

Entregable #5

Presentacion

Trabajo realizado por Jhonatan David Asprilla Arango

CC 1018222341

jasprilla@unal.edu.co

Punto 1

Código

El código de este ejercicio también se encuentra disponible en el [GitHub](#).

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Optional;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;

// Se implementa el patron de diseño subscriber para que los componentes de ui de
// la aplicacion que dependan
// del estado de las notas puedan ser notificados cuando estas cambien y actualicen
// su estado en consecuencia.
interface Subscriber {
    void Notify();
}

interface Suscriptable {
    void registerSubscriber(Subscriber subscriber);
    void notifySubscribers();
}

// Se implementa la interfaz subscribable para indicar que es posible suscribirse
// a los cambios de estado de esta clase
```

```

class Scores implements Suscriptable {
    public Integer max_score_quantity;
    private Double[] scores;
    private ArrayList<Subscriber> subscribers;

    public Scores(Double[] scores) {
        this.scores = scores;
        this.subscribers = new ArrayList<Subscriber>();
        max_score_quantity = scores.length;
    }

    // Se definen getters y setters para la propiedad scores para poder obtener un
    control mas granular sobre
    // las operaciones que se pueden efectuar sobre las notas, ya que este tiene
    una logica asociada al proceso de renderizado
    public Double[] getScores() {
        return scores;
    }

    public void setScores(Double[] scores) {
        this.scores = scores;
        // Aqui se puede ver la dependencia que hay sobre caracterisiticcas de este
        objeto en el proceso de renderizado
        // Esto es debido a que la cantidad de campos que se renderizan esta
        directamente relacionados con la propiedad
        // max_score_quantity, por lo que si esta cambia, se debe volver a
        renderizar el formulario
        resize_scores(scores.length);
        max_score_quantity = scores.length;
        notifySubscribers();
    }

    // Se implementan los metodos necesarios para implementar la interfaz
    suscriptable
    @Override
    public void registerSubscriber(Subscriber subscriber) {
        if (Arrays.stream(subscribers.toArray(new
Subscriber[subscribers.size()])).filter((Subscriber sub) -> sub ==
subscriber).count() > 0) return;
        subscribers.add(subscriber);
    }

    // Notifica a todos los subscriptores sobre cambios en el objeto, sin compartir
    dtos sobre el cambio del estado, el componente
    // que necesite acceder al nuevo estado de la aplicacion para reaccionar al
    evento debe obtenerlo por si mismo en el metodo Notify

```

```

@Override
public void notifySubscribers() {
    Arrays.stream(subscribers.toArray(new
Subscriber[subscribers.size()])).forEach((Subscriber sub) -> sub.Notify());
}

// Se implementa un metodo para redimensionar el arreglo de notas
public void resize_scores(Integer new_size) {
    if (new_size < max_score_quantity) return;
    scores = Arrays.copyOf(scores, new_size);
    scores = Arrays.stream(scores).map((Double score) -> score == null ? 0.0 :
score).toArray(Double[]::new);
    max_score_quantity = new_size;
}

// Desviacion estandar del arreglo de notas
public Double std_dev() {
    Double avg = average();
    Optional<Double> distance_to_avg_sum = Arrays.stream(scores).reduce((Double
acc, Double score) -> acc + Math.pow(score - avg, 2));
    if (distance_to_avg_sum.isPresent()) {
        return Math.sqrt(distance_to_avg_sum.get() / scores.length);
    } else {
        return 0.0;
    }
}

// Mayor nota del arreglo de notas
public Optional<Double> highest_score() {
    return Arrays.stream(scores).reduce((Double acc, Double score) ->
Math.max(acc, score));
}

// Nota mas baja del arreglo de notas
public Optional<Double> lowest_score() {
    return Arrays.stream(scores).reduce((Double acc, Double score) ->
Math.min(acc, score));
}

// Promedio del arreglo de notas
public Double average() {
    Optional<Double> score_sum = Arrays.stream(scores).reduce((Double acc,
Double score) -> acc + score);
    if (score_sum.isPresent()) {
        return score_sum.get() / scores.length;
    } else {

```

```

        return 0.0;
    }
}

class ScoreMetrics extends JPanel implements Subscriber {
    private Scores scores;
    private GridBagLayout layout;
    public ScoreMetrics(Scores scores) {
        this.scores = scores;
        this.scores.registerSubscriber(this);

        // Definimos el layout del JPanel
        layout = new GridBagLayout();
        setLayout(layout);

        // Inicializamos los componentes internos del JPanel
        initializeComponents();
    }

    public void initializeComponents() {
        // Limpiar los componentes del formulario
        Arrays.stream(getComponents()).forEach((Component component) ->
component.setVisible(false));
        Arrays.stream(getComponents()).forEach((Component component) -> {
            remove(component);
            revalidate();
            repaint();
        });

        // Renderizar los componentes del formulario
        JPanel metrics_panel = new JPanel(new GridLayout(0, 1));

metrics_panel.setBorder(javax.swing.BorderFactory.createTitledBorder("Metricas"));
        metrics_panel.add(new javax.swing.JLabel("Promedio: " + scores.average()));
        metrics_panel.add(new javax.swing.JLabel("Desviacion estandar: " +
scores.std_dev()));
        metrics_panel.add(new javax.swing.JLabel("Nota mas alta: " +
scores.highest_score().orElse(0.0)));
        metrics_panel.add(new javax.swing.JLabel("Nota mas baja: " +
scores.lowest_score().orElse(0.0)));

        GridBagConstraints constraints = new GridBagConstraints();
        constraints.gridx = 0;
        constraints.gridy = 0;
        constraints.weightx = 1.0;
    }
}

```

```

        constraints.fill = GridBagConstraints.HORIZONTAL;

        add(metrics_panel, constraints);
    }

    @Override
    public void Notify() {
        initializeComponents();
    }
}

class ScoreForm extends JPanel implements ActionListener {
    // Composicion de la clase ScoreForm con la clase Scores para poder acceder a
    // los datos de las notas
    private Scores scores;

