Федеральное агентство по образованию

Государственноеобразовательноеучреждениевысшегопрофессиональногообразования«Нижегородскийгосударственныйуниверситет  
им. Н.И.Лобачевского»

Институт информационных технологий, математики и механики

Направление: Фундаментальная информатика и информационные технологии

**Отчёт по лабораторной работе по теме**

**«Калькулятор для Android»**

**Выполнил:** студент группы0846-2

Андреев В.С.

Подпись: \_\_\_\_\_\_\_\_

**Проверил:**

Борисов Н.А.

Подпись: \_\_\_\_\_\_\_\_

Нижний Новгород  
20 16

**Оглавление**

[Постановка задачи 3](#_Toc462956391)

[Описание работы программы 4](#_Toc462956392)

[Краткое описание 4](#_Toc462956393)

[Работа с программой на примере функции сложения. 5](#_Toc462956394)

[Листинги исходного кода программы 9](#_Toc462956395)

[Краткое описание 9](#_Toc462956396)

[User Interface часть 9](#_Toc462956397)

[Java часть 14](#_Toc462956398)

# Постановка задачи

1. Разработать программу "Калькулятор" в соответствии с указаниями в Лекциях 2.1-2.3. Программа должна выполнять одну операцию (сложение).

2. Самостоятельно доработать программу для выполнения всех четырех арифметических операций. Предусмотреть проверку корректности вводимых пользователем данных.

3. Продумать идею и предложить вариант реализации калькулятора с "памятью", позволяющего использовать ранее полученные результаты для дальнейших вычислений

# Описание работы программы

## Краткое описание

Программа калькулятор позволяет выполнять простейшие арифметические операции (сложение, умножение, деление, вычитание). Так же имеется возможность сохранять, и использовать в дальнейшем промежуточные результаты (выполняется кнопкой «MR+», которая меняет своё состояние на «MR-», если значение сохранено). Вы имеете возможность менять знак, и очищать экран, если это необходимо, так же есть возможность выполнять вычисления цепочкой (не нажимая на равно после каждой операции).

