Муниципального автономного общеобразовательного учреждения
«Средняя общеобразовательная школа №5
«Научно-технологический центр имени И.В. Мичурина»
(Структурное подразделение Центр цифрового образования детей
«IT-куб. Мичуринск»)

Методическая рекомендация по проведению
занятий
Направление «Java»

«Методы»

Разработала: Колотова А.С.

педагог дополнительного образования

Мичуринск, наукоград – 2022 год

Содержание

ВВЕДЕНИЕ

Данная методическая разработка представляет цикл междисциплинарных занятий технической направленности, реализуемых в рамках реализации дополнительной образовательной программы технической направленности «Java» Занятий носит поисковоисследовательский характер и направлено на решение различных задач по программированию в процессе разработки проектов в среде. 3. Занятия с воспитанниками проходят в группах по 10-12 человек. Возраст обучающихся – 12-18 лет.

Во время занятия происходит частая смена деятельности: восприятие материала на большом экране и на слух, участие в обсуждении поставленной задачи, работа с бумажными инструкциями, работа на компьютере, на экране которого размещена презентация, устное представление своего проекта. Каждый ребенок, участвующий в работе по выполнению предложенного задания, имеет возможность не только решить поставленную задачу, но и доработать творчески свой мини-проект, предложить свой план выполнения предложенного задания.

На занятиях используются различные формы и методы обучения: фронтальная беседа; опрос; мозговой штурм; самостоятельная работа; исследовательская работа. В процессе реализации занятий активно формируются и используются универсальные учебные действия: - регулятивные (организация воспитанниками своей учебноисследовательской и проектной деятельности посредством постановки целей, планирования и контроля, коррекции своих действий и оценки достигнутого результата), - познавательные (логические действия, действия постановки, решения математических задач и проблем, анализ и представление достижений), - коммуникативные (проявляются как умение строить продуктивное взаимодействие и сотрудничество со сверстниками и с учителем; участие в коллективном обсуждении проблем, умение услышать позицию другого и выразить свое отношение, представить свою позицию);

- знаково-символические (проявляются в способности представить учебный материал, описать свой мини-проект, работать с информацией на компьютере, выделять существенное и в итоге сформулировать обобщенные знания);

- личностные (понимание значимости решения поставленных задач, достижение осмысленности учебного материала, поисково-исследовательская мотивация, упорство в достижении поставленной цели, внимательность, аккуратность, дисциплинированность, усидчивость, что необходимо при решении задач по программированию).

Оборудование:

Моноблочное интерактивное устройство, экран, ноутбуки (компьютеры).

Цели: Ознакомить учащихся с языком программирования Java.

Задачи:

1. Расширить и углубить теоретические знания по программировании на языке Java;
2. Развить практические навыки;
3. Углубить представление у учащихся о практическом применении языка программирования C++
4. Способствовать развитию интереса у учащихся к предмету «Информатика».

Методы

Функция – часть программы, имеющая собственное имя. Это имя можно использовать в программе как команду (такая команда называется вызовом функции). При вызове функции выполняются команды, из которых она состоит. Вызов функции может возвращать значение (аналогично операции) и поэтому может использоваться в выражении наряду с операциями.

Функции используются в программировании, чтобы уменьшить его сложность:

1. Вместо того, чтобы писать непрерывную последовательность команд, в которой вскоре перестаешь ориентироваться, программу разбивают на подпрограммы, каждая из которых решает небольшую законченную задачу, а потом большая программа составляется из этих подпрограмм (этот прием называется декомпозицией).
2. Уменьшается общее количество кода, потому что, как правило, одна функция используется в программе несколько раз.
3. Написанная однажды и всесторонне проверенная функция, может быть включена в библиотеку функций и использоваться в других программах (при этом не надо вспоминать, как была запрограммирована эта функция, достаточно знать, что она делает). Существует множество полезных библиотек функций, которыми могут пользоваться все программисты, а некоторые библиотеки поставляются «в комплекте» с языком программирования (например, все, кто программировал на Паскале, пользовались библиотечной функцией writeln() для вывода на экран, а в Java для этих целей доступен метод System.out.println(), входящий в одну из стандартных библиотек).

Метод — это функция, являющаяся частью некоторого класса, которая может выполнять операции над данными этого класса. В языке Java вся программа состоит только из классов и функции могут описываться только внутри них. Именно поэтому все функции в языке Java являются методами.

Суть понятия метод рассматривается на следующем занятии. А пока мы можем использовать его как синоним знакомого (по другим языкам программирования) понятия функция.

Объявление метода

Для того, чтобы использовать в программе собственный метод, его необходимо объявить.