    // Almacena los paneles que contienen los campos de las notas
    // para poder eliminarlos y volver a renderizarlos
    private ArrayList<JPanel> score_panels;
    // Almacena los campos de las notas para poder acceder al valor de estos
    private ArrayList<JTextField> score_fields;
    // Almacena los botones del formulario para poder eliminarlos y volver a
    // renderizarlos
    private ArrayList<JButton> buttons;

    private GridBagLayout layout;

    public ScoreForm(Scores scores) {
        this.scores = scores;
        this.score_fields = new ArrayList<JTextField>();
        this.score_panels = new ArrayList<JPanel>();
        this.buttons = new ArrayList<JButton>();

        layout = new GridBagLayout();

        setLayout(layout);

        initializeComponents();
    }

    public void initializeForm() {
        // Limpiar los componentes del formulario
        Arrays.stream(score_fields.toArray(new
JTextField[score_fields.size()])).forEach((JTextField field) ->

```

```

field.setVisible(false));
    Arrays.stream(score_fields.toArray(new
JTextField[score_fields.size()])).forEach((JTextField field) -> {
        remove(field);
        revalidate();
        repaint();
    });
    Arrays.stream(score_panels.toArray(new
JPanel[score_panels.size()])).forEach((JPanel panel) -> panel.setVisible(false));
    Arrays.stream(score_panels.toArray(new
JPanel[score_panels.size()])).forEach((JPanel panel) -> {
        remove(panel);
        revalidate();
        repaint();
    });
    score_fields.clear();
    score_panels.clear();

    // Renderizar los componentes del formulario
    for (int i = 0; i < scores.max_score_quantity; i++) {
        JPanel form_field_panel = new JPanel(new BorderLayout());

        form_field_panel.setBorder(javax.swing.BorderFactory.createTitledBorder("Nota " +
(i + 1)));

        JTextField score_field = new JTextField(scores.getScores()
[i].toString());
        form_field_panel.add(score_field);
        score_fields.add(score_field);

        GridBagConstraints constraints = new GridBagConstraints();
        constraints.gridx = 0;
        constraints.gridy = i;
        constraints.weightx = 1.0;
        constraints.fill = GridBagConstraints.HORIZONTAL;

        add(form_field_panel, constraints);
        score_panels.add(form_field_panel);
    }
}

public void initializeButtons() {
    // Limpiar los componentes de los botones
    Arrays.stream(buttons.toArray(new
JButton[buttons.size()])).forEach((JButton button) -> button.setVisible(false));
    Arrays.stream(buttons.toArray(new

```

```

JButton[buttons.size()])).forEach((JButton button) -> {
    remove(button);
    revalidate();
    repaint();
});
buttons.clear();

// Renderizar los componentes de los botones
JPanel form_button_panel = new JPanel(new FlowLayout());

JButton calculate_btn = new JButton("Calcular");
calculate_btn.setActionCommand("calculate");
calculate_btn.addActionListener(this);
form_button_panel.add(calculate_btn);
buttons.add(calculate_btn);

JButton add_field_btn = new JButton("Añadir campo");
add_field_btn.setActionCommand("add_field");
add_field_btn.addActionListener(this);
form_button_panel.add(add_field_btn);
buttons.add(add_field_btn);

JButton reset_fields_btn = new JButton("Resetear campos");
reset_fields_btn.setActionCommand("reset_fields");
reset_fields_btn.addActionListener(this);
form_button_panel.add(reset_fields_btn);
buttons.add(reset_fields_btn);

GridBagConstraints constraints = new GridBagConstraints();
constraints.gridx = 0;
constraints.gridy = scores.max_score_quantity;
constraints.weightx = 1.0;
constraints.fill = GridBagConstraints.HORIZONTAL;

add(form_button_panel, constraints);
}

public void initializeComponents() {
    initializeForm();
    initializeButtons();
}

@Override
public void actionPerformed(ActionEvent e) {
    switch (e.getActionCommand()) {
        case "add_field":

```

```

        this.scores.resize_scores(this.scores.max_score_quantity + 1);
        initializeComponents();
        break;
    case "calculate":
        Double[] scores = Arrays.stream(score_fields.toArray(new
JTextField[score_fields.size()])).map((JTextField field) ->
Double.parseDouble(field.getText())).toArray(Double[]::new);
        this.scores.setScores(scores);
        break;
    case "reset_fields":
        this.scores.setScores(Main.default_values);
        initializeComponents();
        break;
    default:
        break;
    }
}
}

class MainWindow extends JFrame {
    private Scores scores;
    private FlowLayout layout;

    public MainWindow(Scores scores) {
        this.scores = scores;

        layout = new FlowLayout();
        setLayout(layout);
        setBounds(0, 0, 500, 500);

        initializeComponents();
    }

    public void initializeComponents() {
        // Componente de formulario
        ScoreForm score_form = new ScoreForm(scores);

score_form.setBorder(javax.swing.BorderFactory.createTitledBorder("Formulario de
notas"));

        // Componente de metricas de notas
        ScoreMetrics score_metrics = new ScoreMetrics(scores);

score_metrics.setBorder(javax.swing.BorderFactory.createTitledBorder("Metricas de
notas"));

```



```

        add(score_form);
        add(score_metrics);
    }
}

class Main {
    private static MainWindow mainWindow;
    public static Double[] default_values = { 0.0 };