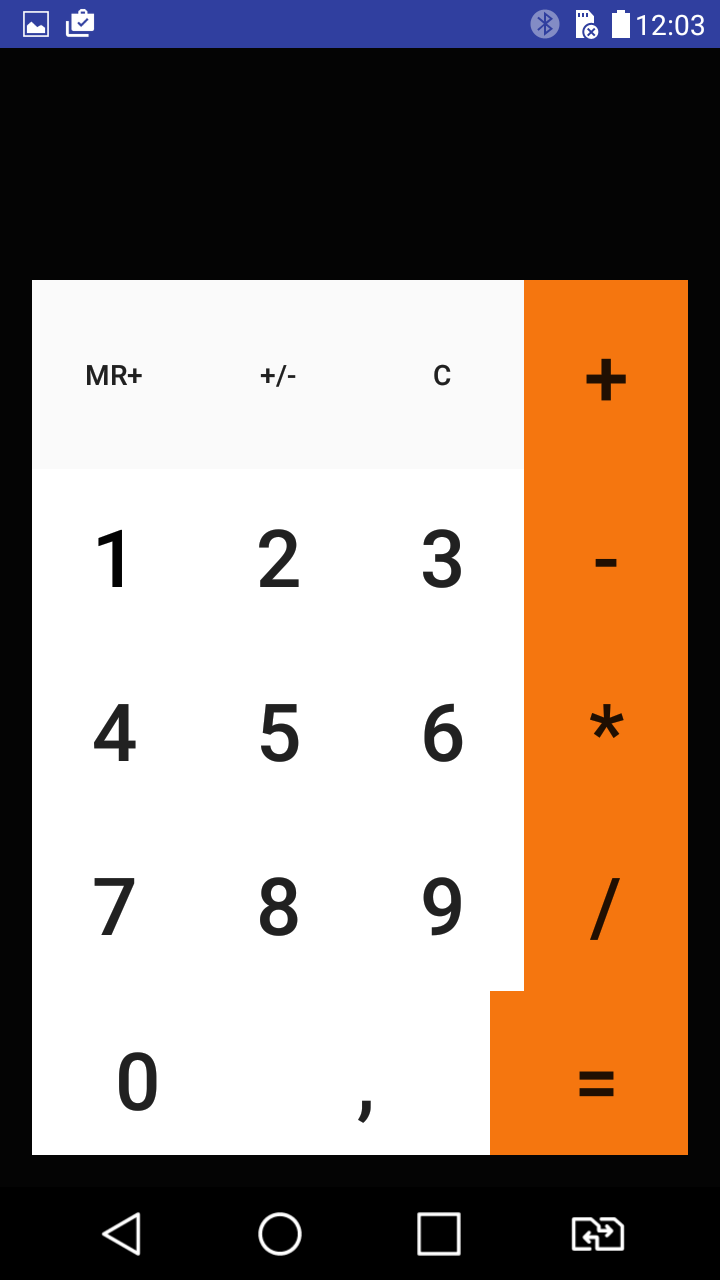


Рис. 1: Программа в начальном состоянии

## Работа с программой на примере функции сложения.

C помощью, панели ввода, вводим необходимое число (15). Затем меняем знак(-15), и пробуем сложить, полученный результат с каким-либо числом(-14). Получаем в результате

Последовательность действий изображена на Рис. 2, Рис. 3, Рис. 4, Рис. 5.

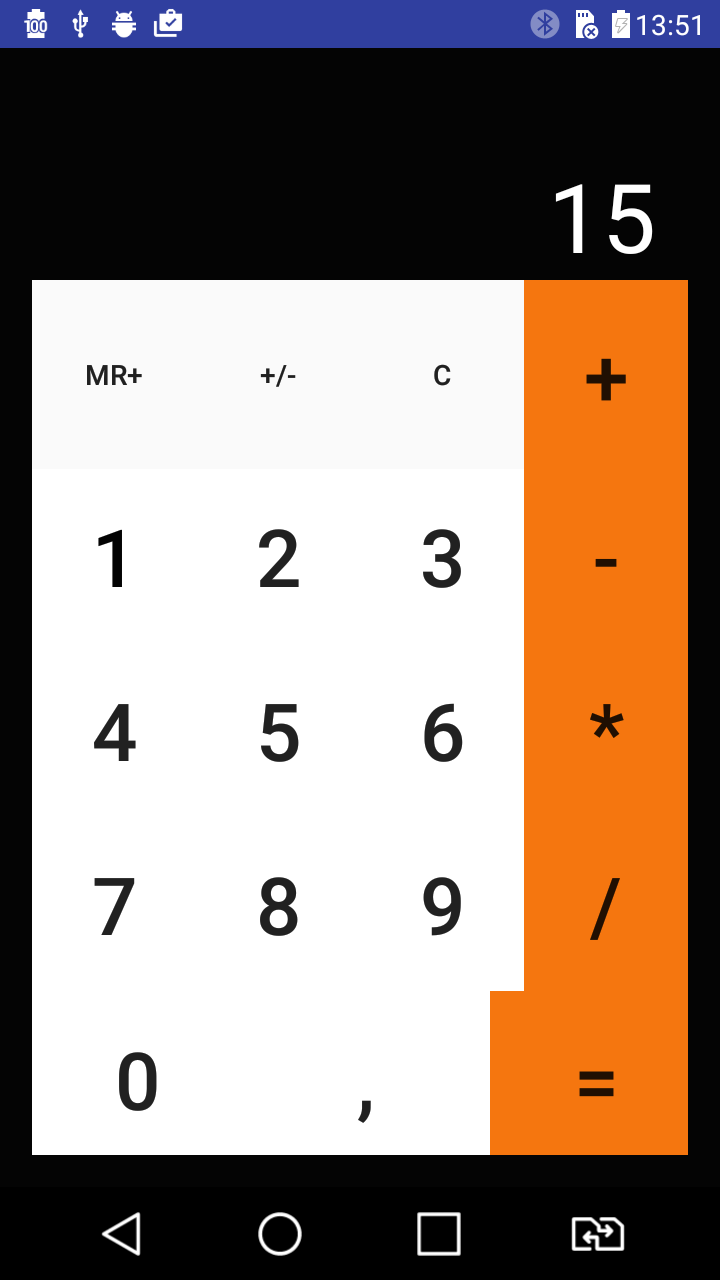


Рис. 2 Отображение введенного операнда “15”

