

Estudiante : \_\_**Nicole Carvajal y Eduardo Jirón**\_\_ Carne: \_\_**2017098785 y 2017101878**\_\_ Nota\_\_\_\_\_

### Instrucciones Generales

La prueba se libera el día domingo 02 de junio al final de la tarde y se entrega el martes 04 de junio al final del día. Puede ser desarrollada en forma individual o en PAREJAS.

Debe ser respondida en este documento y ser entregada en formato PDF. El documento debe llamarse **IIIPARCIAL\_SusNombre.PDF** y ser subido en el TEC Digital en el apartado de **Evaluaciones>IIIParcial**.

Se penalizarán con 5 puntos de la nota final obtenida si esta instrucción no se sigue en la forma solicitada.

En los casos que se les solicite un diagrama de clases debe colocar la imagen en dentro de la prueba. No se aceptarán archivos adicionales.

Se revisará únicamente las respuestas que se coloquen en los espacios destinados para las mismas.

Cualquier otro comentario que se añada en lugares no solicitados, serán ignorados.

### Cuadro de evaluación

I Parte Selección Unica	II Parte Desarrollo	III Parte Modelaje y Programación			Puntos
10	10	10	10	10	50

## I Parte Selección Única (10 puntos)

Del siguiente conjunto de patrones estudiados en clase, seleccione la que se ajusta de manera exacta a cada una de las afirmaciones que se muestran a continuación. Algunas de estas afirmaciones pueden repetirse a un mismo patrón

Cadena de responsabilidad	Memento	Mediator	Command	Strategy	Interpreter
---------------------------	---------	----------	---------	----------	-------------

1. Permite definir un objeto que pueda encapsular la forma en que interactúan un conjunto de objetos asociados, por lo que al provocar bajo acoplamiento, la interacción entre ellos podría variar de manera independiente.  
\_\_\_\_\_Mediator\_\_\_\_\_
2. Patrón que permite crear casi un mini lenguaje para implementar la lógica del programa, a través de la interpretación de frases.  
\_\_\_\_\_Interpreter\_\_\_\_\_
3. Este patrón permite mantener un conjunto de algoritmos de los que el objeto cliente puede elegir aquel que le conviene e intercambiarlo según sus necesidades. El contexto o el cliente pueden elegir el algoritmo que prefiera de entre los disponibles el más apropiado para cada situación.  
\_\_\_\_\_Strategy\_\_\_\_\_
4. El patrón guarda parte o todo el estado interno de un objeto, sin romper el principio de encapsulamiento, con el fin de que este objeto pueda ser restaurado más tarde al estado guardado.  
\_\_\_\_\_Memento\_\_\_\_\_
5. Este patrón busca evitar el acoplar el emisor de una petición a su receptor, dando a más de un objeto la posibilidad de responder a la petición. \_\_\_\_\_Cadena de responsabilidad\_\_\_\_\_
6. Se puede considerar un caso particular de patrón Composite, pero dedicado específicamente a labores de parsing.  
\_\_\_\_\_Interpreter\_\_\_\_\_
7. Evita tener que acoplar el emisor de una petición con el receptor, dando a más de un objeto la posibilidad de responder a la petición \_\_\_\_\_Cadena de responsabilidad\_\_\_\_\_

8. La interacción de varios objetos en este patrón puede provocar que la estructura se torne bastante compleja, por lo que se debe establecer la forma en que dichos objetos interactuarán entre sí.

\_\_\_\_\_Mediator\_\_\_\_\_

9. El patrón sistematiza el uso de implementaciones alternativas.

\_\_\_\_\_Strategy\_\_\_\_\_

10. El patrón ofrece la posibilidad de parametrizar objetos para llevar a cabo acciones.

\_\_\_\_\_Command\_\_\_\_\_

## II Parte Desarrollo (10 puntos)

- a. **Mediator, Command y Chain of Responsibility** son patrones que desacoplan el proceso de comunicación entre sus partes.

Anote el mecanismo que propone cada patrón para lograr un contraste entre los tres patrones que permita reflejar las diferencias que existen entre ellos en cuanto a la forma en que administran la comunicación entre objetos.

Patrón	Detalle de cómo el patrón administra la comunicación entre objetos.
Mediator	Para utilizar este patrón la comunicación debe estar bien definida y seguir un orden. Su objetivo desacoplar los objetos que se comunican entre ellos a través de un mediador. Existe una necesidad de ordenar la comunicación entre las partes.
Command	La comunicación entre objetos es independiente, además este patrón son acciones a realizar a un objeto. Estandariza una orden, también desacopla los objetos de las acciones que realiza. Este patrón no se preocupa por el orden en que se realiza la comunicación.
Chain of Responsibility	Para manejar la comunicación entre objetos, Chain of Responsibility forma una cadena de manejadores de objetos en específico. Si el objeto de entrada no es recibido por un manejador pasa al siguiente y así sucesivamente hasta encontrar el adecuado o llegar al final de la cadena.

