Лабораторные работы по курсу программирование для мобильных приложений

# Перед началом работы…

## Установка компонентов среды разработки

Приложения для Android, как и большинство приложений для коммуникаторов, разрабатываются на стандартном ПК, где ресурсы не ограничены (по сравнению с мобильным устройством) и загружаются на целевой коммуникатор для отладки, тестирования и последующего использования. Приложения можно отлаживать и тестировать на реальном устройстве под управлением Android или на эмуляторе. Для первоначальной разработки и отладки удобнее использовать эмулятор, а затем выполнять окончательное тестирование на реальных устройствах. Разработчикам предоставляется возможность использовать средства разработки приложений для Android на ПК под управлением любой из распространенных операционных систем: Windows, Linux и Mac OS X. Ниже будет детально описан процесс установки компонентов среды разработки под Windows XP, для прочих операционных систем отличия весьма незначительны.

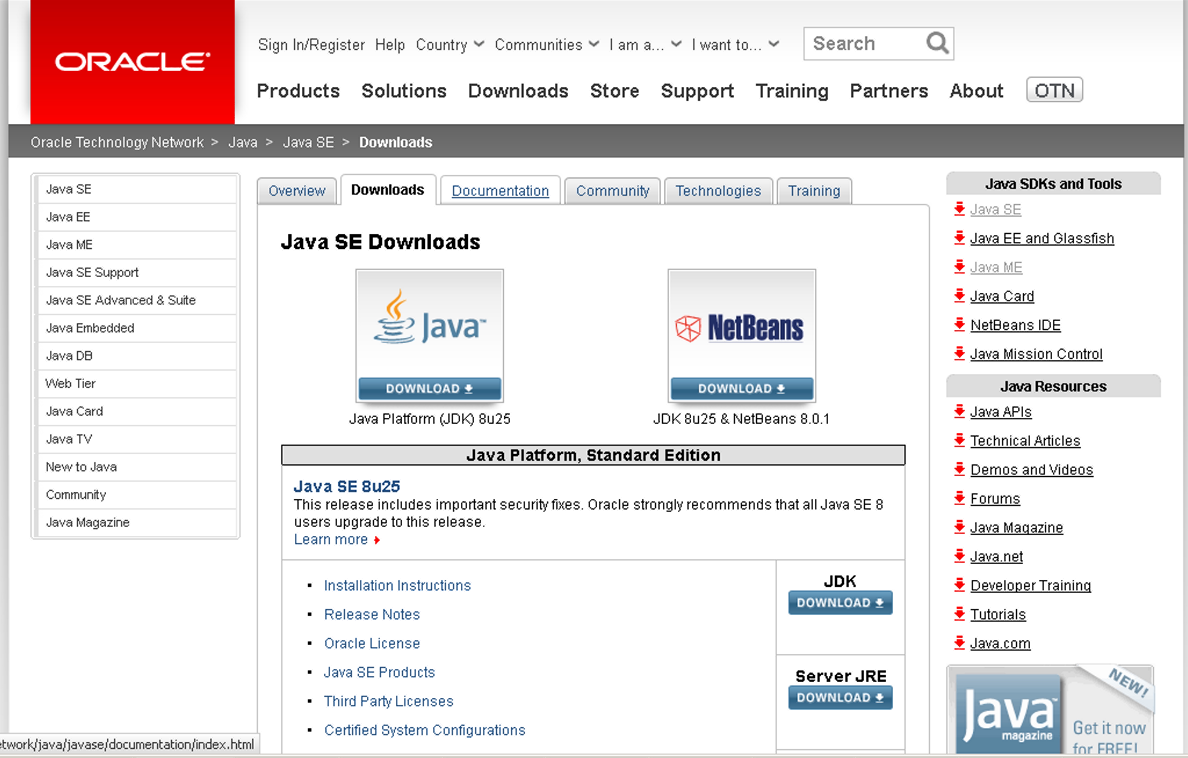
**Но мы с вами начнем программировать на языке Java. А для программирования под Android вам придется (потом) выполнить еще несколько шагов после установки JDK, JRE и NetBeans.**

