

22-04-2025

Arquitectura *Web en tres* *Capas*

Unidad de trabajo 4

Francisco Javier Otero Herrero
Grupo ATU

Arquitectura Web en tres Capas

Arquitectura Web en tres Capas

Contenido

<i>Modelo Vista Controlador</i>	<i>3</i>
<i>Instalar NeatBeans</i>	<i>3</i>
<i>Proyecto Calculadora</i>	<i>5</i>
<i>Modelo</i>	<i>6</i>
<i>Vista</i>	<i>7</i>
<i>Controlador</i>	<i>8</i>

Arquitectura Web en tres Capas

Modelo Vista Controlador

El patrón MVC (**Modelo, Vista, Controlador**), es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz que ve el usuario de la lógica del negocio en tres componentes distintos. Es más frecuente en aplicaciones Web que en aplicaciones de escritorio, sin embargo, es aplicable también a este, sin ningún problema, **Java** ya contaba hace rato con **Observer** y **Observable**, herramientas que nos ayudan a la interacción entre la interfaz y el modelo, sin embargo, el ejemplo que dejamos a continuación no hace uso de estas herramientas.

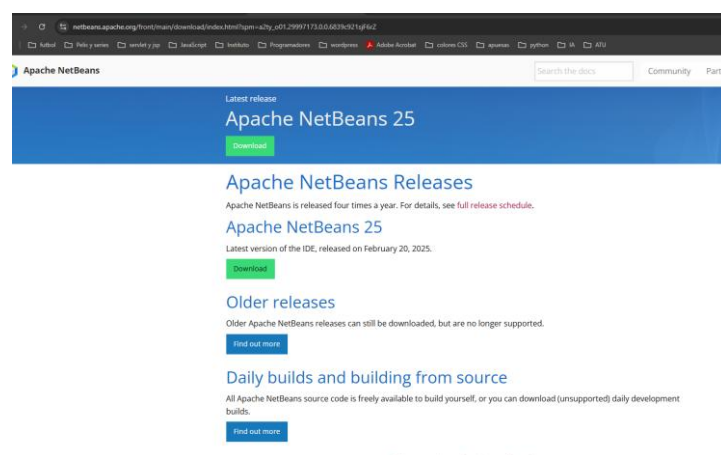
La descripción del patrón **MVC** es:

- ✓ **Vista (View):** Representa la interfaz de usuario y todas las herramientas con las cuales el usuario hace uso del programa.
- ✓ **Modelo (Model):** Es donde está toda la lógica del negocio, la representación de todo el sistema incluido la interacción con una base de datos, si es que el programa así lo requiere.
- ✓ **Controlador (Controller):** Este componente es el que responde a la interacción (eventos) que hace el usuario en la interfaz y realiza las peticiones al modelo para pasar estos a la vista.

En este manual vamos a crear una aplicación con **Java** y **NeatBeans**, siguiendo el **MVC**, para ello comenzaremos instalando todo lo necesario. En nuestro caso ya tenemos instalado un **JDK** de anteriores prácticas por lo que pasaremos a descargar e instalar **NeatBeans**.

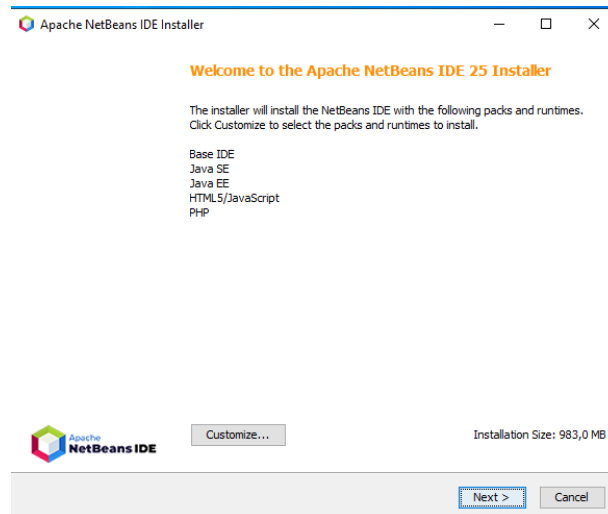
Instalar NeatBeans

https://netbeans.apache.org/front/main/download/index.html?spm=a2ty_o01.2997173.0.0.6839c921sjF6rZ

