связанная с транслированием в байт-код, положительно сказывается и на производительности конечных продуктов. По скорости исполнения однотипные программы на java [уступают](https://benchmarksgame.alioth.debian.org/u64q/java.html) в 1,5-2 раза программам на C/C++, при этом превосходят JavaScript, Ruby, Python.

**Развитая экосистема**

За 22 года жизни язык оброс десятками IDE и фреймворков, сотнями сообществ и форумов, тысячами библиотек и плагинов. Всё это благоприятно сказывается на пороге вхождения в профессию, востребованности и качеству производимых с помощью java продуктов.

Безусловно, каждый популярный язык программирования уникален, каждый имеет свои недостатки и преимущества. Особенности Java не имеют революционного характера, они незначительны, но вместе с тем фундаментальны. Именно то, что отличает хороший язык от лучшего.

JDK и JRE:

Различие заключается в том, что JDK представляет собой пакет инструментов для разработки программного обеспечения, тогда как JRE представляет собой пакет инструментов для запуска Java-кода. JRE может использоваться, как отдельный компонент для простого запуска Java-программ, либо быть частью JDK. JDK требуется JRE, потому что запуск программ является неотъемлемой частью их разработки.

Примитивные типы данных:

целые числа (byte, short, int, long)

числа с плавающей точкой (float, double)

логический (boolean)

символьный (char)

Инструкции циклов и ветвления:

**while** – цикл с *предусловием* – сначала проверяем условие, затем выполняем тело цикла;

**do… while** – цикл с *постусловием* – сначала выполняем один раз тело цикла, затем проверяем условие и, если оно соблюдается, продолжаем;

**for** – цикл со счётчиком – выполняется и при каждой итерации обновляет счётчик, пока условие в объявлении цикла соблюдается (т.е. проверка условия возвращает true);

**сокращенный for** (в других языках известен как foreach) – перебирает массив от первого элемента до последнего и на каждой итерации выполняет тело цикла.

|  |  |  |
| --- | --- | --- |
| Оператор | Описание | Пример |
| == | Проверяет, равны или нет значения двух операндов, если да, то условие становится истинным | (A == B) – не верны |
| != | Проверяет, равны или нет значения двух операндов, если значения не равны, то условие становится истинным | (A != B) – значение истинна |
| > | Проверяет, является ли значение левого операнда больше, чем значение правого операнда, если да, то условие становится истинным | (A > B) – не верны |
| < | Проверяет, является ли значение левого операнда меньше, чем значение правого операнда, если да, то условие становится истинным | (A < B) – значение истинна |
| >= | Проверяет, является ли значение левого операнда больше или равно значению правого операнда, если да, то условие становится истинным | (A >= B) – значение не верны |
| <= | Проверяет, если значение левого операнда меньше или равно значению правого операнда, если да, то условие становится истинным | (A <= B) – значение истинна |

|  |  |  |
| --- | --- | --- |
| + | Складывает значения по обе стороны от оператора | A + B даст 30 |
| - | Вычитает правый операнд из левого операнда | A - B даст -10 |
| \* | Умножает значения по обе стороны от оператора | A \* B даст 200 |
| / | Оператор деления делит левый операнд на правый операнд | B / A даст 2 |
| % | Делит левый операнд на правый операнд и возвращает остаток | B % A даст 0 |
| ++ | Инкремент - увеличивает значение операнда на 1 | B++ даст 21 |
| -- | Декремент - уменьшает значение операнда на 1 | B-- даст 19 |

арифметические операторы;

операторы сравнения;

побитовые операторы;

логические операторы;

операторы присваивания;

прочие операторы.

|  |  |  |
| --- | --- | --- |
| & (побитовое и) | Бинарный оператор AND копирует бит в результат, если он существует в обоих операндах. | (A & B) даст 12, который является 0000 1100 |
| | (побитовое или) | Бинарный оператор OR копирует бит, если он существует в любом из операндов. | (A | B) даст 61 который равен 0011 1101 |
| ^ (побитовое логическое или) | Бинарный оператор XOR копирует бит, если он установлен в одном операнде, но не в обоих. | (A ^ B) даст 49, которая является 0011 0001 |
| ~ (побитовое дополнение) | Бинарный оператор дополнения и имеет эффект «отражения» бит. | (~ A) даст -61, которая является формой дополнением 1100 0011 в двоичной записи |
| << (сдвиг влево) | Бинарный оператор сдвига влево. Значение левых операндов перемещается влево на количество бит, заданных правым операндом. | A << 2 даст 240, который 1111 0000 |
| >> (сдвиг вправо) | Бинарный оператор сдвига вправо. Значение правых операндов перемещается вправо на количество бит, заданных левых операндом. | A >> 2 даст 15, который является 1111 |
| >>> (нулевой сдвиг вправо) | Нулевой оператор сдвига вправо. Значение левых операндов перемещается вправо на количество бит, заданных правым операндом, а сдвинутые значения заполняются нулями. | A >>> 2 даст 15, который является 0000 1111 |

В Java, наивысший приоритет имеют операции постинкремент(i++), постдекремент(i--), префиксный инкремент(++i) и префиксный декремент(--i), меньший приоритет имеют умножение(\*), деление(/), остаток от деления(%), ну и замыкающие с самым наименьшим приоритетом сложение(+) и вычитание(-).

Спецификатор формата Выполняемое форматирование

%a Шестнадцатеричное значение с плавающей точкой

%b Логическое (булево) значение аргумента

%c Символьное представление аргумента

%d Десятичное целое значение аргумента

%h Хэш-код аргумента

%e Экспоненциальное представление аргумента

%f Десятичное значение с плавающей точкой

%g Выбирает более короткое представление из двух: %е или %f

%o Восьмеричное целое значение аргумента

%n Вставка символа новой строки

%s Строковое представление аргумента

%t Время и дата

%x Шестнадцатеричное целое значение аргумента

%% Вставка знака %

Спецификатор формата Выполняемое преобразование

%tH Час (00 – 23)

%tI Час (1 – 12)

%tM Минуты как десятичное целое (00 – 59)

%tS Секунды как десятичное целое (00 – 59)

%tL Миллисекунды (000 – 999)

%tY Год в четырехзначном формате

%ty Год в двузначном формате (00 – 99)

%tB Полное название месяца (“Январь”)

%tb или %th Краткое название месяца (“янв”)

%tm Месяц в двузначном формате (1 – 12)

%tA Полное название дня недели (“Пятница”)

%ta Краткое название дня недели (“Пт”)

%td День в двузначном формате (1 – 31)

%tR То же что и «%tH:%tM«

%tT То же что и «%tH:%tM:%tS«

%tr То же что и «%tI:%tM:%tS %Tp» где %Tp = (AM или PM)

%tD То же что и «%tm/%td/%ty«

%tF То же что и «%tY–%tm–%td«

%tc То же что и «%ta %tb %td %tT %tZ %tY«