Контрольные вопросы часть 1. Базовый синтаксис.

|  |  |
| --- | --- |
| **виртуальная машина Java** — основная часть исполняющей системы **Java**, так называемой **Java**Runtime Environment (JRE). **Виртуальная машина Java** исполняет байт-код **Java**, предварительно созданный из исходного текста **Java**-программы компилятором **Java** (javac). | Что такое виртуальная машина? |
| Java — объектно-ориентированный язык программирования  **Машинные языки и машинно-ориентированные языки** — это языки **низкого уровня**, требующие указания мелких деталей процесса обработки данных. Языки же **высокого уровня**имитируют естественные языки, используя некоторые слова разговорного языка и общепринятые математические символы. Эти языки более удобны для человека. | К какому типа языка программирование относиться Java? |
| * компилятор с языка Java на внутренний язык Java Byteсode (JavaBC): Java-программы и апплеты распространяются по WWW и интерпретируются на JavaBC; * загрузчик-верификатор программ на JavaBC; * интерпретатор JavaBC, называемый виртуальной машиной языка Java (JavaVM - Java Virtual Machine); * многочисленные библиотеки классов и утилиты, существенно упрощающие программирование на Java. | Из каких компонентов состоит Java? |
| **Java Development Kit** (сокращенно *JDK*) — бесплатно распространяемый компанией [*Oracle Corporation*](https://ru.wikipedia.org/wiki/Oracle_Corporation) (ранее [*Sun Microsystems*](https://ru.wikipedia.org/wiki/Sun_Microsystems)) [комплект разработчика приложений](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BB%D0%B5%D0%BA%D1%82_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%87%D0%B8%D0%BA%D0%B0_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) на языке [*Java*](https://ru.wikipedia.org/wiki/Java), включающий в себя [компилятор](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) *Java* ([javac](https://ru.wikipedia.org/wiki/Javac)), стандартные библиотеки классов *Java*, примеры, документацию, различные утилиты и исполнительную систему *Java*([*JRE*](https://ru.wikipedia.org/wiki/JRE)). В состав *JDK* не входит [интегрированная среда разработки](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8) на *Java*, поэтому разработчик, использующий только*JDK*, вынужден использовать внешний текстовый редактор и компилировать свои программы, используя утилиты[командной строки](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D0%BE%D0%B9_%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%B8). | Для чего используется JDK? |
| **Java Runtime Environment** (сокр. **JRE**; *русск.* среда выполнения для Java) — минимальная реализация [виртуальной машины](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0), необходимая для исполнения [Java](https://ru.wikipedia.org/wiki/Java)-приложений, без компилятора и других средств разработки. Состоит из виртуальной машины — [Java Virtual Machine](https://ru.wikipedia.org/wiki/Java_Virtual_Machine) — и [библиотеки Java-классов](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0).  JRE распространяется свободно и для большинства платформ может быть загружена с сайта Oracle[[1]](https://ru.wikipedia.org/wiki/Java_Runtime_Environment#cite_note-1).  Средства разработки вместе с JRE входят в [JDK](https://ru.wikipedia.org/wiki/JDK). | Для чего используется JRE? |
| **Java Virtual Machine** (сокращенно **Java VM**, **JVM**) — [виртуальная машина](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0) Java — основная часть исполняющей системы [Java](https://ru.wikipedia.org/wiki/Java), так называемой *Java Runtime Environment* ([JRE](https://ru.wikipedia.org/wiki/JRE)). Виртуальная машина Java исполняет [байт-код Java](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82-%D0%BA%D0%BE%D0%B4_Java), предварительно созданный из [исходного текста](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D1%82%D0%B5%D0%BA%D1%81%D1%82) Java-программы[компилятором](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) Java ([javac](https://ru.wikipedia.org/wiki/Javac)). JVM может также использоваться для выполнения программ, написанных на других языках программирования. Например, исходный код на языке [Ada](https://ru.wikipedia.org/wiki/%D0%90%D0%B4%D0%B0_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) может быть откомпилирован в байт-код Java, который затем может выполниться с помощью JVM.  JVM является ключевым компонентом платформы Java. Так как виртуальные машины Java доступны для многих аппаратных и программных платформ, Java может рассматриваться и как связующее программное обеспечение, и как самостоятельная платформа. Использование одного [байт-кода](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82-%D0%BA%D0%BE%D0%B4) для многих платформ позволяет описать Java как «скомпилировано однажды, запускается везде» (compile once, run anywhere).  Виртуальные машины Java обычно содержат [Интерпретатор](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B0%D1%82%D0%BE%D1%80) байт-кода, однако, для повышения производительности во многих машинах также применяется [JIT](https://ru.wikipedia.org/wiki/JIT)-компиляция часто исполняемых фрагментов байт-кода в машинный код. | Для чего используется VM? |