III Parte Modelaje y Programación (30 puntos, 10 puntos cada caso)

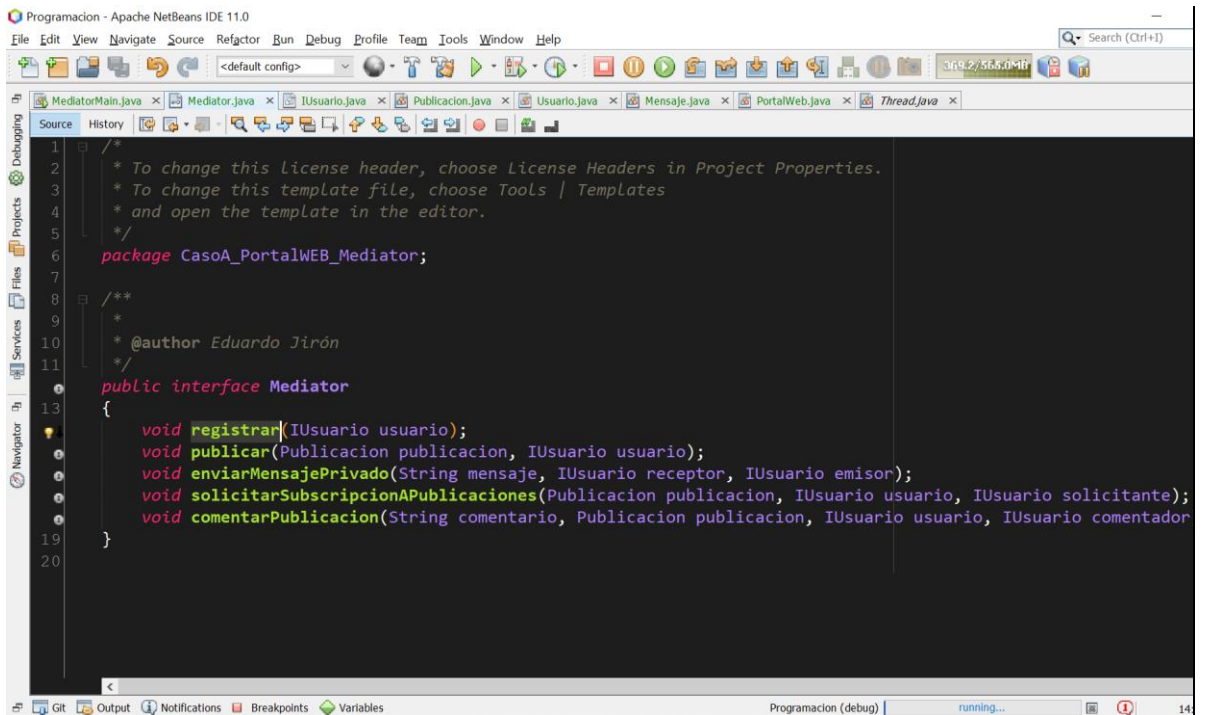
A. Suponga un portal web que permite el registro de usuarios, publicar contenidos en varios formatos (texto, imágenes/video, archivos, entre otros), un usuario puede enviar mensajes privados a otros usuarios, solicitar la suscripción a las publicaciones de otros usuarios, pueden realizar comentarios a de los otros usuarios o sus propias publicaciones.

Proponga la utilización de un patrón adecuado que maneje esta situación, justifique su respuesta, construya el modelo en notación UML 2.0 que soporte esta necesidad y programe el funcionamiento de su propuesta de solución en un proyecto usando el lenguaje de programación Java en un paquete llamado **CasoA\_PortalWEB\_Patron** (donde Patron es el nombre del patrón propuesto)

Interesa la propuesta de diseño del modelo de datos (no el patrón creacional que los genere) pero que se requiere para dar vida al patrón utilizado en su propuesta

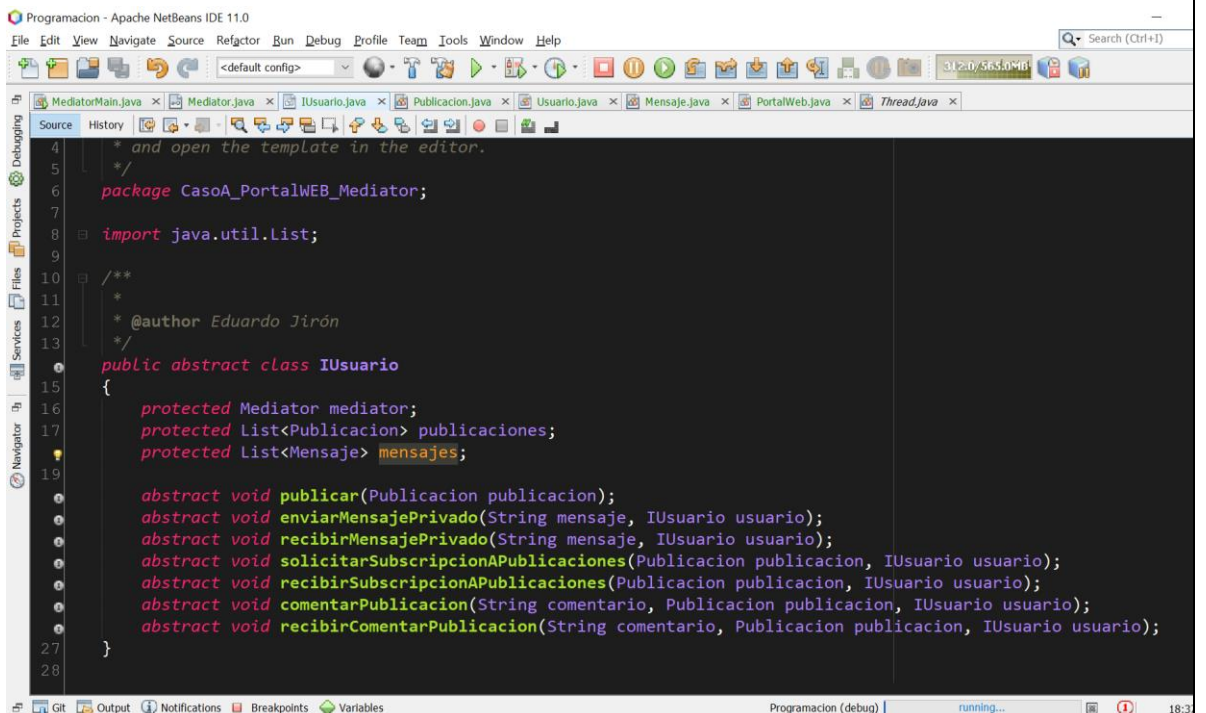
Patrón seleccionado (1 puntos) Mediator	Justificación (2 puntos) Es un mediator porque el portal web se encarga de manejar la comunicación entre los usuarios; sea publicaciones, mensajes, comentarios o suscripciones
Modelo UML 2.0 de implementación propuesta (3 puntos)	





```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6   package CasoA_PortalWEB_Mediator;
7
8   /**
9    *
10   * @author Eduardo Jirón
11   */
12   public interface Mediator
13   {
14       void registrar(IUsuario usuario);
15       void publicar(Publicacion publicacion, IUsuario usuario);
16       void enviarMensajePrivado(String mensaje, IUsuario receptor, IUsuario emisor);
17       void solicitarSubscripcionAPublicaciones(Publicacion publicacion, IUsuario usuario, IUsuario solicitante);
18       void comentarPublicacion(String comentario, Publicacion publicacion, IUsuario usuario, IUsuario comentador);
19   }
20
```