    public static void main(String[] args) {
        Scores scores = new Scores(default_values);
        mainWindow = new MainWindow(scores);
        mainWindow.setVisible(true);
    }
}

```

Diagrama de clases

El archivo original del diagrama para el programa **StarUML** se encuentra disponible en el [GitHub](#).

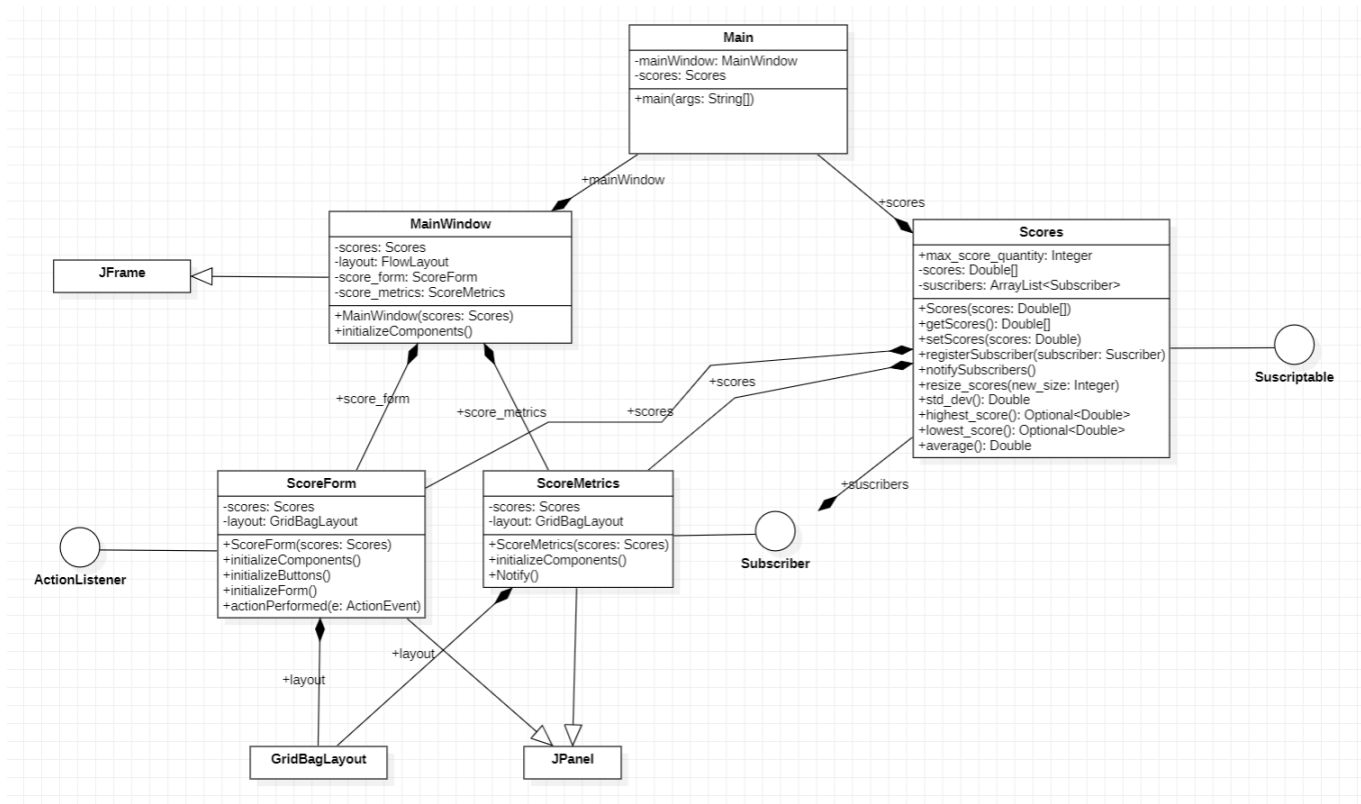
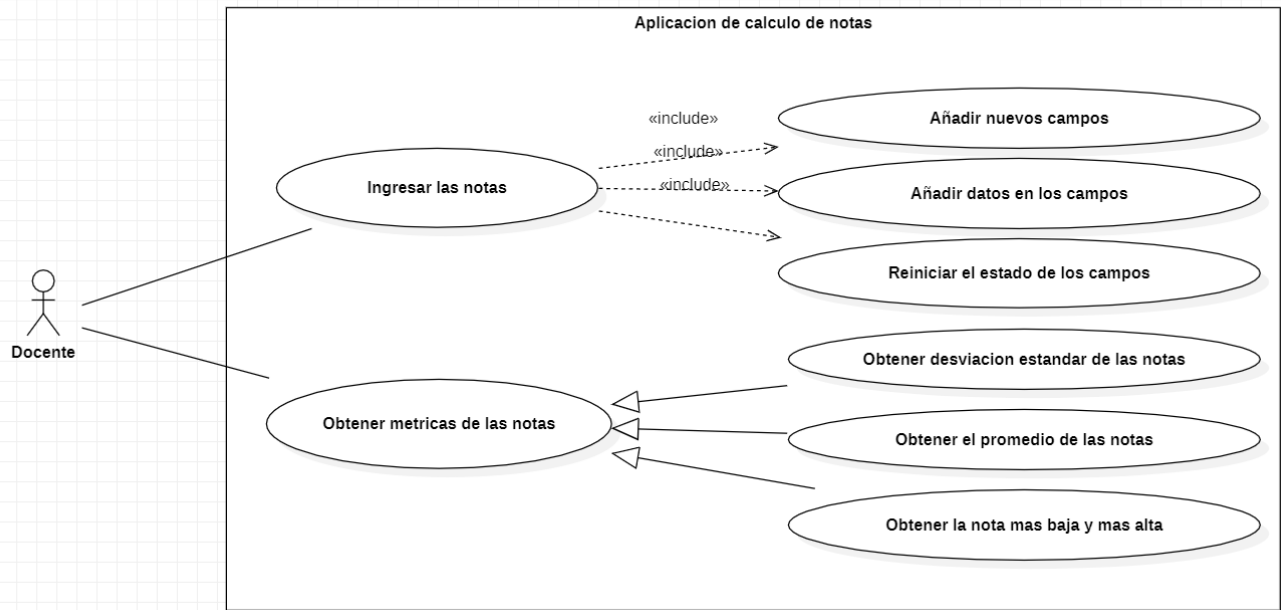



Diagrama de caso de uso

El archivo original del diagrama para el programa **StarUML** se encuentra disponible en el [GitHub](#).



Capturas de pantalla

Sin datos



—

□

×

Formulario de notas

Nota 1

0.0

Nota 2

0.0

Nota 3

0.0

Nota 4

0.0

Nota 5

0.0

Calcular

Añadir campo

Resetear campos

Metricas de notas


Metricas

Promedio: 0.0

Desviacion estandar: 0.0

Nota mas alta: 0.0

Nota mas baja: 0.0



—

□

×

Formulario de notas

Nota 1

0.0

Nota 2

0.0

Nota 3

0.0

Nota 4

0.0

Nota 5

0.0

Nota 6

0.0

Nota 7

0.0

Nota 8

0.0

Nota 9

0.0

Calcular

Añadir campo

Resetear campos

Métricas de notas

Métricas

Promedio: 0.0

Desviacion estandar: 0.0

Nota mas alta: 0.0

Nota mas baja: 0.0

Con datos



Formulario de notas

Nota 1

4

Nota 2

5

Nota 3

3

Nota 4

2.1

Nota 5

1.2

Nota 6

4.3

Nota 7

5

Nota 8

5

Nota 9

4.9

Calcular

Añadir campo

Resetear campos

Metricas de notas

Metricas

Promedio: 3.833333333333335

Desviacion estandar: 1.4934011227999822

Nota mas alta: 5.0

Nota mas baja: 1.2



Formulario de notas

Nota 1

4

Nota 2

4.3

Nota 3

5

Nota 4

5

Nota 5

2.3

Calcular Añadir campo Resetear campos

Metricas de notas

Metricas

Promedio: 4.12

Desviacion estandar: 1.3336866198624024

Nota mas alta: 5.0

Nota mas baja: 2.3

Punto 2

Codigo

El código de este ejercicio también se encuentra disponible en el [GitHub](#).