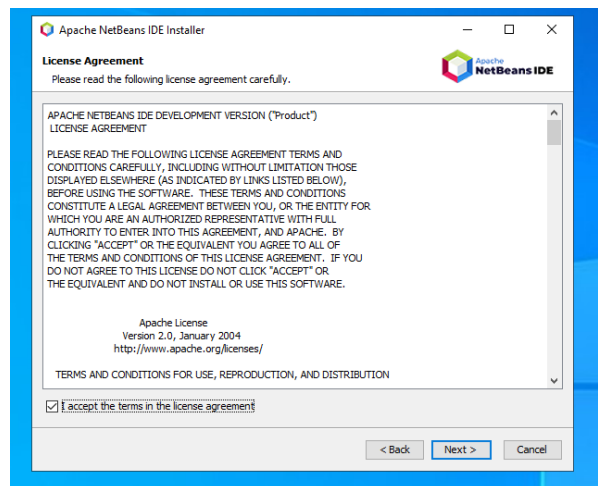


Arquitectura Web en tres Capas

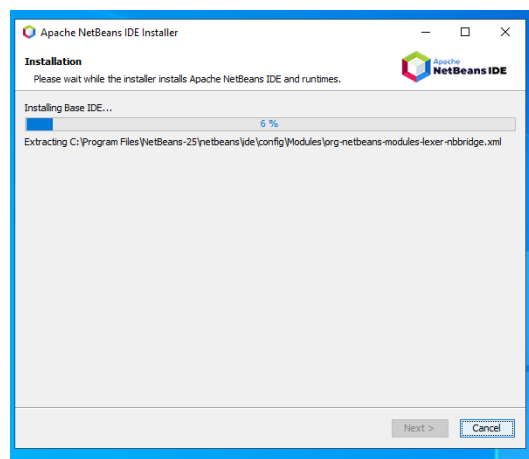
Comenzamos a instalar NetBeans.



Aceptar términos.

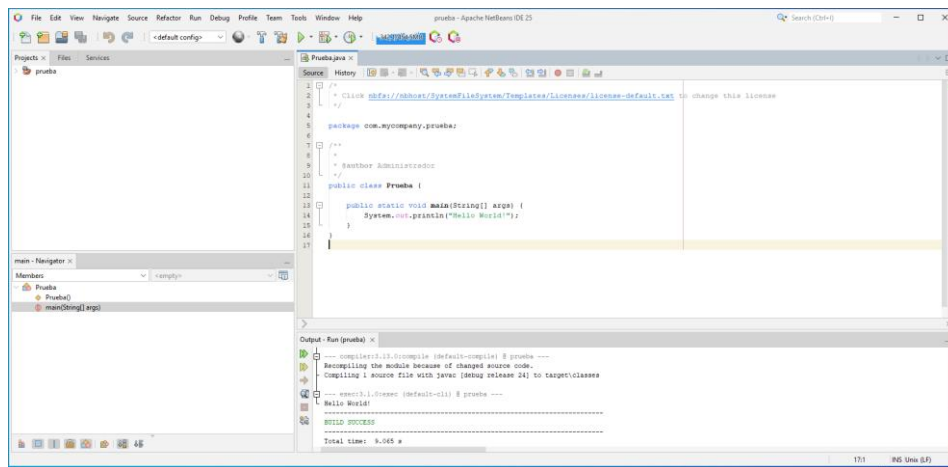


Esperamos que finalice la instalación.



Arquitectura Web en tres Capas

Creamos un nuevo proyecto mediante la interfaz de este IDE para comprobar que funciona correctamente.

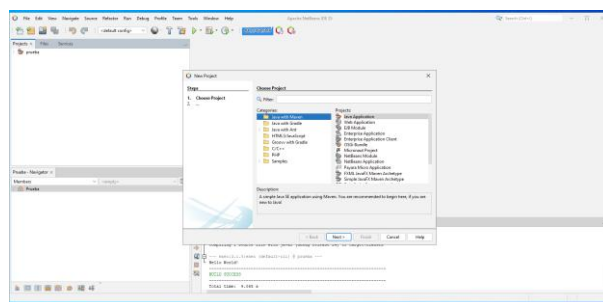


Podemos ver en la parte inferior, en la consola el mensaje de “Hello World”, por lo que podemos confirmar que todo parece que funciona correctamente.