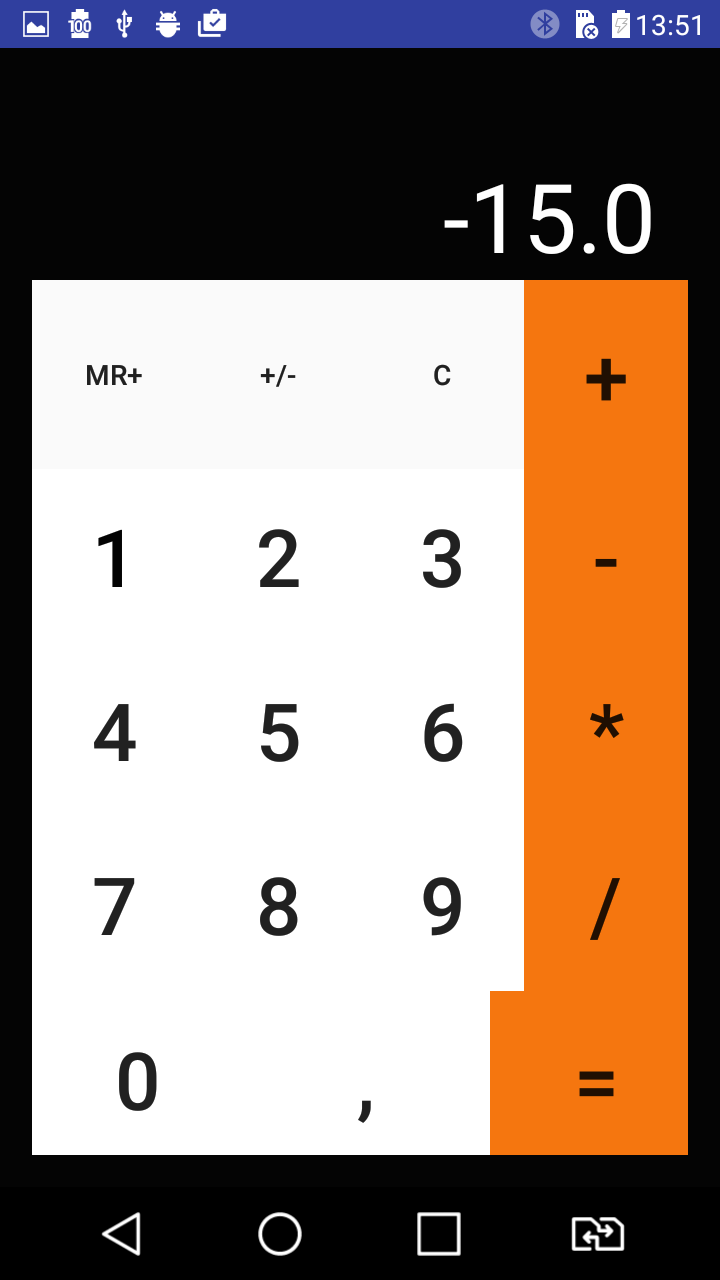


Рис. 3 Выполнение операции «+/-»