## Установка JDK

Для установки **NetBeans** (а затем и Android SDK) требуется JDK версии не ниже 7 (на момент написания данного методического руководства была актуальна 8-я версия Sun JDK. Для Windows традиционно используется Sun JDK, который можно бесплатно загрузить на сайте разработчика:

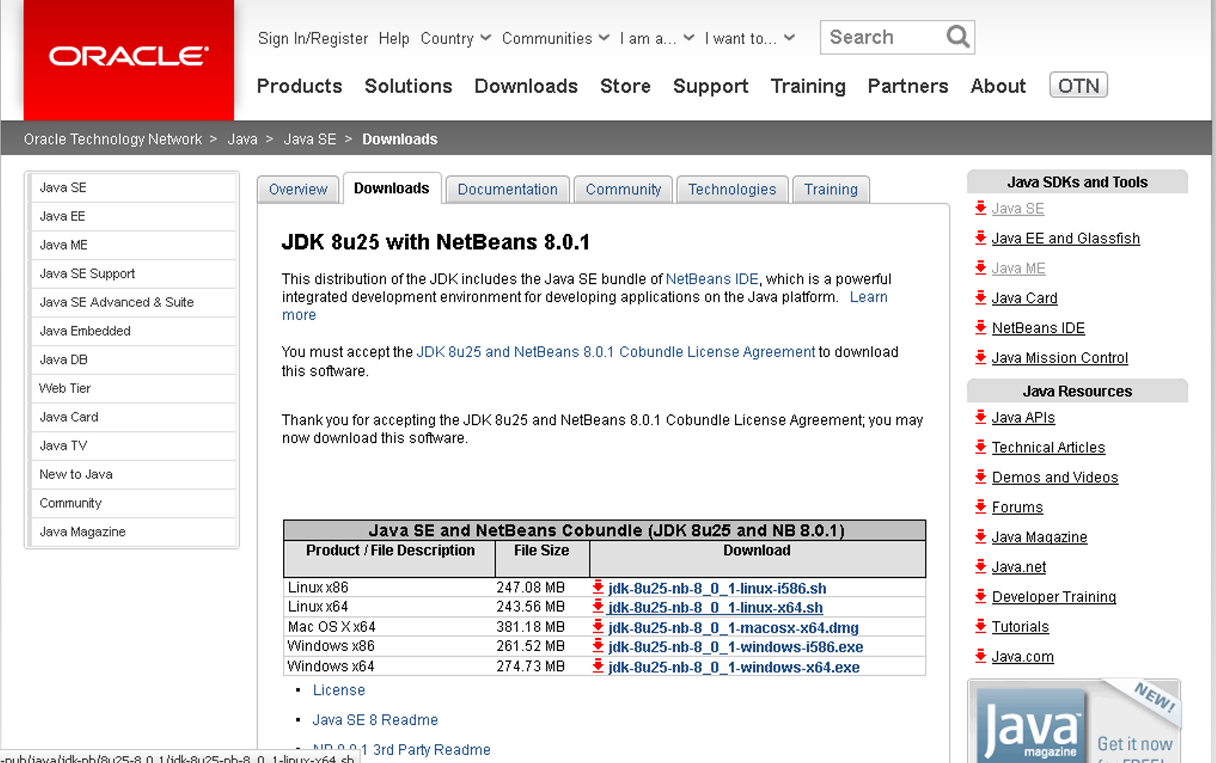
Страница загрузки Sun JDK

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>



Установим последнюю версию **NetBeans**. Вам будут предложены несколько вариантов сборок IDE для скачивания, в контексте данной главы рассмотрим установку Java-сборки среды. Любые модули можно добавить или удалить впоследствии.

Выберите язык пользовательского интерфейса, вашу операционную систему и установщик с компонентами, которые вас интересуют.



Дальше все как обычно: выбираем путь для установки JRE (как правило это: C:\Program Files\Java\jre7). Путь для установки ***JRE (***Java Runtime Environment***)*** (как правило это: ***C:\Program Files\Java\jre7***). Путь для установки ***JDK*** (по умолчанию: ***C:\Program Files\Java\jdk1. jdk1.8.0\_20***).