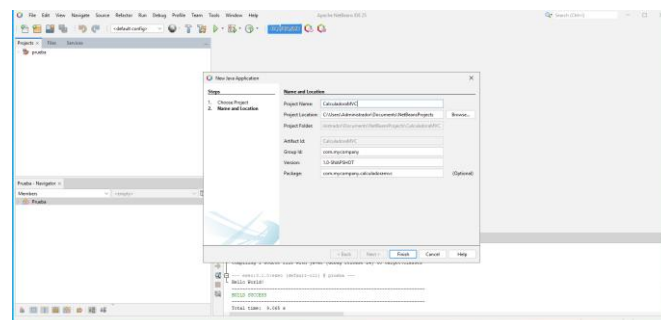
Proyecto Calculadora

En este apartado vamos a crear una aplicación que será una calculadora muy sencilla utilizando el MVC, Java y NeatBeans.

Creamos un nuevo proyecto, seleccionamos Java Application y pulsamos next.



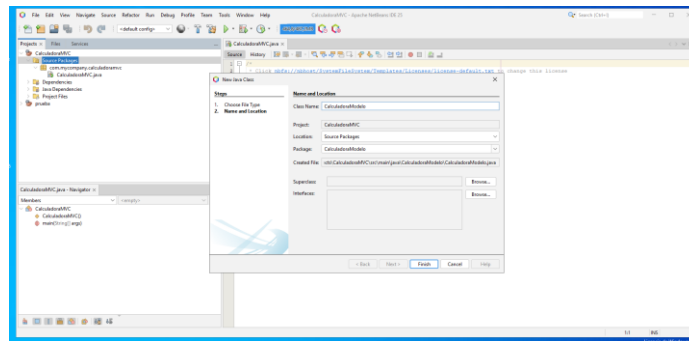
Agregamos el nombre a nuestro nuevo proyecto, en nuestro caso será “**jc_mvc_demo**”.



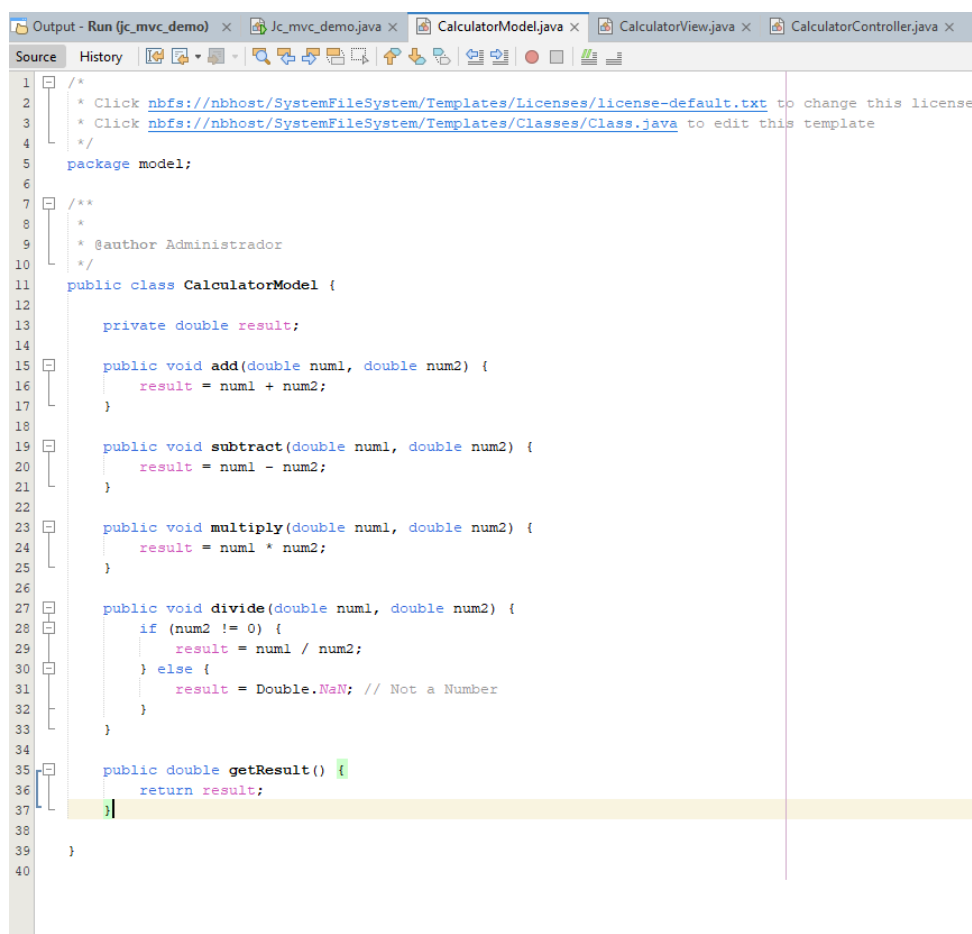
Arquitectura Web en tres Capas

✓ **Modelo**

En esta parte lo primero que debemos crear es un nuevo “*java package*” pulsando con el botón derecho sobre ***source packages***, que llamaremos “***model***”, dentro de este nuevo paquete con botón derechos sobre él creamos una nueva “*java class*” que llamaremos “***CalculatorModel***”