```
import java.awt.BorderLayout;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.Arrays;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
```

```
interface Shape {
    public Double calculate_volume();
    public Double calculate_surface();
}
```

```
// Se implementa el patron de diseño subscriber para que los componentes de ui de
// la aplicacion que dependan
// del estado de las notas puedan ser notificados cuando estas cambien y actualicen
```

su estado en consecuencia.

```
interface Subscriber {  
    void Notify();  
}
```

```
interface Suscriptable {  
    void registerSubscriber(Subscriber subscriber);  
    void notifySubscribers();  
}
```

```
class Cylinder implements Suscriptable, Shape {  
    private Double radius;  
    private Double height;  
    private ArrayList<Subscriber> subscribers;  
  
    public Cylinder(Double radius, Double height) {  
        this.radius = radius;  
        this.height = height;  
        subscribers = new ArrayList<Subscriber>();  
    }  
  
    // Calcula el volumen de la forma  
    @Override  
    public Double calculate_volume() {  
        return Math.PI * height * Math.pow(radius, 2.0);  
    }  
  
    // Calcula la superficie de la forma  
    @Override  
    public Double calculate_surface() {  
        return (2.0 * Math.PI * radius * height) + (2.0 * Math.PI *  
Math.pow(radius, 2.0));  
    }  
  
    // Registra un suscriptor a la forma  
    @Override  
    public void registerSubscriber(Subscriber subscriber) {  
        if (Arrays.stream(subscribers.toArray(new  
Subscriber[subscribers.size()])).filter((Subscriber sub) -> sub ==  
subscriber).count() > 0) return;  
        subscribers.add(subscriber);  
    }  
  
    // Notifica a todos los suscriptores de la forma que esta ha cambiado su estado  
    @Override  
    public void notifySubscribers() {
```

```
        Arrays.stream(subscribers.toArray(new
Subscriber[subscribers.size()])).forEach((Subscriber sub) -> sub.Notify());
    }
```

// Getters y setters de los atributos de la forma, se implementa este patron para que los componentes de UI

// no puedan cambiar el estado directamente y que esta accion pueda ser delegada a la clase, que en consecuencia debe notificar

// a los suscriptores de que ha cambiado su estado

```
public Double getRadius() {
    return radius;
}
```

```
public void setRadius(Double radius) {
    this.radius = radius;
    notifySubscribers();
}
```

```
public Double getHeight() {
    return height;
}
```

```
public void setHeight(Double height) {
    this.height = height;
    notifySubscribers();
}
}
```

```
class CylinderWindow extends JFrame implements Subscriber, ActionListener {
    private Cylinder cylinder;
    private JTextField radius_field;
    private JTextField height_field;
    private JLabel volume_label;
    private JLabel surface_label;
    private GridBagLayout layout;
```

```
    public CylinderWindow(Cylinder cylinder) {
        // Definimos el layout del contenedor de la ventana (creado en
        initializeComponents)
        layout = new GridBagLayout();
        this.cylinder = cylinder;
        // Nos registramos a la forma para que esta nos notifique cuando cambie su
        estado
        cylinder.registerSubscriber(this);
    }
```



```

        // Delegamos el layout al container que se crea en initializeComponents
para que este se expanda por toda la ventana
        setLayout(new BorderLayout());
        setBounds(0, 0, 500, 250);
        // Inicializamos los componentes de la ventana
        initializeComponents();
        // Inicializamos los valores de las metricas con el valor por defecto que
tiene la forma
        this.Notify();
    }

    private void initializeComponents() {
        // Creamos un contenedor que tendra todos los componentes de la ventana
        JPanel container = new JPanel(layout);

container.setBorder(javax.swing.BorderFactory.createTitledBorder("Cilindro"));

        // Contenedor de todo lo relacionado al input de los parametros de la forma
        JPanel form = new JPanel();
        form.setLayout(new GridBagLayout());

        JPanel radius_form_field_panel = new JPanel(new BorderLayout());

radius_form_field_panel.setBorder(javax.swing.BorderFactory.createTitledBorder("Rad
io"));
        radius_field = new JTextField(cylinder.getRadius().toString());
        radius_form_field_panel.add(radius_field);

        JPanel height_form_field_panel = new JPanel(new BorderLayout());

height_form_field_panel.setBorder(javax.swing.BorderFactory.createTitledBorder("Alt
ura"));
        height_field = new JTextField(cylinder.getHeight().toString());
        height_form_field_panel.add(height_field);

        GridBagConstraints constraints = new GridBagConstraints();
        constraints.fill = GridBagConstraints.HORIZONTAL;
        constraints.weightx = 1;
        constraints.gridx = 0;
        constraints.gridy = 0;

        form.add(radius_form_field_panel, constraints);

        constraints.gridy += 1;

        form.add(height_form_field_panel, constraints);

```

```
constraints.gridy = 0;

container.add(form, constraints);

// Contenedor para botones en caso de que se desee añadir multiples botones
JPanel buttons = new JPanel();
buttons.setLayout(new GridBagLayout());

JButton calculate_button = new JButton("Calcular");
calculate_button.setActionCommand("calculate");
calculate_button.addActionListener(this);

constraints.weighty = 0.8;
constraints.gridy = 0;
constraints.gridx = 0;

buttons.add(calculate_button, constraints);

constraints.weighty = 1;
constraints.gridx = 0;
constraints.gridy = 1;

container.add(buttons, constraints);