NetBeans устанавливается по умолчанию в директорию C:\Program Files\ NetBeans № Версии., выбрав предварительно необходимые компоненты.

**ВНИМАНИЕ!** Для выполнения ЛР1 вам необходимо прописать переменную окружения. Делается это так:

в переменную окружения ***PATH*** операционной системы вносится ***путь к каталогу BIN JDK***, например, ***С:\Program Files\Java\jdk1.8.0\_60\bin***. Для того, чтобы добраться до переменной окружении нужно зайти в ***"Панель управления"-->Система-->Дополнительно-->Переменные -->Добавить***.

Добавляем переменную ***JAVA\_HOME***, устанавливаем ее в значение ***C:\Program Files\Java\jdk1.8.0\_60.***

ЛАБОРАТОРНАЯ РАБОТА №1 *(4 часа)*

## 1.1. Разработка простой программы на Java.

## Работа в командной строке - компиляция и запуск на выполнение.

**Цель.** Освоить основы работы в командной строке Java. Изучить синтаксис простой программы. Посмотреть возможности языка.

1.1.1. Создание программы на Java в текстовом редакторе

Xод работы:

1. Установка JDK.

2. Установке IDE NetBeans.

3. Системная переменная JAVA\_HOME.

***Пример 1.1.***

В любом текстовом редакторе создаем файл JavaApplication1.java (имя файла должно совпадать с именем класса):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | **public class JavaApplication1 {**  **public static void main(String[] args) {**  **System.out.println("Hello World!");**  **Calculator calc=new Calculator();**  **System.out.println("2+2="+calc.sum(2,2));**  **}**  **public static class Adder**  **{**  **private int sum;**  **public Adder() { sum=0; }**  **public Adder(int a){ this.sum=a;}**  **public void add(int b)**  **{**  **sum+=b;**  **}**  **public int getSum() { return sum;}**  **}**  **public static class Calculator**  **{**  **public int sum(int... a)**  **{**  **Adder adder=new Adder();**  **for(int i:a)**  **{**  **adder.add(i);**  **}**  **return adder.getSum();**  **}**  **}**  **}** |

## 1.1.2. Компиляция программы на Java в командной строке

4.1. Компиляция в командной строке. Заходим в директорию, где хранится файл, и выполняем компиляцию:

**javac JavaApplication1.java**

В результате получаем скомпилированные файлы JavaApplication1.class, JavaApplication1$Adder.class, JavaApplication1$Calculator.class

## 1.1.3. Запуск из командной строки

**java -classpath . JavaApplication1**

Поучаем:

**"Hello World!"**

**2+2=4**

## 1.1.4. Дополнительные примеры

**Пример 1.2. Примитивное окно приложений с обработкой события Close.**

Можем ли мы создать полноценное окно (как в Windows) из программы, написанной на языке Java? Для этого существует класс Frame (рамка). Если ваш класс является потомком класса Frame, то вы получаете окно со всеми его атрибутами. Запуск на выполнение потомка класса Frame происходит из функции main. Никакого кода HTML не требуется. Программа полностью автономна и браузер для нее не нужен.

import java.awt.\*;

import java.awt.event.\*;

class SimpleFrame extends Frame{

SimpleFrame(String s){

super(s);

setSize(400, 150);

setVisible(true);

addWindowListener(new WindowAdapter(){

public void windowClosing(WindowEvent ev){

dispose();

System.exit(0);

}

});

}

public static void main(String[] args){

new SimpleFrame(" Моя программа");

}

}

Кнопка закрытия окна работать не будет, если не написать специальный код, который присутствует в приведенной выше программе (код располагается в конструкторе):

addWindowListener(

new WindowAdapter()

{ public void windowClosing(WindowEvent ev)

{

//dispose();

System.exit(0);

}

});