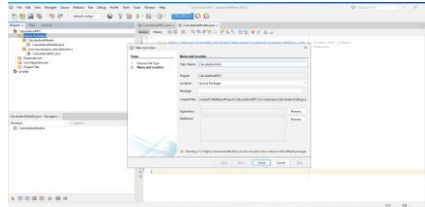
Añadimos el siguiente código que podemos ver en la imagen.



Arquitectura Web en tres Capas

✓ Vista

En este apartado debemos repetir el mismo proceso que en el apartado anterior, creamos nuevo paquete esta vez llamado **“view”**, dentro de este paquete una nueva clase llamada **“CalculatorView”**.



Añadimos el siguiente código.

```
Source | History | Run | Debug | Window | Help | Eclipse
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 32 33 34 35 36 37 38 39 40 41 42 43
44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81

/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package view;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionListener;

/**
 *
 * @author Administrator
 */
public class CalculatorView extends JFrame {

    private JTextField number1Field = new JTextField(10);
    private JTextField number2Field = new JTextField(10);
    private JTextField resultField = new JTextField(10);
    private JButton addButton = new JButton("+");
    private JButton subtractButton = new JButton("-");
    private JButton multiplyButton = new JButton("*");
    private JButton divideButton = new JButton("/");

    public CalculatorView() {
        // Set up the view (JFrame)
        setTitle("Calculadora Java & NetBeans");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(300, 200);
        setLayout(new FlowLayout());

        // Add components to the view
        add(new JLabel("Numero 1:"));
        add(number1Field);
        add(new JLabel("Numero 2:"));
        add(number2Field);
        add(addButton);
        add(subtractButton);
        add(multiplyButton);
        add(divideButton);
        add(new JLabel("Resultado:"));
        add(resultField);

        resultField.setEditable(false); // Make result field read-only

        // Center the window
        setLocationRelativeTo(null);
    }

    public double getNumber1() {
        return Double.parseDouble(number1Field.getText());
    }

    public double getNumber2() {
        return Double.parseDouble(number2Field.getText());
    }

    public void setResult(double result) {
        resultField.setText(Double.toString(result));
    }

    public void addCalculateListener(ActionListener listener) {
        addButton.addActionListener(listener);
    }

    public void subtractCalculateListener(ActionListener listener) {
        subtractButton.addActionListener(listener);
    }

    public void multiplyCalculateListener(ActionListener listener) {
        multiplyButton.addActionListener(listener);
    }

    public void divideCalculateListener(ActionListener listener) {
        divideButton.addActionListener(listener);
    }

    public void displayErrorMessage(String errorMessage) {
        JOptionPane.showMessageDialog(this, errorMessage);
    }
}
```


