Ответы на вопрос (лабораторная 4)

1. **Назначение подпрограмм и методов**  
   Подпрограмма - это отдельная часть программы, имеющая имя и решающая свою отдельную задачу. Располагается подпрограмма в начале основной программы и может быть запущена (вызвана) из основной программы по указанию имени  
     
   Использование подпрограмм позволяет избежать дублирования кода, в случае если необходимо один и тот же код писать в разных местах программы. Библиотеки, которые импортируются в программу (например, математическая библиотека Math) состоят из подпрограмм, которые уже кем-то составлены. Программистам не нужно думать о том, какие алгоритмы в них реализованы, а просто применяют их, задумываясь только о том, что именно они делают. Это большая экономия времени. Нет нужды писать алгоритм, который уже был кем то написан. Каждая подпрограмма должна решать только одну задачу, либо только что-то вычислять, либо выводить какие-либо данные, либо делать что-то еще.  
     
   Подпрограмма может быть реализована в виде функции.  
     
   Метод - это последовательность действий, направленная на решение отдельной задачи.  
   Методы определяются всегда внутри классов.  
   В классе есть главный метод main. Он сразу запускается.  
   Можно создавать своим методы.

**public** — тип доступа (метод может вызываться из другого класса). Существуют и другие типы доступа, к примеру private (метод доступен только внутри класса) и protected (о нем мы будем говорить позже).  
**static** означает что метод статический, т.е. принадлежит классу, а не конкретному экземпляру. Мы можем вызвать этот метод из другого класса так: Main.do().  
(Не статические методы в Java используются чаще, чем статические методы. Эти методы могут принадлежать любому объекту, экземпляру класса, а не всему классу. Не статические методы могут получать доступ и изменять поля объекта.)  
**void** значит, что этот метод не возвращает значение. Методы могут возвращать значение в Java и оно должно быть определено при объявлении метода. Для выхода из метода можно использовать return  
  
(Кроме перегрузки существует также замещение, или переопределение методов (англ. overriding). Замещение происходит, когда класс потомок (подкласс) определяет некоторый метод, который уже есть в родительском классе(суперклассе), таким образом новый метод заменяет метод суперкласса. У нового метода подкласса должны быть те же параметры или сигнатура, тип возвращаемого результата, что и у метода родительского класса.)

1. **Формальные и фактические параметры:**  
   Формальные параметры – это те параметры, которые указаны в описании функции. Он указан в объявлении функции (также указан его тип: int, double …). Пример:  
   public int sum (int a, int b)  
   { return a+b;}  
   Фактические параметры – это те параметры, которые ПЕРЕДАЛИ при вызове функции. Пример: sum(5, 5) //5, 5 – фактические параметры
2. **Статический метод. Описание и понятие**  
   **static** означает что метод статический, т.е. принадлежит классу, а не конкретному экземпляру. Мы можем вызвать этот метод из другого класса так: Main.do().  
   Статический метод можно вызвать не создавая экземпляр класса:  
   public class Counter {   
   static int i;   
   public static void count() {   
   i++;   
   System.out.println("Текущее значение счётчика - " + count);   
    }   
   }  
   Counter.count(); //вызов метода без создания экземпляра класса.
3. **Тип возвращаемого значения.**  
   Если функция возвращает какое-то значение, то в её описании не пишут слово void, а указывают тип возвращаемого значения (int, double …)

public int someMethod() // Здесь возвращаемый тип — int (целое число).

1. **Модификаторы методов:**  
   Методы в Java могут быть объявлены как   
   **public** – класс, метод, конструктор, интерфейс и т.д. объявленные как public могут быть доступны из любого другого класса. Поэтому  
   **private** – методы, переменные и конструкторы, которые объявлены как private в Java могут быть доступны только в пределах самого объявленного класса.  
   **static** – (3 пункт).  
   **final** - не может быть переопределен в наследниках.  
   **abstract** - абстрактный метод может существовать в абстрактном классе. Метод состоит только из объявления в классе и не имеет тела: abstract void yourMethod(); реализация метода (внутренний код) может быть предоставлен в классах-наследниках.  
   **native** – если нужно вызвать подпрограмму, которая написана на другом языке, а не на Java, то используется модификатор **native.** он (метод) реализуется на платформозависимом коде, обычно написаном на другом языке программирования, например C или C++ и т.д. Единственное отличие методов native состоит в том, что они реализуются на другом языке программирования.  
   **synchronized** - может быть такое, что несколько методов обращаются к одним и тем же переменным. В этом случае используется **synchronized** для указания того, что метод может быть доступен только одним потоком одновременно (чтобы разделить потоки)

**strictfp** (читайте в конце статьи),

**(по-моему, это не про методы) transient** – указывает на сохранять переменную (т.е не переводить) в последовательно байт.  
**(по-моему, это не про методы) volatile** (пер: волатильность)– это значит что переменная может меняться. Этот модификатор гарантирует, что значение переменной volatile всегда будет считаться из основной памяти, а не из локального кэша Thread. Это означает, что каждое чтение изменчивой переменной будет считаться из основной памяти компьютера, а не из кэша ЦП, и что каждая запись в изменчивую переменную будет записана в основную память, а не только на кеш процессора.