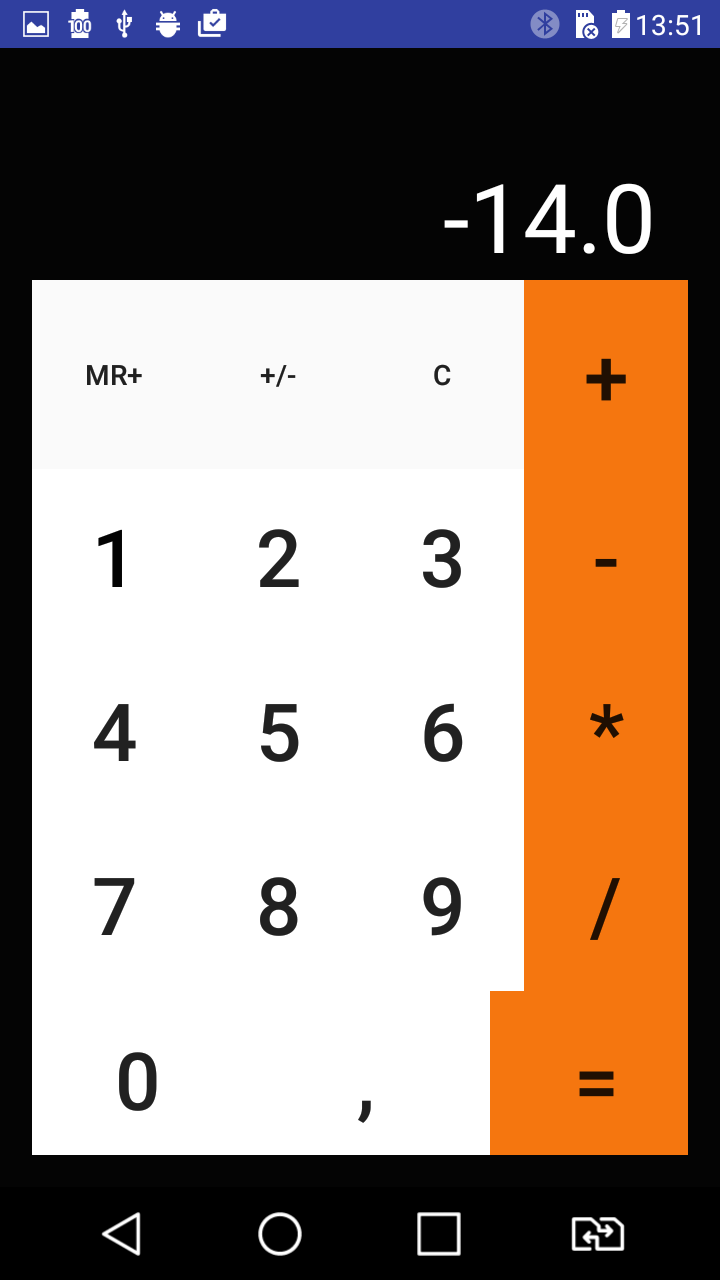


Рис. 4 Отображение результата сложения двух операндов “-15” и “1”

# Листинги исходного кода программы

## Краткое описание

Приложение выполнено с применением модели MVC (Model—View–Controller).

MVC позволяет отделить UI, управление и логику приложения в отдельные части программы.

## User Interface часть (Представленный xml)

|  |
| --- |
|  |

## Java часть

КлассCalculatorModel – Представляетлогикупрограммы, содержитстекоперацийиоперандов, областьдлязапоминаниярезультатаоперацийработыспамятьюиметодыдлядотсупакнимиработысними.

Реализуетинтерфейсы ICalculatorBasic (базовыеарифметическиеоперации) и ICalculatorMemory (Операцииработыспамятью)

|  |
| --- |
| package com.itmm.calculator.model;  */\*\*  \* Created by Dmitry on 9/29/2016.  \*/  /\*\*  \* ICalculatorBasic interface  \*/* public interface ICalculatorBasic {  public double operationAddition(double operand1, double operand2);   public double operationSubtract(double operand1, double operand2);   public double operationMultiply(double operand1, double operand2);   public double operationDivision(double operand1, double operand2); } |
| package com.itmm.calculator.model;  */\*\*  \* Created by Dmitry on 9/29/2016.  \*/  /\*\*  \* ICalculatorBasic interface  \*/* public interface ICalculatorMemory {  public void addToMemoryValue(double value);   public void subFromMemoryValue(double value);   public double getMemoryValue();  } |