Arquitectura Web en tres Capas

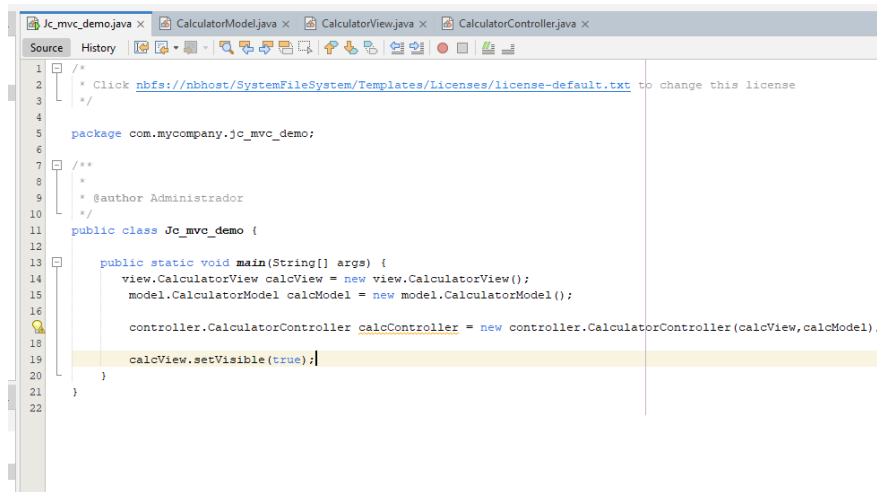
✓ Controlador

El controlador conectará la vista con el modelo y manejará las acciones del usuario. Debemos hacer la misma operación que en apartados anteriores, crearemos un nuevo paquete llamado **“controller”**, y dentro de este, una clase que llamaremos **“CalculatorController.java”** y añadimos el siguiente código.

```
Source | Run | Output | History | ... | CalculatorModel.java | CalculatorView.java | CalculatorController.java |
1  /**
2   * Click https://www.eclipse.org/legal/epl-1.0/ to change this license
3   * Click https://www.eclipse.org/legal/epl-1.0/ to edit this template
4   */
5   package controller;
6
7   import model.CalculatorModel;
8   import view.CalculatorView;
9
10  import java.awt.event.ActionEvent;
11  import java.awt.event.ActionListener;
12
13  /**
14   *
15   * @author Administrator
16   */
17  public class CalculatorController {
18
19      private CalculatorView view;
20      private CalculatorModel model;
21
22      public CalculatorController(CalculatorView view, CalculatorModel model) {
23          this.view = view;
24          this.model = model;
25
26          // Set up listeners for the buttons
27          view.addCalculateListener(new AddListener());
28          view.addSubtractListener(new SubtractListener());
29          view.addMultiplyListener(new MultiplyListener());
30          view.addDivideListener(new DivideListener());
31      }
32
33      private class AddListener implements ActionListener {
34          public void actionPerformed(ActionEvent e) {
35              double number1, number2 = 0;
36
37              // Surround calculations in try block to catch NumberFormatException
38              try {
39                  number1 = view.getNumber1();
40                  number2 = view.getNumber2();
41
42              } catch (NumberFormatException ex) {
43                  view.displayErrorMessage("You Need to Enter 2 Integers");
44                  return;
45              }
46
47              model.add(number1, number2);
48              view.setResult(model.getResult());
49          }
50      }
51
52      private class SubtractListener implements ActionListener {
53          public void actionPerformed(ActionEvent e) {
54              double number1, number2 = 0;
55
56              try {
57                  number1 = view.getNumber1();
58                  number2 = view.getNumber2();
59
60              } catch (NumberFormatException ex) {
61                  view.displayErrorMessage("You Need to Enter 2 Integers");
62                  return;
63              }
64
65              model.subtract(number1, number2);
66              view.setResult(model.getResult());
67          }
68      }
69
70      private class MultiplyListener implements ActionListener {
71          public void actionPerformed(ActionEvent e) {
72              double number1, number2 = 0;
73
74              try {
75                  number1 = view.getNumber1();
76                  number2 = view.getNumber2();
77
78              } catch (NumberFormatException ex) {
79                  view.displayErrorMessage("You Need to Enter 2 Integers");
80                  return;
81              }
82
83              model.multiply(number1, number2);
84              view.setResult(model.getResult());
85          }
86      }
87
88      private class DivideListener implements ActionListener {
89          public void actionPerformed(ActionEvent e) {
90              double number1, number2 = 0;
91
92              try {
93                  number1 = view.getNumber1();
94                  number2 = view.getNumber2();
95
96              } catch (NumberFormatException ex) {
97                  view.displayErrorMessage("You Need to Enter 2 Integers");
98                  return;
99              }
100
101              model.divide(number1, number2);
102              view.setResult(model.getResult());
103          }
104      }
105  }
```