## IUsuario



```
4   * and open the template in the editor.
5   */
6   package CasoA_PortalWEB_Mediator;
7
8   import java.util.List;
9
10  /**
11   *
12   * @author Eduardo Jirón
13   */
14  public abstract class IUsuario
15  {
16      protected Mediator mediator;
17      protected List<Publicacion> publicaciones;
18      protected List<Mensaje> mensajes;
19
20      abstract void publicar(Publicacion publicacion);
21      abstract void enviarMensajePrivado(String mensaje, IUsuario usuario);
22      abstract void recibirMensajePrivado(String mensaje, IUsuario usuario);
23      abstract void solicitarSubscripcionAPublicaciones(Publicacion publicacion, IUsuario usuario);
24      abstract void recibirSubscripcionAPublicaciones(Publicacion publicacion, IUsuario usuario);
25      abstract void comentarPublicacion(String comentario, Publicacion publicacion, IUsuario usuario);
26      abstract void recibirComentarPublicacion(String comentario, Publicacion publicacion, IUsuario usuario);
27  }
28
```

## Publicacion

Programacion - Apache NetBeans IDE 11.0

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

<default config> 477.0/565.0MB

MediatorMain.java Mediator.java IUuario.java Publicacion.java Usuario.java Mensaje.java PortalWeb.java Thread.java

```
13  /* @author Eduardo Jirón
14  */
15  public class Publicacion
16  {
17      private Object contenido;
18      private List<String> comentarios;
19      private List<Usuario> subscriptores;
20
21      public Publicacion(Object contenido)
22      {
23          this.contenido = contenido;
24          this.comentarios = new ArrayList<>();
25          this.subscriptores = new ArrayList<>();
26      }
27
28      public void agregarComentario(String comentario)
29      {
30          comentarios.add(comentario);
31      }
32
33      public void agregarSubscriptor(Usuario subscriptor)
34      {
35          subscriptores.add(subscriptor);
36      }
37  }
```

Git Output Notifications Breakpoints Variables Programacion (debug) running... 30