| Простые типы в Java не являются объектно-ориентированными, они аналогичны простым типам большинства традиционных языков программирования. В Java имеется восемь простых типов: — byte, short, int, long, char, float, double и boolean. Их можно разделить на четыре группы:   1. Целые. К ним относятся типы byte, short, int и long. Эти типы предназначены для целых чисел со знаком. 2. Типы с плавающей точкой — float и double. Они служат для представления чисел, имеющих дробную часть. 3. Символьный тип char. Этот тип предназначен для представления элементов из таблицы символов, например, букв или цифр. 4. Логический тип boolean. Это специальный тип, используемый для представления логических величин. | Расскажите про базовые типы |
| Создавать объект простой переменной с помощью new недостаточно эффективно, так как new перемещает объект в кучу. Вместо этого создается «автоматическая» переменная, которая не является ссылкой на объект. Переменная хранит единственное значение и располагается в стеке. Стек — это область хранения данных, расположена в RAM. Процессор имеет прямой доступ до этой области через указатель на стек, поэтому стек — очень быстрый и эффективный способ хранения данных. По скорости стек уступает только регистрам (логично, так как регистры расположены внутри процессора). Все размеры примитивных типов строго фиксированы и не зависят от машинной архитектуры. Это одна с причин улучшенной переносимости Java-программ. | Что такое примитивные типы? |
| Несмотря на то что примитивные типы обеспечивают выигрыш производительности, бывают случаи, когда вам может понадобиться объектное представление. Например, вы не можете передать в метод примитивный тип по ссылке. Кроме того, многие из стандартных структур данных, реализованных в Java, оперируют с объектами, что означает, что вы не можете применять эти структуры данных для сохранения примитивных типов. Чтобы справиться с такими (и подобными) ситуациями, Java предлагает обертки примитивных типов, которые представляют собой классы, помещающие примитивный тип в объект.  Обертки для примитивных типов — это **Double**, **Float**, **Long**, **Integer**, **Short**, **Byte**, **Character** и **Boolean**. Эти классы предоставляют широкий диапазон ***методов***, позволяющий в полной мере интегрировать примитивные типы в иерархию объектных типов Java. | Что такое классы обертки? |
| Автоупаковка (autoboxing) — это процесс автоматической инкапсуляции данных простого типа, такого как int или double, в эквивалентную ему оболочку типа, как только понадобится объект этого типа. При этом нет необходимости в явном создании объекта нужного типа. Автораспаковка (auto-unboxing) — это процесс автоматического извлечения из упакованного объекта значения, когда оно потребуется. Вызовы методов, таких как intValue() и doubleValue(), становятся ненужными.  Integer iOb = 100;  int i = iOb; // автораспаковка | Что такое автобоксинг и анбоксинг? |
| Передача по ссылке подразумевает передачу ссылки на объект. В этом случае реализация метода потенциально может модифицировать переданный объект (например, вызвав метод, изменяющий состояние объекта).  В случае передачи по значению параметр копируется. Изменение параметра не будет заметно на вызывающей стороне.  *В Java параметры передаются по значению. В случае с объектами в метод передается ссылка на объект, но опять же по значению.* | Как в Java передаются параметры в методы? |
| Адрес ссылки на объект - это значение, которое копируется и передается в метод! Вы же не можете изменить ссылку, которую передали в метод в самом методе - это два разных числа. | Почему нельзя изменить ссылку на объект в методе? |
| У платформы java имеется две особенности. Для обеспечения кроссплатформенности программа сначала компилируется в промежуточный язык низкого уровня — байт-код. Вторая особенность загрузка исполняемых классов происходит с помощью расширяемых classloader. Это механизм обеспечивает большую гибкость и позволяет модифицировать исполняемый код при загрузке, создавать и подгружать новые классы во время выполнения программы.  Каждому классу в java соответствует один откомпилированный файл. Это справедливо даже для подклассов или анонимным классов. Такой файл содержит информацию об имени класса, его родителе, список интерфейсов которые он реализует, перечисление его полей и методов. Важно отметить, что после компиляции информации, которая содержит директива import, теряется и все классы именуются теперь через полный путь. Например в место String будет записано java/lang/String. | Описать механизм создания байт кода. |