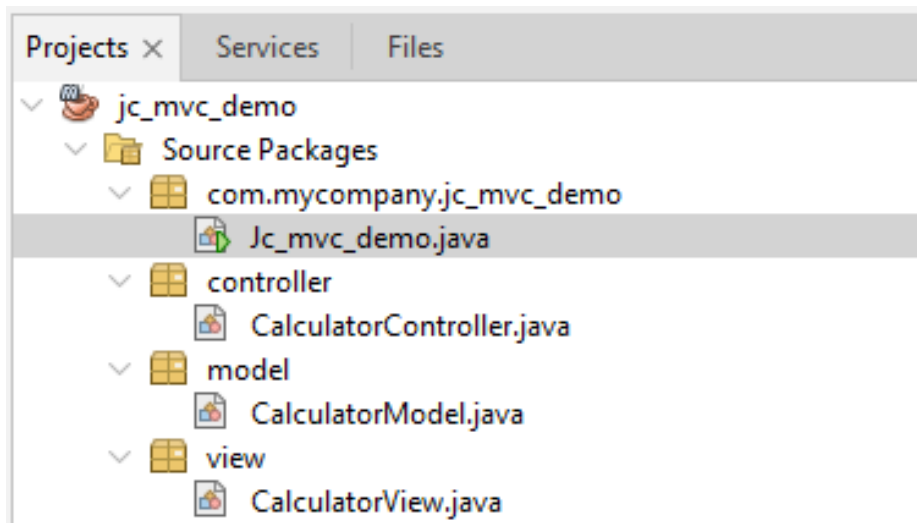
Arquitectura Web en tres Capas

Finalmente, solo nos quedaría conectar todo, es decir, modelo, vista y controlador en la clase principal de nuestro proyecto, que se llama **"jc_mvc_demo.java"**.



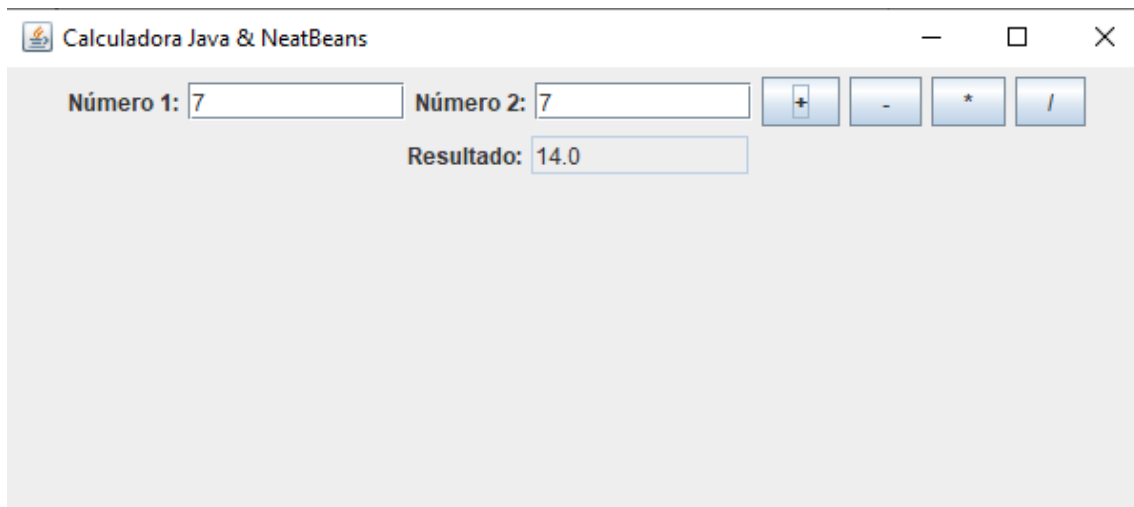
```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  */
4
5  package com.mycompany.jc_mvc_demo;
6
7  /**
8   *
9   * @author Administrador
10  */
11  public class Jc_mvc_demo {
12
13      public static void main(String[] args) {
14          view.CalculatorView calcView = new view.CalculatorView();
15          model.CalculatorModel calcModel = new model.CalculatorModel();
16
17          controller.CalculatorController calcController = new controller.CalculatorController(calcView, calcModel);
18
19          calcView.setVisible(true);
20      }
21  }
22
```

En este punto vamos a comprobar el funcionamiento de la calculadora que hemos creado con **Java, en el IDE de NeatBeans**, y siguiendo las pautas que marca el **modelo de tres capas(MVC)**. Si hemos llegado hasta este punto deberemos tener una estructura en nuestro proyecto similar a la siguiente:



Arquitectura Web en tres Capas

Finalmente comprobamos que todo funciona correctamente, este es el aspecto de la interfaz de la calculadora que acabamos de crear.



Una interfaz sencilla con dos campos para que el usuario introduzca dos números, al lado cuatro botones para que el usuario haga una suma, resta, multiplicación o división y por debajo otro campo que mostrará el resultado.

La finalidad de este trabajo es entender el modelo de tres capas, el cual divide el modelo, la vista y el controlador en tres paquetes distintos los cuales son fáciles de ampliar si el proyecto creciera, fáciles de mantener, y lo más importante que se pueden reutilizar en cualquier otro proyecto que podamos realizar en el futuro y requiera de alguna de estas funcionalidades.