## Usuario

Programacion - Apache NetBeans IDE 11.0

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

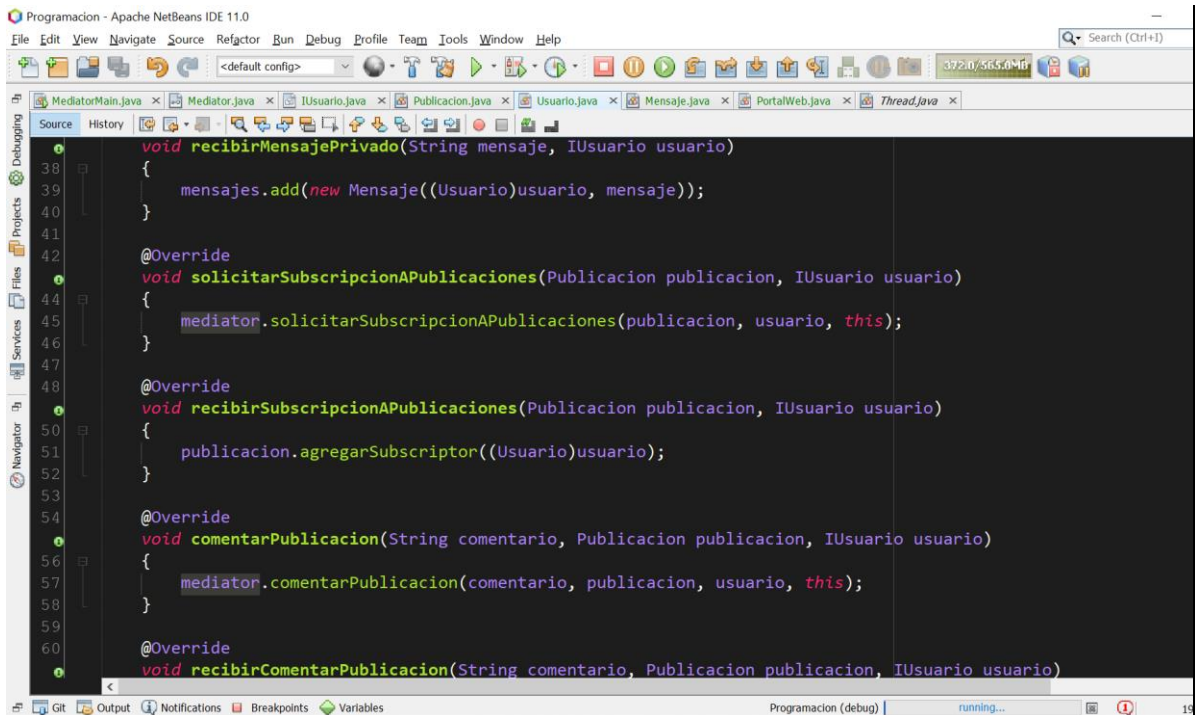
<default config> 477.0/565.0MB

MediatorMain.java Mediator.java IUuario.java Publicacion.java Usuario.java Mensaje.java PortalWeb.java Thread.java

```
13  /*
14  */
15  public class Usuario extends IUuario
16  {
17      public Usuario(Mediator mediator)
18      {
19          super();
20          super.mediator = mediator;
21          mensajes = new ArrayList<>();
22          publicaciones = new ArrayList<>();
23      }
24
25      @Override
26      void publicar(Publicacion publicacion)
27      {
28          publicaciones.add(publicacion);
29      }
30
31      @Override
32      void enviarMensajePrivado(String mensaje, IUuario usuario)
33      {
34          mediator.enviarMensajePrivado(mensaje, usuario, this);
35      }
36
37      @Override
38      void recibirMensajePrivado(String mensaje, IUuario usuario)
```

Git Output Notifications Breakpoints Variables Programacion (debug) running... 19





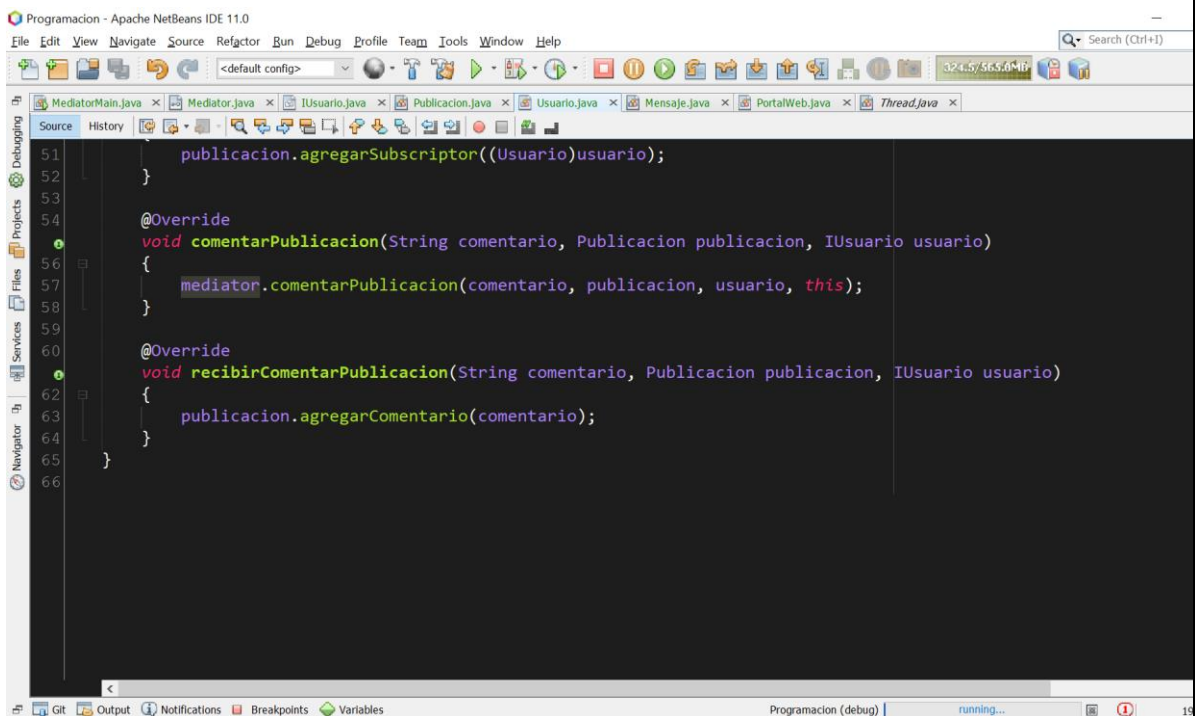
```
void recibirMensajePrivado(String mensaje, IUserario usuario)
{
    mensajes.add(new Mensaje((Usuario)usuario, mensaje));
}

@Override
void solicitarSubscripcionAPublicaciones(Publicacion publicacion, IUserario usuario)
{
    mediator.solicitarSubscripcionAPublicaciones(publicacion, usuario, this);
}

@Override
void recibirSubscripcionAPublicaciones(Publicacion publicacion, IUserario usuario)
{
    publicacion.agregarSubscriptor((Usuario)usuario);
}

@Override
void comentarPublicacion(String comentario, Publicacion publicacion, IUserario usuario)
{
    mediator.comentarPublicacion(comentario, publicacion, usuario, this);
}

@Override
void recibirComentarPublicacion(String comentario, Publicacion publicacion, IUserario usuario)
```



```
publicacion.agregarSubscriptor((Usuario)usuario);
}

@Override
void comentarPublicacion(String comentario, Publicacion publicacion, IUserario usuario)
{
    mediator.comentarPublicacion(comentario, publicacion, usuario, this);
}

@Override
void recibirComentarPublicacion(String comentario, Publicacion publicacion, IUserario usuario)
{
    publicacion.agregarComentario(comentario);
}
}
```