При объявлении метода необходимо указать тип значения, которое будет возвращено после выполнения метода в программу. Если значение возвращать не нужно, указывается ключевое слово **void**. Затем идет произвольный идентификатор — имя метода. После имени метода в круглых скобках указывается список параметров (может быть пустым), а затем — в фигурных скобках — команды, составляющие тело метода.

Параметры — это данные, которые нужны методу для работы. Например, метод, рисующий круг, должен получить радиус и координаты центра круга. (Можно, конечно, было бы разработать метод без параметров, который рисует круг единичного радиуса с центром в начале координат, но он был бы значительно менее полезен).

Описание каждого параметра аналогично объявлению переменной (тип, а затем идентификатор — имя параметра). Параметры перечисляются через запятую.

В теле метода, возвращающего значение, должна быть команда **return**, после которой через пробел указывается выражение соответствующего типа. Эта команда заканчивает работу метода и передает указанное выражение в качестве возвращаемого значения основной программе — в то место, откуда метод был вызван*.

В качестве примера создадим метод для вычисления суммы квадратов двух целых чисел. Как и в случае с программой, важно в первую очередь определить входные и выходные данные. Входные данные — это параметры метода. Выходные данные — это его возвращаемое значение.

Замечание: в объектно-ориентированной концепции акцент несколько иной. Методы могут получать данные для работы, обращаясь к атрибутам своего класса, а результат их работы может заключаться в изменении этих атрибутов. Однако в рамках данного раздела мы рассматриваем методы как классические функции — подпрограммы.

Входными данными для рассматриваемого метода будут, очевидно, два целых числа. Выходные данные (результат) — целое число, представляющее собой сумму их квадратов.

Простейший вариант метода будет выглядеть следующим образом:

```
long squearSum(int x, int y) {return x*x + y*y;}
```

Перед возвращаемым типом указывается одно или несколько ключевых слов модификаторов (которые будут изучены позднее). Нам необходимо добавить к объявлению метода squearSum() модификатор **static**, поскольку мы собираемся обращаться к нему из метода main(), имеющего такой модификатор.

Описание метода squearSum() должно находиться внутри того единственного класса, из которого состоит наша простая программа, но не внутри метода main(), а на одном уровне с ним. То есть:

```
package mainPack;public class MyClass {/*** @param args*/public static void main(String[] args) {// TODO Auto-generated method stub}static long squearSum(int x, int y) {return x*x + y*y;}}
```

В результате в классе MyClass теперь два метода, один из которых, main(), выполняется при старте программы. Чтобы выполнился второй метод, его необходимо вызвать.

Вызов метода

Чтобы вызвать метод из другого метода того же класса, необходимо указать его имя, а затем в скобках список фактических параметров (если метод не требует параметров, скобки все равно ставятся).

Например, мы можем обратиться к описанному нами методу squearSum(), передав ему в качестве параметров два целых числа 10 и 20 следующим образом:

System.out.println(squearSum(10,20)+1);

В консоль будет выведено число 501.

Обратите внимание, вызов метода используется как операция, которую можно комбинировать с другими операциями (в данном случае, суммой) в выражении.

Чтобы вызвать метод другого класса, необходимо иметь объект этого класса*. Имя метода указывается через точку после имени объекта.

Например, в классе String (строка) имеется метод length(), возвращающий длину строки. Обратиться к этому методу можно только через объект класса String (что вполне логично, метод вызывается для той строки, длину которой мы хотим узнать).

String S = "Привет"; // Создание объекта класса String, подробнее см. нижеint x = S.length(); // Вызов метода length() для объекта S. В результате x = 6

Подробнее о вызове методов других классов будет рассказано позже.

В дальнейшем, рассказывая о каких-либо методах, мы будем называть их не только именем, но и для наглядности указывать в скобках параметры метода, например charAt(int i). В результате становится легко объяснить назначение метода: «Метод charAt(int i) возвращает символ строки с индексом i». Тип возвращаемого значения мы будем указывать только при необходимости.

Массивы

Определения

Массив — это совокупность переменных одного типа, имеющих общее имя. Каждая такая переменная называется **элементом массива**. С каждым элементом связано целое число — **индекс**, который используется (вместе с именем массива) для обращения к нему.

Создание массива

Создание массива происходит в два этапа.

1. Объявление массива. На этом этапе создается переменная типа ссылка на массив, с помощью которой мы сможем потом к массиву обращаться. Для этого указывается тип элементов массива, затем квадратные скобки (они показывают, что мы имеем дело с массивом, а не с обычным типом данных) и идентификатор — имя массива. Идентификаторов может быть несколько (как и в случае с переменными простых типов).