// Contiene las metricas de la forma (volumen y superficie)
JPanel metrics = new JPanel();

metrics.setBorder(javax.swing.BorderFactory.createTitledBorder("Metricas"));
metrics.setLayout(new GridBagLayout());

volume_label = new JLabel("");
surface_label = new JLabel("");

constraints.gridy = 0;
constraints.gridx = 0;

metrics.add(volume_label, constraints);

constraints.gridy += 1;

metrics.add(surface_label, constraints);

constraints.gridy = 2;

container.add(metrics, constraints);
```

```

        add(container);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        switch (e.getActionCommand()) {
            case "calculate":
                // Actualizamos el estado de la forma con los valores que se
                // encuentran en los campos de texto,
                // no es necesario manejar ningun cambio en la UI ya que esto se
                // hara desde Notify cuando la forma notifique
                // sobre el cambio en su estado
                cylinder.setRadius(Double.parseDouble(radius_field.getText()));
                cylinder.setHeight(Double.parseDouble(height_field.getText()));
                break;
            default:
                break;
        }
    }

    @Override
    public void Notify() {
        // Actualizamos el texto de las metricas con el valor que retorna la forma
        // al calcularlas
        volume_label.setText("Volumen: " + cylinder.calculate_volume().toString());
        surface_label.setText("Superficie: " +
        cylinder.calculate_surface().toString());
    }
}

class Sphere implements Suscriptable, Shape {
    private Double radius;
    private ArrayList<Subscriber> subscribers;

    public Sphere(Double radius) {
        this.radius = radius;
        this.subscribers = new ArrayList<Subscriber>();
    }

    // Calcula el volumen de la forma
    @Override
    public Double calculate_volume() {
        return (4/3) * Math.PI * Math.pow(this.radius, 3.0);
    }
}

```

```

// Calcula la superficie de la forma
@Override
public Double calculate_surface() {
    return 4.0 * Math.PI * Math.pow(this.radius, 2.0);
}

// Registra un suscriptor a la forma
@Override
public void registerSubscriber(Subscriber subscriber) {
    if (Arrays.stream(subscribers.toArray(new
Subscriber[subscribers.size()])).filter((Subscriber sub) -> sub ==
subscriber).count() > 0) return;
    subscribers.add(subscriber);
}

// Notifica a todos los suscriptores de la forma que esta ha cambiado su estado
@Override
public void notifySubscribers() {
    Arrays.stream(subscribers.toArray(new
Subscriber[subscribers.size()])).forEach((Subscriber sub) -> sub.Notify());
}

// Getters y setters de los atributos de la forma, se implementa este patron
para que los componentes de UI
// no puedan cambiar el estado directamente y que esta accion pueda ser
delegada a la clase, que en consecuencia debe notificar
// a los suscriptores de que ha cambiado su estado
public Double getRadius() {
    return radius;
}

public void setRadius(Double radius) {
    this.radius = radius;
    notifySubscribers();
}
}

```

```

class SphereWindow extends JFrame implements Subscriber, ActionListener {
    private Sphere sphere;
    private JTextField radius_field;
    private JLabel volume_label;
    private JLabel surface_label;
    private GridBagLayout layout;
}

```

```

    public SphereWindow(Sphere sphere) {
        // Definimos el layout del contenedor de la ventana (creado en
        initializeComponents)
        layout = new GridBagLayout();
        this.sphere = sphere;
        // Nos registramos a la forma para que esta nos notifique cuando cambie su
        estado
        sphere.registerSubscriber(this);
        // Delegamos el layout al container que se crea en initializeComponents
        para que este se expanda por toda la ventana
        setLayout(new BorderLayout());
        setBounds(0, 0, 500, 200);
        // Inicializamos los componentes de la ventana
        initializeComponents();
        // Inicializamos los valores de las metricas con el valor por defecto que
        tiene la forma
        this.Notify();
    }

```

```

    private void initializeComponents() {
        // Creamos un contenedor que tendra todos los componentes de la ventana
        JPanel container = new JPanel(layout);

        container.setBorder(javax.swing.BorderFactory.createTitledBorder("Esfera"));

        // Contenedor de todo lo relacionado al input de los parametros de la forma
        JPanel form = new JPanel();
        form.setLayout(new GridBagLayout());

        JPanel radius_form_field_panel = new JPanel(new BorderLayout());

        radius_form_field_panel.setBorder(javax.swing.BorderFactory.createTitledBorder("Radio"));

        radius_field = new JTextField(sphere.getRadius().toString());
        radius_form_field_panel.add(radius_field);

        GridBagConstraints constraints = new GridBagConstraints();
        constraints.fill = GridBagConstraints.HORIZONTAL;
        constraints.weightx = 1;
        constraints.gridx = 0;
        constraints.gridy = 0;

        form.add(radius_form_field_panel, constraints);
    }

```

```

constraints.gridy = 0;
container.add(form, constraints);

// Contenedor para botones en caso de que se desee añadir multiples botones
JPanel buttons = new JPanel();

buttons.setLayout(new GridBagLayout());

JButton calculate_button = new JButton("Calcular");
calculate_button.setActionCommand("calculate");
calculate_button.addActionListener(this);
constraints.weighty = 0.8;
constraints.gridy = 0;
constraints.gridx = 0;
buttons.add(calculate_button, constraints);

constraints.weighty = 1;
constraints.gridx = 0;
constraints.gridy = 1;

container.add(buttons, constraints);
// Contiene las metricas de la forma (volumen y superficie)
JPanel metrics = new JPanel();

metrics.setBorder(javax.swing.BorderFactory.createTitledBorder("Metricas"));

metrics.setLayout(new GridBagLayout());

volume_label = new JLabel("");
surface_label = new JLabel("");

constraints.gridy = 0;
constraints.gridx = 0;
metrics.add(volume_label, constraints);
constraints.gridy += 1;
metrics.add(surface_label, constraints);

constraints.gridy = 2;
container.add(metrics, constraints);
add(container);
}

@Override
public void actionPerformed(ActionEvent e) {

```

```

        switch (e.getActionCommand()) {
            case "calculate":
                // Actualizamos el estado de la forma con los valores que se
                // encuentran en los campos de texto,
                // no es necesario manejar ningun cambio en la UI ya que esto se
                // hara desde Notify cuando la forma notifique
                // sobre el cambio en su estado
                sphere.setRadius(Double.parseDouble(radius_field.getText()));
                break;
            default:
                break;
        }
    }