Mensaje

Programacion - Apache NetBeans IDE 11.0

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

<default config> 356.2/565.0MB

MediatorMain.java Mediator.java IUuario.java Publicacion.java Usuario.java Mensaje.java PortalWeb.java Thread.java

```
1  /**
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6   package CasoA_PortalWEB_Mediator;
7
8   /**
9   *
10  * @author Eduardo Jirón
11  */
12  public class Mensaje
13  {
14      private Usuario emisor;
15      private String mensaje;
16
17      public Mensaje(Usuario emisor, String mensaje)
18      {
19          this.emisor = emisor;
20          this.mensaje = mensaje;
21      }
22  }
23
```

Git Output Notifications Breakpoints Variables Programacion (debug) running... 20

## PortalWeb

Programacion - Apache NetBeans IDE 11.0

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

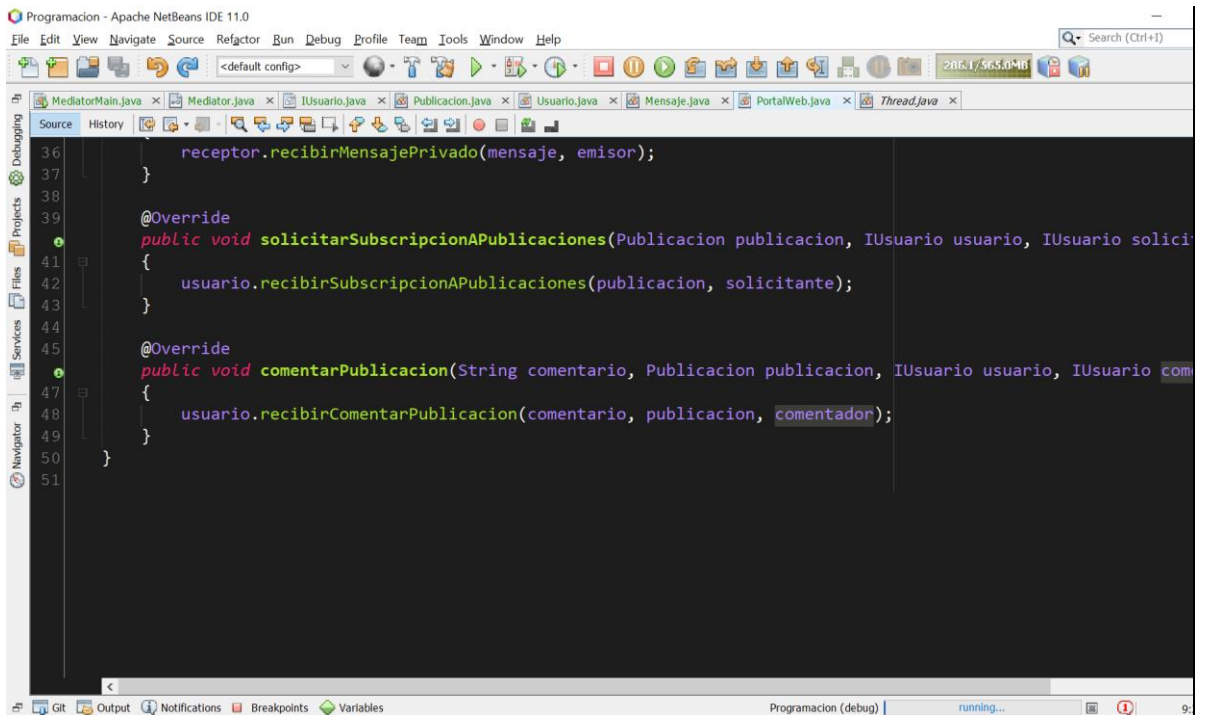
<default config> 289.0/565.0MB

MediatorMain.java Mediator.java IUuario.java Publicacion.java Usuario.java Mensaje.java PortalWeb.java Thread.java

```
15  public class PortalWeb implements Mediator
16  {
17      private List<Usuario> usuariosRegistrados;
18
19      @Override
20      public void registrar(IUsuario usuario)
21      {
22          if(usuariosRegistrados == null)