1. **Перегрузка методов**  
   В языке Java в пределах одного класса можно определить два или более ме­тодов, которые совместно используют одно и то же имя, но имеют разное количество параметров. Когда это имеет место, методы называют перегру­женными, а о процессе говорят как о перегрузке метода
2. **Передача параметров в методы Java**  
   Данные передаются между методами через параметры. Есть два способа передачи параметров:
3. **Передача по значению (by value**). Значения фактических параметров копируются. Вызываемый метод создает свою копию значений аргументов и затем ее использует. Поскольку работа ведется с копией, на исходный параметр это никак не влияет.
4. **Передача по ссылке (by reference)**. Параметры передаются как ссылка (адрес) на исходную переменную. Вызываемый метод не создает свою копию, а ссылается на исходное значение. Следовательно, изменения, сделанные в вызываемом методе, также будут отражены в исходном значении.  
     
   **Java всегда передает параметры по значению**(Объекты передаются по ссылке, примитивы по значению. Массивы примитивов - это объекты, поэтому передаются также по ссылке. Обновлено: под передачей по ссылке я имею ввиду, что объект таки передается по значению. Но в значении содержится ссылка на объект, а не копия объекта.)  
   (Примитивные типы передаются по значению. Объекты передаются в метод по ссылке. Поэтому все изменения ссылочного объекта влияют на переданный объект. Если переменная примитивный тип, то передается только значение переменной, то есть копия этого значения.)
5. **Отличия функции и процедуры.**  
   Функция - подпрограмма, выполняющая какие-либо операции. **(Она самостоятелна)**

Метод - это функция или процедура, которая принадлежит классу или экземпляру класса. **(Он принадлежит классу)**

1. **Рекурсия. Механизм работы рекурсии.**Рекурсивные функции в java – это функции, которые вызывают сами себя.  
   public void recursia() {   
   System.out.println("Привет, JavaRush!");  
   recursia();   
   }  
   (Есть еще два важных для понимания рекурсии определения:   
   Базис рекурсии – условие выхода из блока рекурсивных вызовов – базисное решение задачи, при условиях, когда нет необходимости вызывать рекурсию.   
   Шаг рекурсии – вызов функцией самой себя при изменении параметров)
2. **Стек вызова**Стек работает по схеме LIFO (последним вошел, первым вышел). Всякий раз, когда вызывается новый метод, содержащий примитивные значения или ссылки на объекты, то на вершине стека под них выделяется блок памяти.  
   Когда метод завершает выполнение, блок памяти (frame), отведенный для его нужд, очищается, и пространство становится доступным для следующего метода. При этом поток выполнения программы возвращается к месту вызова этого метода с последующим переходом к следующей строке кода.  
     
   Помимо того, что мы рассмотрели, существуют и другие особенности стека:

* Он заполняется и освобождается по мере вызова и завершения новых методов
* Переменные в стеке существуют до тех пор, пока выполняется метод в котором они были созданы
* Если память стека будет заполнена, Java бросит исключение java.lang.StackOverFlowError
* Доступ к этой области памяти осуществляется быстрее, чем к куче
* Является потокобезопасным, поскольку для каждого потока создается свой отдельный стек

**Структура описания рекурсивных подпрограмм (методов)**public void recursia() {   
System.out.println("Привет, JavaRush!");  
recursia();   
}  
В конце подпрограммы вызывается она же. Можно её вызвать при определённых условиях  
 **12. Доступ к статическому содержимому классов  
Статическая переменная** - принадлежит классу, а не объекту (экземпляру). **Статический метод** можно вызвать не создавая экземпляр класса:  
**Статический блок** представляет собой блок утверждения внутри класса Java , который будет выполняться , когда класс впервые загружается в JVM. **Статический блок помогает инициализировать статические элементы данных** , так же , как конструкторы помогают инициализировать член экземпляра.

class Test{

static {

//Code goes here

}

}

**Пример: как получить доступ к статическому блоку**

public class Demo {

static int a;

static int b;

static {

a = 10;

b = 20;

}

public static void main(String args[]) {

System.out.println("Value of a = " + a);

System.out.println("Value of b = " + b);

}

}

вы получите следующий вывод программы.

Value of a = 10

Value of b = 20