    @Override
    public void Notify() {
        // Actualizamos el texto de las metricas con el valor que retorna la forma
        // al calcularlas
        volume_label.setText("Volumen: " + sphere.calculate_volume().toString());
        surface_label.setText("Superficie: " +
        sphere.calculate_surface().toString());
    }
}

class Pyramid implements Suscriptable, Shape {
    private Double base;
    private Double height;
    private Double apothem;
    private ArrayList<Subscriber> subscribers;

    public Pyramid(Double base, Double height, Double apothem) {
        this.base = base;
        this.height = height;
        this.apothem = apothem;
        this.subscribers = new ArrayList<Subscriber>();
    }

    // Calcula el volumen de la forma
    @Override
    public Double calculate_volume() {
        return (Math.pow(base, 2.0) * height) / 3.0;
    }

    // Calcula la superficie de la forma
    @Override
    public Double calculate_surface() {

```

```

        return Math.pow(base, 2.0) + (base * apothem * 2.0);
    }

    // Registra un suscriptor a la forma
    @Override
    public void registerSubscriber(Subscriber subscriber) {
        if (Arrays.stream(subscribers.toArray(new
Subscriber[subscribers.size()])).filter((Subscriber sub) -> sub ==
subscriber).count() > 0) return;
        subscribers.add(subscriber);
    }

    // Notifica a todos los suscriptores de la forma que esta ha cambiado su estado
    @Override
    public void notifySubscribers() {
        Arrays.stream(subscribers.toArray(new
Subscriber[subscribers.size()])).forEach((Subscriber sub) -> sub.Notify());
    }

    // Getters y setters de los atributos de la forma, se implementa este patron
para que los componentes de UI
    // no puedan cambiar el estado directamente y que esta accion pueda ser
delegada a la clase, que en consecuencia debe notificar
    // a los suscriptores de que ha cambiado su estado
    public Double getBase() {
        return base;
    }

    public void setBase(Double base) {
        this.base = base;
        notifySubscribers();
    }

    public Double getHeight() {
        return height;
    }

    public void setHeight(Double height) {
        this.height = height;
        notifySubscribers();
    }

    public Double getApothem() {
        return apothem;
    }

```



```

    public void setApothem(Double apothem) {
        this.apothem = apothem;
        notifySubscribers();
    }
}

```

```

class PyramidWindow extends JFrame implements Subscriber, ActionListener {
    private Pyramid pyramid;
    private JTextField base_field;
    private JTextField height_field;
    private JTextField apothem_field;
    private JLabel volume_label;
    private JLabel surface_label;
    private GridBagLayout layout;

    public PyramidWindow(Pyramid pyramid) {
        // Definimos el layout del contenedor de la ventana (creado en
        initializeComponents)
        layout = new GridBagLayout();
        this.pyramid = pyramid;
        // Nos registramos a la forma para que esta nos notifique cuando cambie su
        estado
        pyramid.registerSubscriber(this);

        // Delegamos el layout al container que se crea en initializeComponents
        para que este se expanda por toda la ventana
        setLayout(new BorderLayout());
        setBounds(0, 0, 500, 275);
        // Inicializamos los componentes de la ventana
        initializeComponents();
        // Inicializamos los valores de las metricas con el valor por defecto que
        tiene la forma
        this.Notify();
    }

    private void initializeComponents() {
        // Creamos un contenedor que tendra todos los componentes de la ventana
        JPanel container = new JPanel(layout);

        container.setBorder(javax.swing.BorderFactory.createTitledBorder("Piramide"));

        // Contenedor de todo lo relacionado al input de los parametros de la forma
        JPanel form = new JPanel();
        form.setLayout(new GridBagLayout());
    }
}

```

```

        JPanel base_form_field_panel = new JPanel(new BorderLayout());

base_form_field_panel.setBorder(javax.swing.BorderFactory.createTitledBorder("Base"
));

        base_field = new JTextField(pyramid.getBase().toString());
        base_form_field_panel.add(base_field);

        JPanel height_form_field_panel = new JPanel(new BorderLayout());

height_form_field_panel.setBorder(javax.swing.BorderFactory.createTitledBorder("Alt
ura"));
        height_field = new JTextField(pyramid.getHeight().toString());
        height_form_field_panel.add(height_field);

        JPanel apothem_form_field_panel = new JPanel(new BorderLayout());

apothem_form_field_panel.setBorder(javax.swing.BorderFactory.createTitledBorder("Ap
otema"));
        apothem_field = new JTextField(pyramid.getApothem().toString());
        apothem_form_field_panel.add(apothem_field);

        GridBagConstraints constraints = new GridBagConstraints();
        constraints.fill = GridBagConstraints.HORIZONTAL;
        constraints.weightx = 1;
        constraints.gridx = 0;
        constraints.gridy = 0;

        form.add(base_form_field_panel, constraints);

        constraints.gridy += 1;

        form.add(height_form_field_panel, constraints);

        constraints.gridy += 1;

        form.add(apothem_form_field_panel, constraints);

        constraints.gridy = 0;

        container.add(form, constraints);

// Contenedor para botones en caso de que se desee añadir multiples botones
JPanel buttons = new JPanel();
buttons.setLayout(new GridBagLayout());

```

```

        JButton calculate_button = new JButton("Calcular");
        calculate_button.setActionCommand("calculate");
        calculate_button.addActionListener(this);

        constraints.weighty = 0.8;
        constraints.gridy = 0;
        constraints.gridx = 0;
        buttons.add(calculate_button, constraints);

        constraints.weighty = 1;
        constraints.gridx = 0;
        constraints.gridy = 1;
        container.add(buttons, constraints);