Это пример **определения безымянного класса**. Рассмотрим его подробнее. Мы вызываем метод **addWindowListener** для того, чтобы назначить слушателя оконных событий. В качестве параметра создаем объект класса **WindowAdapter**. Но этот класс является абстрактным! Поэтому мы неявно создаем производный от него класс и переопределяем нужные нам методы - в данном случае обработку события закрытия окна (метод **dispose** уничтожает объект **Frame**). После этого останавливаем виртуальную машину **Java** вызовом метода **System.exit(0)**. При компиляции будет создан класс с именем **simpleFrame$1.class**

**Пример 1.3. Окно приложений с выводом текста**

import java.awt.\*;

import java.awt.event.\*;

class HelloWorldFrame extends Frame{

HelloWorldFrame(String s){

super(s);

}

**public void paint(Graphics g){**

**g.setFont(new Font("Serif", Font.ITALIC | Font.BOLD, 30));**

**g.drawString("Hello, XXI century World!", 20, 100);**

}

public static void main(String[] args){

**Frame f = new HelloWorldFrame("Здравствуй, мир XXI века!");**

**f.setSize(400, 150);**

**f.setVisible(true);**

f.addWindowListener(new WindowAdapter(){

public void windowClosing(WindowEvent ev){

//dispose();

System.exit(0);

}

});

}

}

**Пример 1.4. Окно приложений c графическим примитивом линии**

import java.awt.\*;

import java.awt.event.\*;

class GraphTest01 extends Frame{

GraphTest01(String s) {

super(s);

setBounds(0, 0, 500, 300);

setVisible(true);

}

public void paint(Graphics g){

Dimension d = getSize();

int dx = d.width / 20, dy = d.height / 20;

int myWidth = 250, myHeight = 250;

g.drawLine(0, 0, myWidth, myHeight);

g.drawLine(0, 0, d.width, d.height);

setBackground(Color.blue);

setForeground(Color.red);

}

public static void main(String[] args){

GraphTest01 f = new GraphTest01(" Пример рисования");

f.addWindowListener(new WindowAdapter(){

public void windowClosing(WindowEvent ev){

System.exit(0);

}

});

}

}

**Справочная информация по классу Graphics**

**Класс Graphics** - работа с графикой осуществляется через методы paint(), update().

// Рисует линию.

void **drawLine**(int startX, int startY, int endX, int endY);

// Контурный прямоугольник.

void **drawRect**(int top, int left, int width, int height);

// Закрытый прямоугольник.

void **fillRect**(int top, int left, int width, int height);

// Контурный прямоугольник с закругленными углами.

void **drawRoundRect**(int top, int left, int width, int height,

int xDiam, int yDiam);

// Закрытый прямоугольник с закругленными углами.

void **fillRoundRect**(int top, int left, int width, int height,

int xDiam, int yDiam);

// Контурный эллипс внутри заданного прямоугольника

void **drawOval**(int top, int left, int width, int height);

// Закрытый эллипс внутри заданного прямоугольника

void **fillOval**(int top, int left, int width, int height);

// Контурная дуга

void **drawArc**(int top, int left, int width, int height,

int startAngle, int sweepAngle);

// Закрытая дуга

void **fillArc**(int top, int left, int width, int height,

int startAngle, int sweepAngle);

// Контурный многоугольник

void **drawPolygon**(int x[], int y[], int numPoints);

// Закрытый многоугольник

void **fillPolygon**(int x[], int y[], int numPoints);

1.1.5. Задания к лабораторной работе №1 (часть 1)

Задание 1.1:

1. Скомпилируйте и выполните примеры 1-4 из командной строки.
2. Используя справочную информацию по классу Graphics, поэкспериментируйте с графическими примитивами.
3. Сформулируйте выводы о структуре Java-программы и методах использования классов, включите их в отчет.
4. Сформируйте отчет с текстами программ и скринами результатов.

## 1.2. Работа с пакетами и Jar – архивами Java (часть 2)

## Цель. Освоить основы работы с пакетами и архивами jar.

**Мотвация:** Некоторые программные продукты Java вообще не работают с безымянным пакетом (которые создаются по умолчанию). Поэтому в технологии Java рекомендуется все классы помещать в пакеты.