|  |
| --- |
| package com.itmm.calculator.model;  import java.util.Stack;  */\*\*  \* Created by Dmitry on 9/29/2016.  \*/* public class CalculatorModel implements ICalculatorBasic, ICalculatorMemory {  protected static Stack<String>*CompStack*;   protected double MemoryValue;   public CalculatorModel() { *CompStack* = new Stack<>(); MemoryValue = 0; }  */\*\*  \* It adds value to current value in Memory  \** ***@param*** *value \*/* public void addToMemoryValue(double value) { MemoryValue += value; }  */\*\*  \* It subtracts value from current value in Memory  \** ***@param*** *value \*/* public void subFromMemoryValue(double value) { MemoryValue -= value; }  */\*\*  \* It current value from Memory in a String representation  \** ***@return*** *the String representation of Memory value  \*/* public double getMemoryValue() { return MemoryValue; }  */\*\*  \* Adds to value to another value and returns the result  \** ***@return*** *the result of addition  \*/* public double operationAddition(double operand1, double operand2) { return operand1 + operand2; }  */\*\*  \* Subtract one value from another values and return sthe result  \** ***@return*** *the result of subtraction  \*/* public double operationSubtract(double operand1, double operand2) { return operand1 - operand2; }  */\*\*  \* Multiply one value to another value and returns the result  \** ***@return*** *the result of multiplication  \*/* public double operationMultiply(double operand1, double operand2) { return operand1 \* operand2; }  */\*\*  \* Division one value from another value and returns the result  \** ***@return*** *the result of division  \*/* public double operationDivision(double operand1, double operand2) { return operand1 / operand2; }  */\*\*  \* Calculates result of mathematical expression that has been  \* stored in Current Stack or just retrieves and returns Operand  \** ***@return*** *the result of calculation or value from Current Stack  \*/* public static double calculate() { double tmpResult = 0;  double tmpOperand = Double.*valueOf*(*CompStack*.pop());   if(*CompStack*.isEmpty()) { return tmpOperand; }   String tmpOperation = *CompStack*.pop();   if (tmpOperation.equals("+")) {  tmpResult = *calculate*() + tmpOperand; } else if( tmpOperation.equals("-")) {  tmpResult = *calculate*() - tmpOperand; } else if (tmpOperation.equals("X")) {  tmpResult = *calculate*() \* tmpOperand; } else if (tmpOperation.equals("/")) {  tmpResult = *calculate*() / tmpOperand; } return tmpResult; }  */\*\*  \* Calculates result of mathematical expression that has been  \* stored in Stack or just retrieves and returns Operand  \** ***@param*** *stack The Stack  \** ***@return*** *the result of calculation or value from Stack  \*/* public static double calculate(Stack<String> stack) { double tmpResult = 0;  double tmpOperand = Double.*valueOf*(stack.pop());   if(stack.isEmpty()) { return tmpOperand; }   String tmpOperation = stack.pop();   if (tmpOperation.equals("+")) {  tmpResult = *calculate*() + tmpOperand; } else if( tmpOperation.equals("-")) {  tmpResult = *calculate*() - tmpOperand; } else if (tmpOperation.equals("X")) {  tmpResult = *calculate*() \* tmpOperand; } else if (tmpOperation.equals("/")) {  tmpResult = *calculate*() / tmpOperand; }  return tmpResult; }  */\*\*  \* Pushes String value (operand or operation) to CompStack  \** ***@param*** *item the String value (operand or operation)  \*/* public void pushToStack(String item) { *CompStack*.push(item); }  */\*\*  \* Pops String value (operand or operation) from CompStack  \** ***@return*** *the String value ()  \*/* public String popFromStack() { return *CompStack*.pop(); }  */\*\*  \* Removes all elements (operands and operations) from CompStack  \*/* public void removeAllElementsFromStack() { *CompStack*.removeAllElements(); }  */\*\*  \* Clone all elements (operands and operations) from CompStack  \* to new Stack and return new Stack  \** ***@retun*** *new Stack  \*/* public Stack<String>cloneStack() { return (Stack<String>) *CompStack*.clone(); } } |

КлассCalculatorControllerсвязывает UIс логикой программы, т.е. класс, который управляет операциями, предоставляемыми классом ControllerModel.

|  |
| --- |
| package com.itmm.calculator.controller;  import java.util.Stack; import com.itmm.calculator.model.CalculatorModel;  */\*\*  \* Created by Dmitry on 9/29/2016.  \*/  /\*\*  \* CalculatorController Class  \*/* public class CalculatorController {  protected boolean performOperation;  private CalculatorModel Model;  */\*\*  \* Constructor for CalculatorController Class  \*/* public CalculatorController() { Model = new CalculatorModel(); }  */\*\*  \* Pushes operand's value to stack  \** ***@param*** *operand The operand of expression  \*/* public void pushOperand(String operand) { Model.pushToStack(operand); performOperation = false; }  */\*\*  \* Pushes operation's value to stack  \* if '=' is specified, the expression's result is computed  \** ***@param*** *operation The operation of expression  \*/* public double pushOperation(String operation) { double tmpResult = 0;   if (performOperation) { Model.popFromStack(); }  if (operation.equals("=")) {  tmpResult = Model.*calculate*(); Model.pushToStack(String.*valueOf*(tmpResult)); } else {  Stack<String> tmpStack = Model.cloneStack(); tmpResult = Model.*calculate*(tmpStack); Model.pushToStack(operation); performOperation = true; }  return tmpResult; }  */\*\*  \* Removes all elements from stack  \*/* public void reset() { Model.removeAllElementsFromStack(); performOperation = false; }  */\*\*  \* Pushes memory operation's value to stack  \* if 'MR' is specified, the stored value in Memory will be retrieved  \** ***@param*** *memoryOperation \** ***@param*** *number \** ***@return*** *\*/* public double pushMemoryOperation(String memoryOperation, double number) { double result = 0;   if (memoryOperation.equalsIgnoreCase("MR")) {  result = Model.getMemoryValue(); } else if (memoryOperation.equalsIgnoreCase("M+")) { Model.addToMemoryValue(number); } else if (memoryOperation.equalsIgnoreCase("M-")) { Model.subFromMemoryValue(number); }  return result; } } |