Примеры:

int[] a; // Создается ссылка на массив типа int**double**[] b, c; // Создаются две ссылки на массивы типа double

2. Создание массива. Создать массив — значит выделить в памяти место, достаточное для хранения всех его элементов. Для этого надо указать длину массива — количество элементов в нем. Кроме того, переменная-ссылка, объявленная на предыдущем этапе, теперь будет «указывать» не в пустоту (в Java эта «пустота» называется **null**), а на конкретный массив, с элементами которого можно работать.

Массив создается операцией **new**, которая выделяет участок памяти и возвращает указатель на этот участок. После ключевого слова **new** должен быть указан тип данных массива и его длина в квадратных скобках:

a = new **int**[5]; // В памяти выделяется место под массив из пяти целочисленных элементов, переменная a будет указывать на этот массивb = new **double**[4]; // В памяти выделяется место под массив из четырех действительных элементов, на него указывает переменная b

При этом элементам массива присваиваются значения по умолчанию*. Можно сразу проинициализировать массив нужными значениями, если перечислить их через запятую в фигурных скобках (длина массива при этом не указывается):

c = **new double**[]{2.3, 1.02, 8}; // В памяти выделяется место под массив из трех действительных элементов, на него указывает переменная c, элементы массива сразу получают нужные значения

Работа с массивом

После объявления массива с ним можно работать. Например, присваивать значения элементам массива и вообще обращаться с ними как с обычными переменными. Для обращения к конкретному элементу необходимо указать имя переменной, указывающей на массив, и его индекс в квадратных скобках. Нумерация элементов массива начинается с нуля.

Примеры:

a[0] = 5; // Первому элементу массива a, присваивается значение 5a[3] = 17; // Четвертому элементу массива a, присваивается значение 17a[1] = a[0] – a[3]; // Второму элементу массива a присваивается значение -12

о быстром создании массивов и особенностях переменных типа "ссылка на массив"

Длину массива можно определить, используя запись:

идентификатор_массива.length

Например, a.length будет равняться 5.

Очень удобно перебирать все элементы массива в цикле типа **for**. При этом обычно используется следующая форма:

for (**int** i = 0; i < a.length; i++) {// здесь можно что-нибудь сделать с элементом a[i]}

Например, следующий код присваивает всем элементам массива b числа от 1 до 4 (поскольку в массиве b четыре элемента):

for (**int** i = 0; i < b.length; i++) {b[i] = i;}

За границы массива выходить нельзя. Т.е., если в массиве a пять элементов, то обращение к шестому или восьмому элементу приведет к ошибке (точнее, возбуждению исключения).

a[5] = 8; // Нельзя, в массиве a только 5 элементов: a[0], a[1], a[2], a[3], a[4]

о многомерных массивах

Работа со строками

Создание строк

Строки тоже являются переменными ссылочного типа, а точнее — ссылками на объекты одного из нескольких строковых классов Java. Мы рассмотрим класс String.

Самый распространенный способ создать строку — это организовать ссылку типа String на строку-константу:

String s = "Это строка"

Можно просто сначала объявить переменную (которая получит значение **null**), а потом заставить ее ссылаться на строку-константу, другую строку или воспользоваться командой **new**, чтобы явным образом выделить память для строки:

String s1, s2, s3; // Объявление трех переменных, которые пока не связаны ни с какой строкойs1 = "Да здравствует день танкиста"; // Переменная s1 теперь ссылается на область памяти, в которой хранится строка "Да здравствует день танкиста"s2 = s1; // Теперь обе переменные s1 и s2 ссылаются на одно и то же место памятиs3 = **new** String(); // s3 ссылается на место в памяти, где хранится пустая строка

Объединение (сцепление) строк

Сцепление строк производится командой +. Ей соответствует оператор +=, который часто бывает очень удобен:

String S = "Привет";String S1 = "мир";S += ", " + S1 + "!"; // Теперь S ссылается на строку “Привет, мир!”

Длина строки

Определить длину строки можно методом length():

int x = S.length(); // Переменная x получит значение 12

Обратите внимание, String является классом (подробнее классы рассматриваются на следующем занятии), а length() - его методом, и поэтому

указывается через точку после имени переменной. Аналогично записываются и другие методы класса String.

Получение отдельных символов строки

о способах получения отдельных символов строки

Метод charAt(int i) возвращает символ строки с индексом i. Индекс первого символа строки — 0 (т.е. символы строки индексируются (нумеруются) аналогично элементам массива. Например:

```
char ch = S.charAt(2); // Переменная ch будет иметь значение 'и'
```

Метод toCharArray() преобразует строку в массив символов:

```
char[] ch = S.toCharArray(); // ch будет иметь представлять собой массив {'П','р','и','в','е','т',' ',',',' ','м','и','р','!'}
```

Замена отдельного символа

Метод replace(int old, int new) возвращает новую строку, в которой все вхождения символа old заменены на символ new.

```
String SS = S.replace('и', 'ы'); // SS = "Прывет, мыр!"
```