23              usuariosRegistrados = new ArrayList<>();
24          usuariosRegistrados.add((Usuario)usuario);
25      }
26
27      @Override
28      public void publicar(Publicacion publicacion, IUsuario usuario)
29      {
30          ((Usuario)usuario).publicar(publicacion);
31      }
32
33      @Override
34      public void enviarMensajePrivado(String mensaje, IUsuario receptor, IUsuario emisor)
35      {
36          receptor.recibirMensajePrivado(mensaje, emisor);
37      }
38
39      @Override

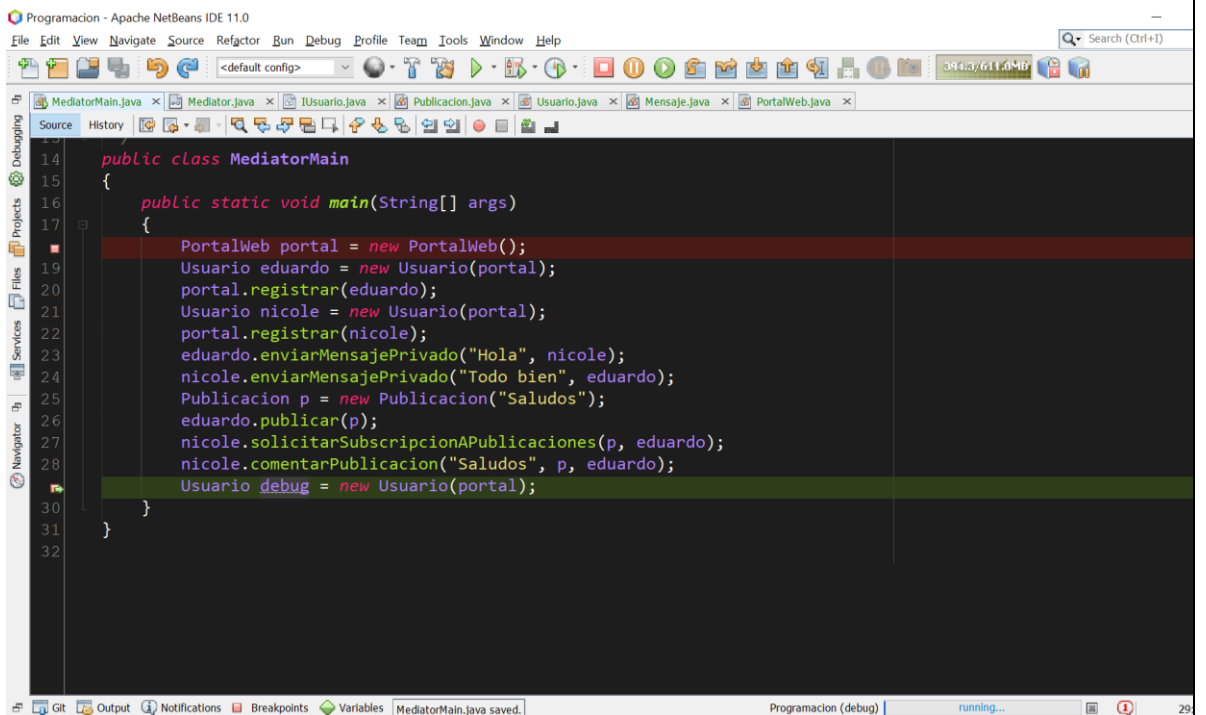
```

Git Output Notifications Breakpoints Variables Programacion (debug) running... 9:



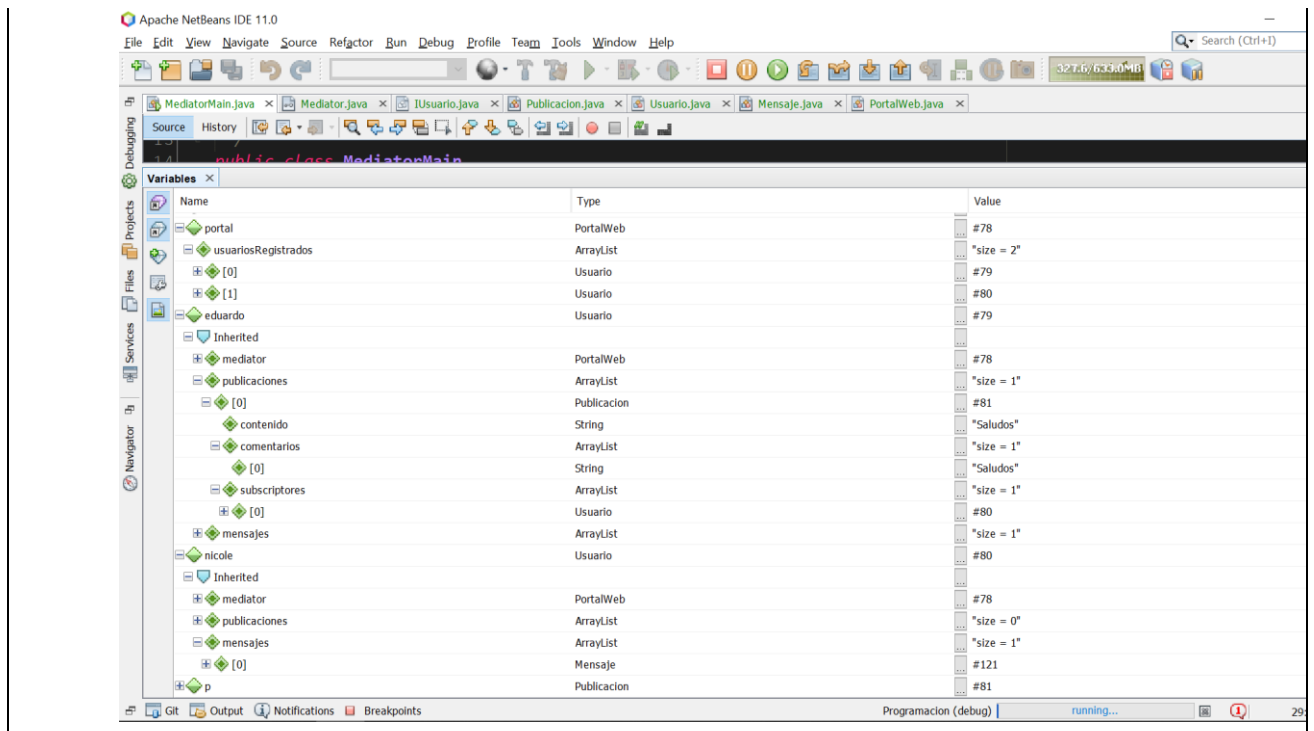
```
36         receptor.recibirMensajePrivado(mensaje, emisor);
37     }
38
39     @Override
40     public void solicitarSubscripcionAPublicaciones(Publicacion publicacion, IUsuario usuario, IUsuario solicitante) {
41         usuario.recibirSubscripcionAPublicaciones(publicacion, solicitante);
42     }
43
44     @Override
45     public void comentarPublicacion(String comentario, Publicacion publicacion, IUsuario usuario, IUsuario comentador) {
46         usuario.recibirComentarPublicacion(comentario, publicacion, comentador);
47     }
48
49 }
50
51
```