Класс CalculatorOutput представляет собой набор инструментов для работы с UI, в частности, вывод информации на TextView.

|  |
| --- |
| package com.itmm.calculator.view;  import com.itmm.calculator.R; import android.app.Activity; import android.widget.TextView;  */\*\*  \* Created by Dmitry on 9/29/2016.  \*/  /\*\*  \* Class for writing changes to TextViews in Activity  \*/* public class CalculatorOutput {  private TextView OutputTextView;  */\*\*  \* Constructor for CalculatorOutput Class  \* Initializes TextView by TextView from activity  \** ***@param*** *activity The Activity  \*/* public CalculatorOutput(Activity activity) { OutputTextView = (TextView) activity.findViewById(R.id.*txtSolution*); }  */\*\*  \* Write text 'str' to TextView from Activity that has been specified  \* through Constructor  \** ***@param*** *activity The Activity  \*/* public void setOutputString(String str) { OutputTextView.setText(str); } } |

Класс CalculatorInput представляет собой набор инструментов для работы с UI, в частности,считывание информации с TextViewи нажатых клавиш.

|  |
| --- |
| package com.itmm.calculator.view;  import android.app.Activity; import android.content.res.Resources; import android.view.View; import android.widget.Button;  import com.itmm.calculator.R;  import java.util.ArrayList;  */\*\*  \* Created by Dmitry on 9/29/2016.  \*/  /\*\*  \* Class for reading changes from TextViews from Activity  \*/* public class CalculatorInput {  private ArrayList<Button>OperandsButtons;  private ArrayList<Button>OperatesButtons;  private ArrayList<Button>MemoryButtons;   private CalculatorInput.InputHappend Delagete;  */\*\*  \* Constructor for CalculatorInput class  \** ***@param*** *activity The activity with views  \** ***@param*** *delegate The Delegate interface which has an definition of methods  \*/* public CalculatorInput(Activity activity, CalculatorInput.InputHappend delegate) { Delagete = delegate; OperandsButtons = new ArrayList<Button>(); OperatesButtons = new ArrayList<Button>(); MemoryButtons = new ArrayList<Button>();  setUIComponent(activity); }  */\*\*  \* Set UI Components (Buttons) for activity  \** ***@param*** *activity The activity with views  \*/* private void setUIComponent(Activity activity) {  Resources tmpRes = activity.getResources();   for (int i = 0; i <10; i++) { int tmpOperandId = tmpRes.getIdentifier("btn" + i, "id", activity.getPackageName()); OperandsButtons.add((Button) activity.findViewById(tmpOperandId)); } OperandsButtons.add((Button) activity.findViewById(R.id.*btnDecimal*));  OperatesButtons.add((Button) activity.findViewById(R.id.*btnClear*)); OperatesButtons.add((Button) activity.findViewById(R.id.*btnEquals*)); OperatesButtons.add((Button) activity.findViewById(R.id.*btnAdd*)); OperatesButtons.add((Button) activity.findViewById(R.id.*btnSubtract*)); OperatesButtons.add((Button) activity.findViewById(R.id.*btnMultiply*)); OperatesButtons.add((Button) activity.findViewById(R.id.*btnDivide*));  MemoryButtons.add((Button) activity.findViewById(R.id.*btnMplus*)); MemoryButtons.add((Button) activity.findViewById(R.id.*btnMminus*)); MemoryButtons.add((Button) activity.findViewById(R.id.*btnMR*));   for (Button btn : OperandsButtons) {  btn.setOnClickListener(onClickOperand); }  for (Button btn : OperatesButtons) {  btn.setOnClickListener(onClickOperation); }  for (Button btn : MemoryButtons) {  btn.setOnClickListener(onClickMemory); }  }  */\*\*  \* Listener for Operand Buttons in MainActivity  \*/* private Button.OnClickListener onClickOperand = new Button.OnClickListener() { @Override public void onClick(View v) {  Button tmpClickButton = (Button) v; String tmpText = tmpClickButton.getText().toString(); Delagete.operandIn(tmpText); }  };  */\*\*  \* Listener for Operation Buttons in MainActivity  \*/* private Button.OnClickListener onClickOperation = new Button.OnClickListener() { @Override public void onClick(View v) {  Button tmpClickButton = (Button) v; String tmpText = tmpClickButton.getText().toString(); Delagete.operationIn(tmpText); }  };  */\*\*  \* Listener for Memory Operation Buttons in MainActivity  \*/* private Button.OnClickListener onClickMemory = new Button.OnClickListener() { @Override public void onClick(View v) {  Button tmpClickButton = (Button) v; String tmpText = tmpClickButton.getText().toString(); Delagete.memoryOperationIn(tmpText); }  };  */\*\*  \* Interface with predefined methods as a actions for listener  \*/* public interface InputHappend { // When user input operand, notify which operand by input. void operandIn(String operand); // When user input operation, notify which operate by input. void operationIn(String operation); // When user input memory operation, notify which operate by input. void memoryOperationIn(String memoryOperation); } } |