1.2.1. Сведения о работе с пакетами

В стандартную библиотеку Java API входят сотни классов. Каждый программист в ходе

работы добавляет к ним десятки своих классов. Множество классов растет и становится необозримым. Уже давно принято отдельные классы, решающие какую-то одну определенную задачу, объединять в библиотеки классов. Но библиотеки классов, кроме стандартной библиотеки, не являются частью языка.

Разработчики Java включили в язык дополнительную конструкцию — пакеты (packages). Все классы Java распределяются по пакетам. Кроме классов пакеты могут содержать интерфейсы и вложенные подпакеты (subpackages). Образуется древовидная структура пакетов и подпакетов.

Эта структура в точности отображается на структуру файловой системы. Все файлы с расширением class (содержащие байт-коды), образующие один пакет, хранятся в одном каталоге файловой системы. Подпакеты образуют подкаталоги этого каталога.

Каждый пакет создает одно пространство имен (namespace). Это означает, что все имена классов, интерфейсов и подпакетов в пакете должны быть уникальны. Имена в разных пакетах могут совпадать, но это будут разные программные единицы. Таким образом, ни один класс, интерфейс или подпакет не может оказаться сразу в двух пакетах. Если надо в одном месте программы использовать два класса с одинаковыми именами из разных пакетов, то имя класса уточняется именем пакета: ***пакет.Класс.*** Такое уточненное имя называется полным именем класса (fully qualified name).

Если член класса не отмечен ни одним из модификаторов private, protected, public, то по умолчанию к нему осуществляется ***пакетный доступ*** ***(default access)***, т. е. **к такому члену может обратиться любой метод любого класса только из того же пакета**. Если **класс не помечен модификатором public**, **то все его члены, даже открытые, public, не будут видны из других пакетов.**

Члены с пакетным доступом не видны в подпакетах данного пакета.

1.2.2. Сведения о работе с архивами

### Для упаковки нескольких файлов в один архивный файл, со сжатием или без сжатия, в технологии Java разработан формат архивирования JAR. Имя архивного jar-файла может быть любым, но обычно оно получает расширение jar. Способ упаковки и сжатия основан на методе ZIP. Название JAR (Java ARchive) перекликается с названием известной утилиты TAR (Tape ARchive), разработанной в UNIX.

Отличие jar-файлов от zip-файлов только в том, что в jar-файлы автоматически включается каталог META-INF, содержащий несколько файлов с информацией об упакованных в архив файлах.

Архивные файлы очень удобно использовать в апплетах, поскольку весь архив загружается по сети сразу же, одним запросом. Все файлы апплета с байт-кодами, изображениями, звуковые файлы упаковываются в один или несколько архивов. Для их загрузки достаточно в теге <applet> указать имена архивов в параметре archive.

Архивные файлы удобно использовать и в приложениях (applications). Все файлы приложения упаковываются в архив, например appl.jar. Приложение выполняется прямо из архива, интерпретатор запускается с параметром –jar, например:

java –jar appl.jar

Имя основного класса приложения, содержащего метод main(), указывается в файле MANIFEST.MF, который автоматически создается при создании архива.

Jar-архивы создаются с помощью классов пакета java.util.jar или посредством утилиты командной строки jar.