        // Contiene las metricas de la forma (volumen y superficie)
        JPanel metrics = new JPanel();

        metrics.setBorder(javax.swing.BorderFactory.createTitledBorder("Metricas"));
        metrics.setLayout(new GridBagLayout());

        volume_label = new JLabel("");
        surface_label = new JLabel("");

        constraints.gridy = 0;
        constraints.gridx = 0;

        metrics.add(volume_label, constraints);

        constraints.gridy += 1;

        metrics.add(surface_label, constraints);

        constraints.gridy = 2;

        container.add(metrics, constraints);
        add(container);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        switch (e.getActionCommand()) {
            case "calculate":
                // Actualizamos el estado de la forma con los valores que se
                // encuentran en los campos de texto,
                // no es necesario manejar ningun cambio en la UI ya que esto se
                // hara desde Notify cuando la forma notifique

```

```

        // sobre el cambio en su estado
        pyramid.setBase(Double.parseDouble(base_field.getText()));
        pyramid.setHeight(Double.parseDouble(height_field.getText()));
        pyramid.setApothem(Double.parseDouble(apothem_field.getText()));
        break;
    default:
        break;
    }
}

@Override
public void Notify() {
    // Actualizamos el texto de las metricas con el valor que retorna la forma
    al calcularlas
    volume_label.setText("Volumen: " + pyramid.calculate_volume().toString());
    surface_label.setText("Superficie: " +
pyramid.calculate_surface().toString());
}
}

class MenuWindow extends JFrame implements ActionListener {
    private Cylinder cylinder;
    private CylinderWindow cylinderWindow;
    private Sphere sphere;
    private SphereWindow sphereWindow;
    private Pyramid pyramid;
    private PyramidWindow pyramidWindow;

    private GridBagLayout layout;

    public MenuWindow() {
        // Se definen las formas que se van a utilizar en la aplicacion y sus
        respectivas ventanas,
        // se prefiere este metodo sobre su creacion en el constructor ya que se
        puede centralizar el manejo
        // del estado de la aplicacion
        // Esto es util en caso de que se desee serializar, tambien permite la
        persistencia de los datos de
        // las formas entre ventanas y modificacion de las mismas independiente del
        contexto (ventana)
        // en el que se encuentre el usuario, ademas de la ventaja mas
        significativa que es la de
        // desvincular por completo el manejo de estado de la aplicacion de la capa
        del cliente (UI)
        cylinder = new Cylinder(0.0, 0.0);
        cylinderWindow = new CylinderWindow(cylinder);

```

```

    sphere = new Sphere(0.0);
    sphereWindow = new SphereWindow(sphere);
    pyramid = new Pyramid(0.0, 0.0, 0.0);
    pyramidWindow = new PyramidWindow(pyramid);

    this.layout = new GridBagLayout();

    setBounds(0, 0, 500, 200);
    setLayout(layout);
    initializeComponents();
}

private void initializeComponents() {
    JButton cylinderButton = new JButton("Cilindro");
    cylinderButton.setActionCommand("cylinder");

    JButton sphereButton = new JButton("Esfera");
    sphereButton.setActionCommand("sphere");

    JButton pyramidButton = new JButton("Piramide");
    pyramidButton.setActionCommand("pyramid");

    cylinderButton.addActionListener(this);
    sphereButton.addActionListener(this);
    pyramidButton.addActionListener(this);

    GridBagConstraints constraints = new GridBagConstraints();
    constraints.fill = GridBagConstraints.HORIZONTAL;
    constraints.gridx = 0;
    constraints.gridy = 0;

    this.add(cylinderButton, constraints);
    constraints.gridx += 1;
    this.add(sphereButton, constraints);
    constraints.gridx += 1;
    this.add(pyramidButton, constraints);
}

@Override
public void actionPerformed(ActionEvent e) {
    switch (e.getActionCommand()) {
        case "cylinder":
            this.cylinderWindow.setVisible(true);
            break;
        case "sphere":
            this.sphereWindow.setVisible(true);

```

```

        break;
    case "pyramid":
        this.pyramidWindow.setVisible(true);
        break;
    default:
        break;
    }
}