| Иногда, в зависимости от условия, нужно выполнить различные действия. Для этого используется оператор if | Что такое оператор условия? |
| Single, if else, if else if else, ternal, switch | Какие типы оператороа условия существуют? |
| Ссылочные типы хранят не значение, а ссылку на него. Таким образом, если вы сравните две переменные ссылочного типа через операцию ==, то вы сравните лишь их адреса: | Почему нельзя сравнивать ссылочные типы по оператору ==? |
|  | Почему примитивные типы можно сравнивать через оператор ==? |
| Можно использовать простые типы **byte**, **short**, **char**, **int**. Также можно использовать **Enum** и **String** (начиная с JDK7), а также специальные классы, которые являются обёрткой для примитивных типов: Character, Byte, Short, Integer.  Дублирование значений **case** не допускается. Тип каждого значения должен быть совместим с типом выражения.  Команду **switch** часто называют командой выбора. Выбор осуществляется в зависимости от целочисленного выражения. Форма команды выглядит так:  switch(ВыражениеДляСравнения) { case Совпадение1: команда;break;case Совпадение2: команда;break;case Совпадение3: команда;break;default: оператор; break;} | Какие объекты могут быть использованы в операторе switch? |
| Существуют четыре логических оператора - &, &&, | и ||.  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. | Расскажите про булевы операции || && ? |
| Вопросительный знак – единственный оператор, у которого есть аж три аргумента, в то время как у обычных операторов их один-два. Поэтому его называют *«тернарный оператор»*.от [лат.](https://ru.wikipedia.org/wiki/%D0%9B%D0%B0%D1%82%D0%B8%D0%BD%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *ternarius* — «тройной» | Что такое тернарное условие? |
| **Java** операторы **цикла** (for, while, do-while), оператор break. **Цикл** в программировании используется для многократного повторения определенного фрагмента кода. В **Java** существует 3 оператора **цикла**: for, while, do-while. | Что такое циклы и для чего они используются? |
| тип «n-раз» предназначен для повторения каких-то действий заранее известное количество раз.  Оператор for содержит три параметра. Первый называется инициализацией, второй — условием повторения, третий — итерацией. | Для чего используется цикл for? |
| Конструкция foreach не требует ручного изменения переменной-шага для перебора - цикл автоматически выполняет эту работу.  Кроме того при таком способе исключается возможность ошибок выхода за пределы массива. | Для чего используется цикл foreach? |
| тип «пока» предназначен для повторения какого-то действия до тех пор, пока выполняется некоторое условие  Условие, определяющее будет ли цикл повторятся снова, проверяется перед каждым шагом цикла, в том числе перед самым первым. Говорят, что происходит *предпроверка* условия. | Для чего используется цикл while? |
| Бывает цикл типа «пока» с *постпроверкой* условия. Для его записи используется конструкция из операторов do…while. | Для чего используется цикл do while? |
| Массив — это конечная последовательность упорядоченных элементов одного типа, доступ к каждому элементу в которой осуществляется по его индексу. | Что такое массив? |
| Возможные следующие варианты объявления массива:  тип[] имя; тип имя[]; | Как создать массив? |
| создать (или как ещё говорят инициализировать) массивы можно следующим образом:  int[] a = new int[10]; // массив из 10 элементов  int n = 5; double[] a = new double[n]; // Массив из 5 элементов  double a[] = {3, 1, 0, 5, 3}; // Массив из 6 элементов | Как присвоить значение ячейке массива? |
| for(int i = 0; i <= ar2.length - 1; i++) {  System.out.print(ar2[i] + " "); } | Как можно пройти по всем элементам массива? |
| По индексу | Как можно найти элемент в массиве? |
| Java жестко контролирует выход за пределы массива. При попытке обратиться к несуществующему элементу массива возникает IndexOutOfBoundException. | Что будет если обратить к несуществующей ячейки массива? |
| Можно только занулить значение | Как удалить ячейку в массиве? |
| для того, чтобы обратиться к элементу массива, нужно указать его имя и, затем, в квадратных скобках — индекс элемента. Элемент массива с конкретным индексом ведёт себя также, как и переменная. | Как отредактировать ячейку в массиве? |