## 1.2.3. Задания к лабораторной работе №1 (часть 2)

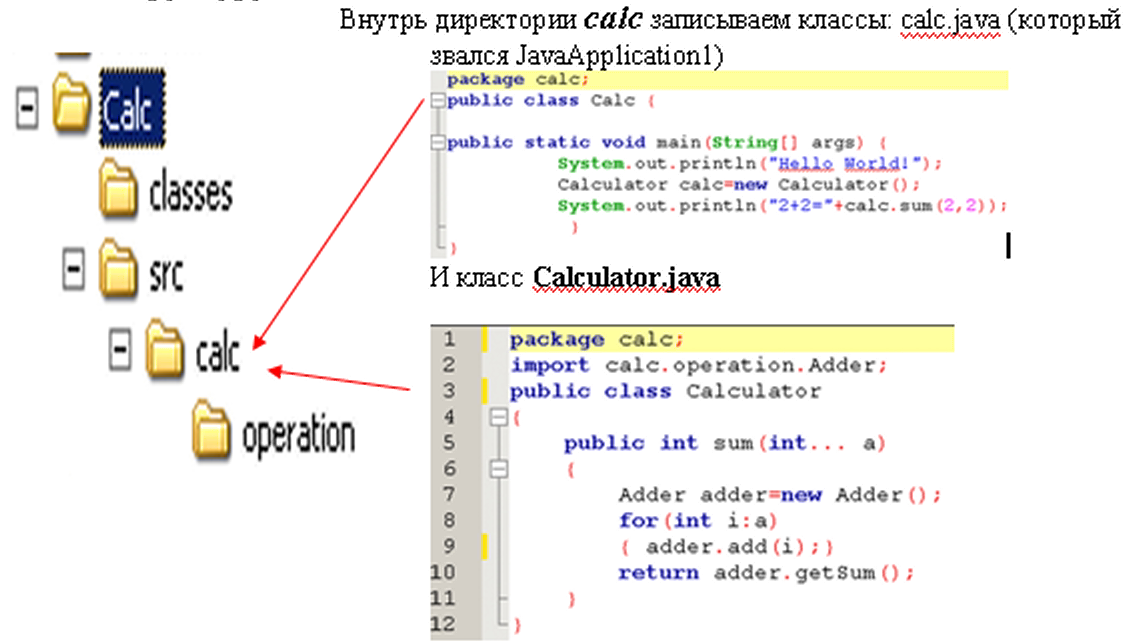
Задание 1.2:

Возьмите программу из первой части лабораторной работы 1 - **JavaApplication1.java (Пример 1.1.)** и разделите классы так, чтобы все они находились в разных пакетах.

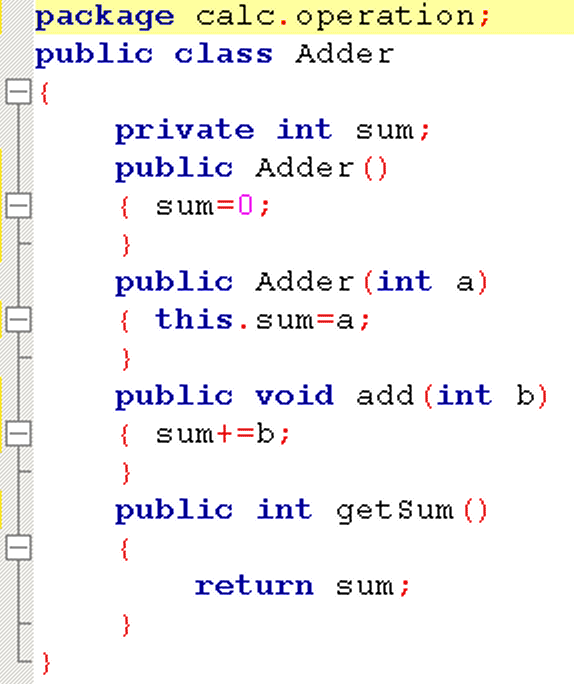
Соглашение "Code Conventions" рекомендует записывать имена пакетов строчными буквами. Тогда они не будут совпадать с именами классов, которые, по соглашению, начинаются с прописной буквы. Кроме того, соглашение советует использовать в качестве имени пакета или подпакета доменное имя своего сайта, записанное в обратном порядке, например: **com.sun.developer**

Это обеспечит уникальность имени пакета во всем Интернете.

**Создаем структуру:**

****

**! Обратите внимание на первые строки файлов, содержащие указание, в каком пакете находится класс, а также на строку *import*.**

В директорию operation определяем класс **Adder**:

Компилируем с указанием

**-sourcepath –** пути, где находятся файлы- источники - это директория **src**

**-d –** пути, куда положить классы – это директория **classes,**

- ну и конечно пути к главному классу: **src/calc/Calc.java:**

**…Calc>javac -sourcepath src -d**

**classes src/calc/Calc.java**

При этом директории ***calc*** и ***operation*** будут созданы компилятором внутри ***clssses*** автоматически.