}

}

class Main {
    public static MenuWindow menuWindow;
    public static void main(String[] args) {
        menuWindow = new MenuWindow();
        menuWindow.setVisible(true);
    }
}

```

Diagrama de clases

El archivo original del diagrama para el programa **StarUML** se encuentra disponible en el [GitHub](#).

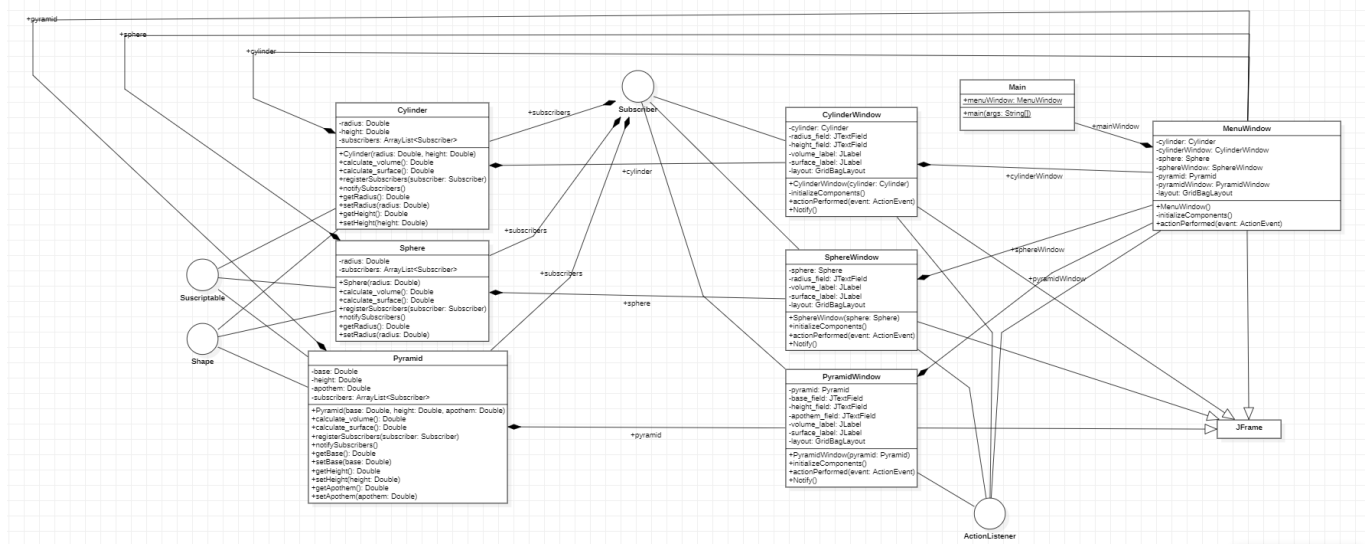
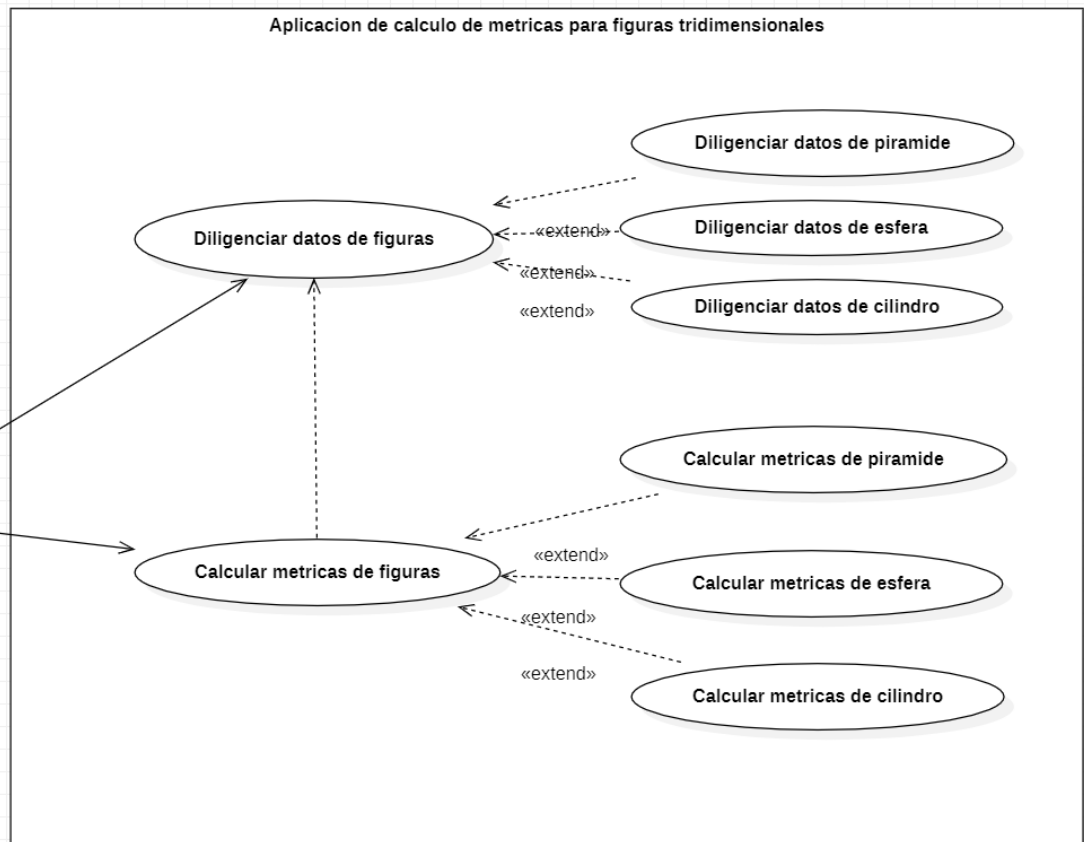


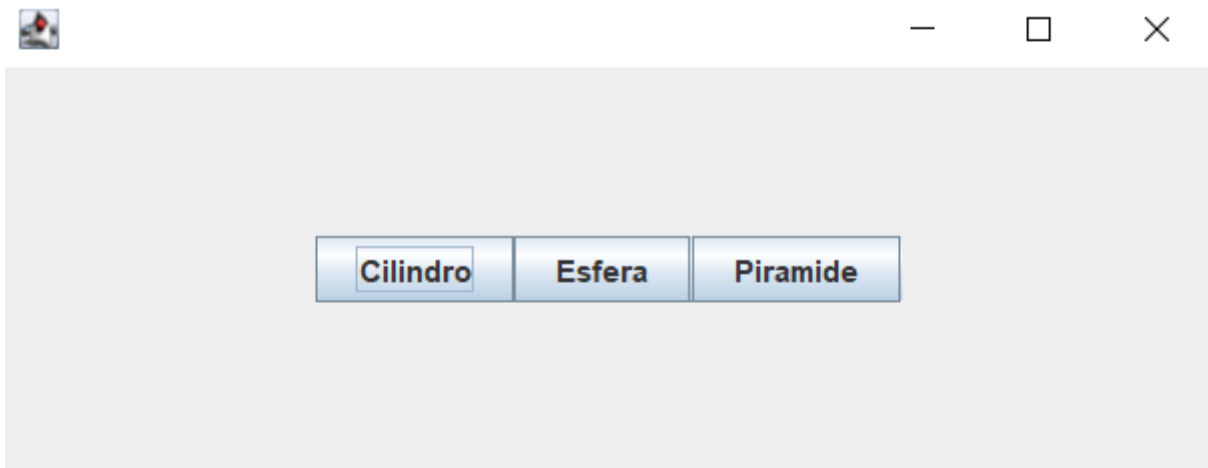
Diagrama de caso de uso

El archivo original del diagrama para el programa **StarUML** se encuentra disponible en el [GitHub](#).




Capturas de pantalla

Ventana principal



Sin datos

— □ ×


Cilindro

Radio

Altura

Calcular

Metricas
Volumen: 0.0
Superficie: 0.0


— □ ×

Esfera

Radio

Calcular

Metricas
Volumen: 0.0
Superficie: 0.0

— □ ×

Piramide

Base


Altura

Apotema

Calcular

Metricas
Volumen: 0.0
Superficie: 0.0

Con datos

—□×


Cilindro

Radio

Altura

Calcular

Metricas
Volumen: 1570.7963267948965
Superficie: 942.4777960769379


—□×

Esfera

Radio

Calcular

Metricas
Volumen: 10602.875205865552
Superficie: 2827.4333882308138

—□×

Piramide

Base

Altura

Apotema

Calcular

Metricas
Volumen: 1000.0
Superficie: 380.0