Класс CalculatorViewпредставляет собой набор объектов классов CalculatorOutput иCalculatorInput.

|  |
| --- |
| package com.itmm.calculator.view;  import android.app.Activity; import com.itmm.calculator.view.CalculatorInput.InputHappend;   */\*\*  \* Created by Dmitry on 9/29/2016.  \*/  /\*\*  \* Class that contains both objects  \*/* public class CalculatorView {  public CalculatorInput InputView;  public CalculatorOutput OutputView;  */\*\*  \* Constructor for CalculatorView Class  \** ***@param*** *activity \** ***@param*** *delegate \*/* public CalculatorView(Activity activity, InputHappend delegate) { InputView = new CalculatorInput(activity, delegate); OutputView = new CalculatorOutput(activity); }  } |

Класс MainActivity

|  |
| --- |
| package com.itmm.calculator;  import android.support.v7.app.AppCompatActivity; import android.os.Bundle;  import com.itmm.calculator.controller.CalculatorController; import com.itmm.calculator.view.CalculatorInput.InputHappend; import com.itmm.calculator.view.CalculatorView;  */\*\*  \* MainActivity Class  \*/* public class MainActivity extends AppCompatActivity implements InputHappend {   CalculatorView View; CalculatorController Controller;  private String Number = "0";  @Override protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.*main\_screen*);  Controller = new CalculatorController(); View = new CalculatorView(this, this); }  @Override public void operandIn(String operand) { Number = Number.equals("0") ? operand : Number + operand; View.OutputView.setOutputString(Number); }  @Override public void operationIn(String operation) { if (operation.equalsIgnoreCase("C")) { Controller.reset(); Number = "0"; View.OutputView.setOutputString(Number);  return; } if (!Number.equals("0")) { Controller.pushOperand(Number); }  double tmpResult = Controller.pushOperation(operation);   if (tmpResult % 1d == 0d) { int tmp = Double.*valueOf*(tmpResult).intValue(); View.OutputView.setOutputString(String.*valueOf*(tmp)); } else { View.OutputView.setOutputString(String.*valueOf*(tmpResult)); } Number = "0"; }  @Override public void memoryOperationIn(String memoryOperation) { double tmpResult = Controller.pushMemoryOperation(memoryOperation, Double.*valueOf*(Number));   if (memoryOperation.equalsIgnoreCase("MR"))  { if (tmpResult % 1d == 0d) { int tmp = Double.*valueOf*(tmpResult).intValue(); View.OutputView.setOutputString(String.*valueOf*(tmp)); } else { View.OutputView.setOutputString(String.*valueOf*(tmpResult)); } Number = String.*valueOf*(tmpResult); }  } } |

#### Вывод

В данной лабораторной работе я познакомился с основами написания Android-приложений с помощью среды разработки AndroidStudio. А также с применение метода MVCдля конкретной задачи – разработка архитектуры приложения Калькулятора.

Данная работа дала мне необходимые теоретические и практические навыки работы с Javaкодом, XMLразметкой, основными стандартными пакетамиJava иAndroidSDK.