Получение подстроки

Метод substring(int begin, int end) возвращает фрагмент исходной строки от символа с индексом begin до символа с индексом end-1 включительно. Если не указывать end, будет возвращен фрагмент исходной строки, начиная с символа с индексом begin и до конца:

```
String S2 = S.substring(4,7); // S2 = "ет,"S2 = S.substring(4); // S2 = "ет, мир!"
```

Разбиение строки на подстроки

Метод split(String regExp) разбивает строку на фрагменты, используя в качестве разделителей символы, входящие в параметр regExp, и возвращает ссылку на массив, составленный из этих фрагментов. Сами разделители ни в одну подстроку не входят.

```
String parts[] = S.split(" "); // Разбили строку S на отдельные слова, используя пробел в качестве разделителя, в результате получили массив parts, где parts[0] = "Привет,", а parts[1] = "мир!"String parts[] = S.split(" и"); // Разбили
```

строку S на отдельные слова, используя в качестве разделителя пробел и букву и, в результате parts[0] = "Пр", parts[1] = "вет,", parts[2] = "м", parts[3] = "р!"

Сравнение строк

Если сравнивать строки, используя логическую операцию ==, то ее результатом будет **true** только в том случае, если строковые переменные указывают (ссылаются) на один и тот же объект в памяти.

Если же необходимо проверить две строки на совпадение, следует использовать стандартный метод equals(Object obj). Он возвращает **true**, если две строки являются полностью идентичными вплоть до регистра букв, и **false** в противном случае. Его следует использовать следующим образом:

S1.equals(S2); // Вернет true, если строки S1 и S2 идентичныS2.equals(S1); // Абсолютно то же самое**boolean** b = S.equals("Привет, мир!"); // b = true

Метод equalsIgnoreCase(Object obj) работает аналогично, но строки, записанные в разных регистрах, считает совпадающими.

Поиск подстроки

Метод indexOf(int ch) возвращает индекс первого вхождения символа ch в исходную строку. Если задействовать этот метод в форме indexOf(int ch, int i), то есть указать два параметра при вызове, то поиск вхождения начнется с символа с индексом i. Если такого символа в строке нет, результатом будет -1.

int pos = S.indexOf('в'); // pos = 3pos = "Вася".indexOf('с'); // pos = 2pos = "Корова".indexOf('о', 2); // pos = 3pos = "Корова".indexOf('К', 2); // pos = -1, поиск ведется с учетом регистра

Последнее вхождение символа можно найти с помощью метода lastIndexOf(int ch) или lastIndexOf(int ch, int i), который работает аналогично, но просматривает строку с конца.

У всех перечисленных методов есть одноименные варианты, которые принимают в качестве параметра строку вместо символа и проверяют, содержится ли эта строка в исходной строке.

pos = "Корова".indexOf("ор"); // pos = 1pos = "Барабанщик барабанил в барабан".indexOf("барабан", 5); // pos = 11pos = "Корова".indexOf("Вася"); // pos = -1

Изменение регистра символов в строке

Метод toLowerCase() возвращает новую строку, в которой все буквы сделаны строчными. Метод toUpperCase() возвращает новую строку, в которой все буквы сделаны прописными.

S = S.toUpperCase(); // S = "ПРИВЕТ, МИР!"

Заголовок метода main()

Теперь мы можем, наконец, понять большую часть описания метода main() (за исключением ключевых слов **public** и **static**.

Заголовок **public static void** main(String[] args) означает, что метод main() не возвращает значения (и действительно, мы ни разу не использовали в его теле команду **return**), а в качестве единственного параметра принимает массив строк args.

В качестве параметра args методу main() передаются так называемые аргументы командной строки. Дело в том, что каждую программу можно запустить не просто щелкнув мышкой по ее значку. Можно ввести имя исполняемого файла программы в командной строке (нажмите комбинацию Windows + R, чтобы увидеть командную строку Windows, если вы работаете в этой операционной системе), а после имени через пробел указать один или несколько дополнительных параметров (аргументов командной строки).

Таким образом, в методе main() можно обратиться к элементам массива args и увидеть, какие дополнительные аргументы пользователь указал при запуске программу. Можно научить программу на некоторые из этих аргументов реагировать.

При этом необязательно запускать программу из командной строки. В диалоговом окне Run --> Run... есть вкладка (x) = Arguments, перейдя на которую, можно перечислить интересующие вас аргументы командной

строки и в зависимости от них протестировать реакцию программу. Если, конечно, это необходимо: большинство программ никак не реагирует на аргументы командной строки.

Список использованных источников

1. Вязовик Н.А. Программирование на Java.
2. Хабибуллин И.Ш. Самоучитель Java 2.