Запускаем на выполнение так:

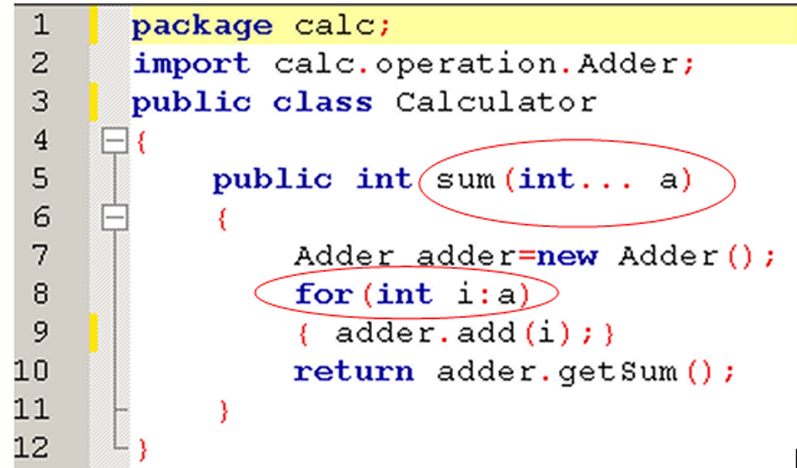
**Calc\classes>****java -classpath . calc/Calc**

Или так:

**Calc\classes>java -classpath . calc.Calc**

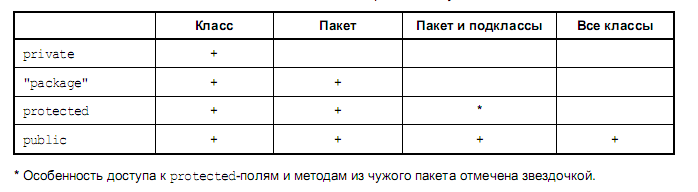
Здесь обращаю ваше внимание на два момента класса ***Calculator***:

**Попробуйте объяснить смысл обведенных фрагментов кода ниже:**

****

Для справки:

Таблица 2.1. Права доступа к полям и методам класса

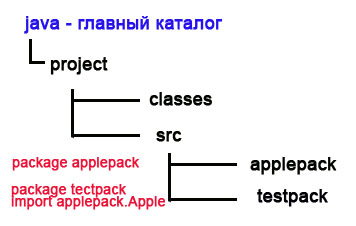


Задание 1.3:

***1.3.1. Разработайте:***

Класс ***Apple*** (в пакете ***src.applepack*** ) с методами ***getMass( )***, ***setMass( )***, которые предоставляют доступ к полю ***float mass*** – вес контейнера с яблоками.

Класс ***Test*** (в пакете ***src.testpack***) с методом ***main***, который создает экземпляр класса ***Apple***, должен инициализировать массу, а затем распечатать ее значение.

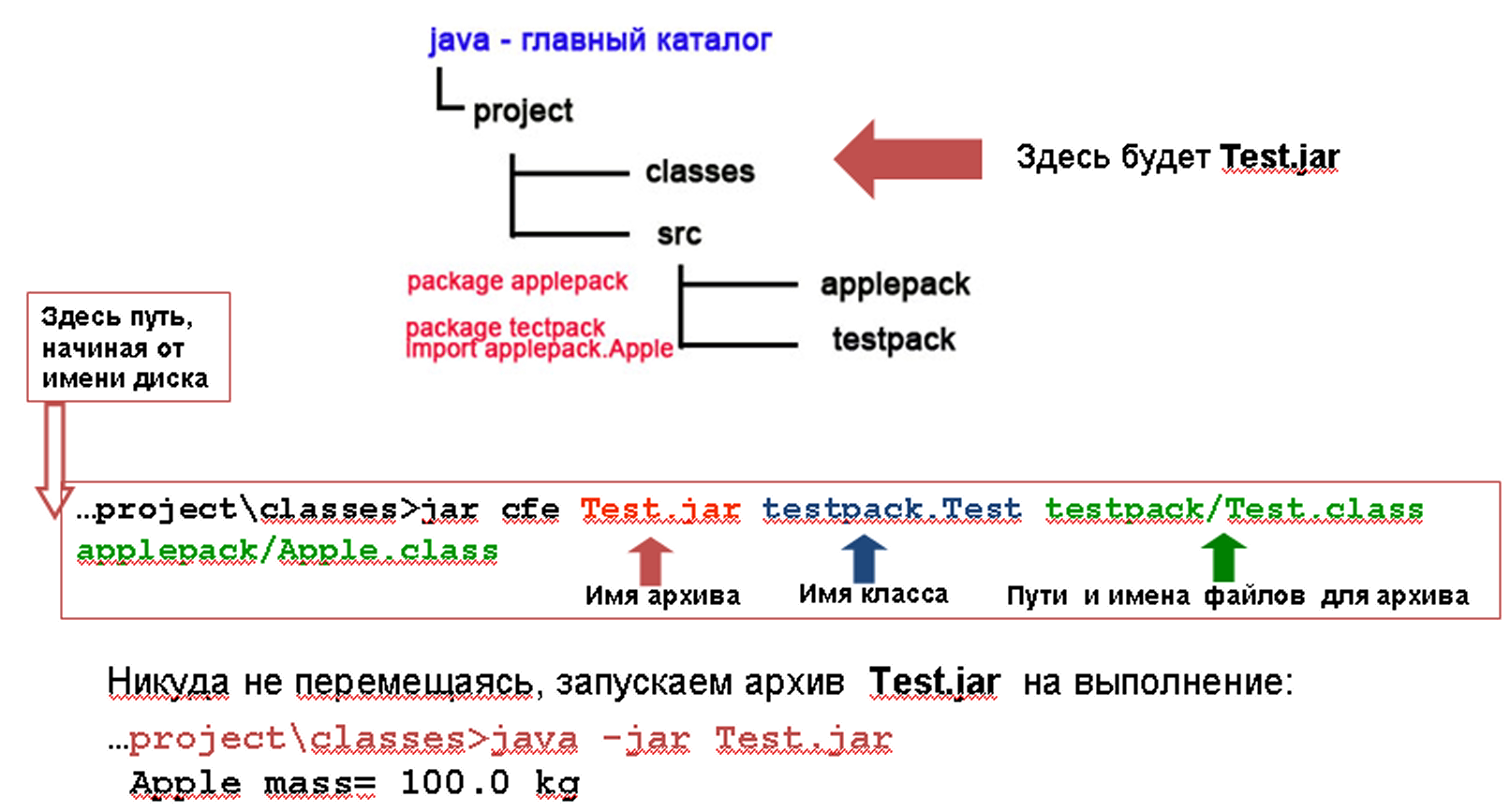
***1.3.2.Создайте структуру папок:***

***project\src\applepack // для класса Apple***

***project\src\testpack // для класса Test***

**Выполнить:**

1. Компиляцию классов с размещением в указанной папке classes. Исходные файлы находятся в каталоге ***project\src\***
2. Запуск приложения из папки ***classes***
3. Создать jar архив для всего проекта + запустить приложение из jar архива. (см. Лекцию 4 и пример создания архива ниже по тексту.)



Задание 1.4:

Доработайте пример 1.1. до калькулятора с основными математическими действиями: сложение (уже есть), вычитание, умножение, деление согласно варианту:

|  |  |  |
| --- | --- | --- |
| **Вариант** | **Тип переменной** | **Доп. операция** |
| 1 | Byte | Сдвиг влево на 1 |
| 2 | Short | Сдвиг вправо на 1 |
| 3 | Int | Возведение в квадрат |
| 4 | Long | Сдвиг влево на ***х*** позиций |
| 5 | Double | Сдвиг вправо на ***х*** позиций |
| 6 | Float | Остаток от деления |
| 7 | Iht | Инвертирование |
| 8 | Short | Логическое умножение |

## Требования к отчету к ЛР №1

1. Листинг к заданию 1.2.
2. Листинг к заданию 1.3. Скриншоты, демонстрирующие создание архива и его запуск.
3. Листнг к заданию 1.4.
4. Краткое описание функций пакетов и архивов.
5. Готовность ответить на вопросы по примерам из текста ЛР и домашних заданий.