## MediatorMain



```
14 public class MediatorMain
15 {
16     public static void main(String[] args)
17     {
18         PortalWeb portal = new PortalWeb();
19         Usuario eduardo = new Usuario(portal);
20         portal.registrar(eduardo);
21         Usuario nicole = new Usuario(portal);
22         portal.registrar(nicole);
23         eduardo.enviarMensajePrivado("Hola", nicole);
24         nicole.enviarMensajePrivado("Todo bien", eduardo);
25         Publicacion p = new Publicacion("Saludos");
26         eduardo.publicar(p);
27         nicole.solicitarSubscripcionAPublicaciones(p, eduardo);
28         nicole.comentarPublicacion("Saludos", p, eduardo);
29         Usuario debug = new Usuario(portal);
30     }
31 }
32
```

## Resultados



- B. Considere un simulador orientado a eventos genérico, es decir un simulador esqueleto que simula un rango amplio de sistemas. El simulador contiene una cola de eventos y una variable que indica la hora de simulación (contador). Cada evento tiene grabado el tiempo que indica el momento en que éste debe ocurrir. La cola contiene eventos, los cuales son almacenados en orden ascendente respecto al tiempo en que el evento ocurre. El simulador orientado a eventos genérico ejecuta el siguiente ciclo infinito en el método **simular** de la clase SimuladorGenerico:

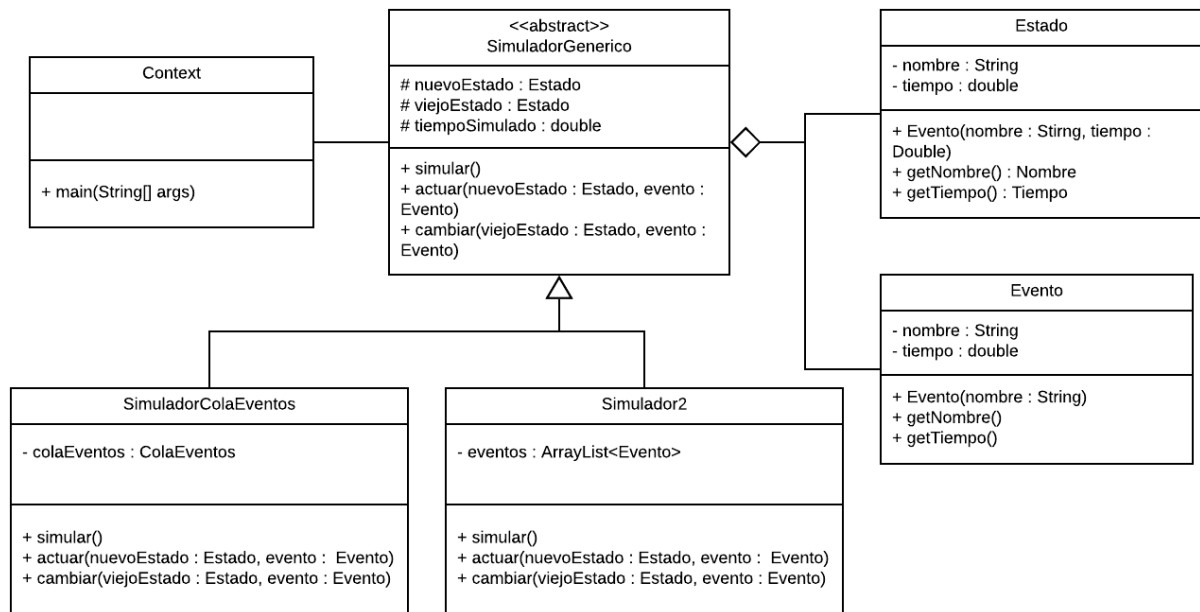
```
abstract class SimuladorGenerico
{
    protected Estado nuevoEstado, viejoEstado;
    protected double tiempoSimulado;
    protected ColaEventos colaEventos;
    protected Evento eventoActual;
    public abstract Estado cambiar(Estado viejoEstado, Evento evento);
    public abstract void actuar(Estado nuevoEstado, Estado viejoEstado);
    public void simular()
    {
        while (true)
        {
            tiempoSimulado = colaEventos.top().getTiempo();
            do
            {
                eventoActual = colaEventos.pop();
                nuevoEstado = cambiar(viejoEstado, eventoActual);
                actuar(nuevoEstado, viejoEstado);
                viejoEstado = nuevoEstado;
            } while(tiempoSimulado <= colaEventos.top().getTiempo());
        }
    }
}
```

Considere las siguientes cuestionantes y conteste el siguiente cuadro:

- Si pudiera sugerir la incorporación de un patrón de diseño que satisfaga la necesidad, ¿cuál sería?
- Justifique su análisis, incluya en esta sección una explicación de cómo sería el cambio en el código dado para incorporar su propuesta?
- Muestre el diagrama del patrón sugerido usando la notación UML 2.0
- Programe el funcionamiento de su propuesta de solución en un proyecto usando el lenguaje de programación Java en un paquete llamado **CasoB\_Simulador\_Patron** (donde Patron es el nombre del patrón propuesto). Aporte muestras visuales (screenshoots) del nuevo código según su propuesta y muestras de funcionamiento que solucionan esta inquietud

Patrón seleccionado (1 puntos) Strategy	Justificación (2 puntos) El problema busca permitir distintas implementaciones del método simular y para que el usuario pueda escoger el que le sirva. El cambio principal es hacer el método simular un método abstracto y crear una implementación concreta con el contenido de este método.
--------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

### Modelo UML 2.0 de implementación propuesta (3 puntos)



### Screenshots del código generado y muestras de funcionamiento adecuado del patrón. (5 puntos)

```

public abstract class SimuladorGenerico {
    protected Estado nuevoEstado;
    protected Estado viejoEstado;
    protected Evento eventoActual;
    protected double tiempoSimulado;

    public abstract Estado cambiar(Estado viejoEstado, Evento evento);
    public abstract void actuar(Estado nuevoEstado, Estado viejoEstado);
    public abstract void simular();
}

```

```
public class SimuladorColaEventos extends SimuladorGenerico{
```

```
    private Queue<Evento> colaEventos;
```

```
    @Override
```

```
    public Estado cambiar(Estado viejoEstado, Evento evento) {
        Estado nuevoEstado = new Estado(evento.getNombre(), viejoEstado.getTiempo() + 1);
        return nuevoEstado;
    }
```

```
    @Override
```

```
    public void actuar(Estado nuevoEstado, Estado viejoEstado) {
        System.out.println("Se cambia del estado " + viejoEstado.getNombre() + " al estado " + nuevoEstado.getNombre());
    }
```

```
    @Override
```

```
    public void simular() {
        viejoEstado = new Estado("Inicio", 0.0d);

        tiempoSimulado = colaEventos.peek().getTiempo();
        do{
            eventoActual = colaEventos.poll();
            nuevoEstado = cambiar(viejoEstado, eventoActual);
            actuar(nuevoEstado, viejoEstado);
            viejoEstado = nuevoEstado;
        }while(!colaEventos.isEmpty());
    }
```

```
    public SimuladorColaEventos(Queue<Evento> cola){
        this.colaEventos = cola;
    }
}
```

```
public class Simulador2 extends SimuladorGenerico {
```

```
    ArrayList<Evento> eventos = new ArrayList<>();
```

```
    @Override
```

```
    public Estado cambiar(Estado viejoEstado, Evento evento) {
        Estado nuevoEstado = new Estado(evento.getNombre(), viejoEstado.getTiempo() + 2);
        return nuevoEstado;
    }
```

```
    @Override
```

```
    public void actuar(Estado nuevoEstado, Estado viejoEstado) {
        System.out.println("Se cambia del estado " + viejoEstado.getNombre() + " al estado " + nuevoEstado.getNombre() );
    }
```

```
    @Override
```

```
    public void simular() {
        viejoEstado = new Estado("Inicial", 0.0d);
        for(Evento e : eventos){
            eventoActual = e;
            nuevoEstado = cambiar(viejoEstado, eventoActual);
            actuar(nuevoEstado, viejoEstado);
            viejoEstado = nuevoEstado;
        }
    }
```

```
    public Simulador2(ArrayList<Evento> eventos){
        this.eventos = eventos;
    }
}
```

```
public class Estado {  
  
    private String nombre;  
    private double tiempo;  
  
    public Estado(String nombre, double tiempo){  
        this.nombre = nombre;  
        this.tiempo = tiempo;  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public Double getTiempo() {  
        return tiempo;  
    }  
  
}
```



```
public class Evento {  
  
    private String nombre;  
    private double tiempo;  
  
    public Evento(String nombre, double tiempo){  
        this.nombre = nombre;  
        this.tiempo = tiempo;  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public Double getTiempo() {  
        return tiempo;  
    }  
}
```

```

public class Context {

    public static void main (String[] args){
        Scanner scan = new Scanner(System.in);
        SimuladorGenerico simulador;
        Evento e1 = new Evento("Desayunar",8.0d);
        Evento e2 = new Evento("Almorzar",12.0d);
        Evento e3 = new Evento("Cenar",18.0d);
        System.out.println("Seleccione un tipo de simulador:");
        System.out.println("    1. Simulador con cola");
        System.out.println("    2. Simulador2");
        System.out.println(">> ");

        int opcion = scan.nextInt();
        switch(opcion){
            case 1:
                Queue<Evento> eventos = new LinkedList<>();
                eventos.add(e1);
                eventos.add(e2);
                eventos.add(e3);
                simulador = new SimuladorColaEventos(eventos);
                break;
            default:
                ArrayList<Evento> eventos2 = new ArrayList<>();
                eventos2.add(e2);
                eventos2.add(e1);
                eventos2.add(e3);
                simulador = new Simulador2(eventos2);
                break;
        }

        simulador.simular();
    }
}

```

Evidencia del funcionamiento:

```
Seleccione un tipo de simulador:
  1. Simulador con cola
  2. Simulador2
>>
1
Se cambia del estado Inicio al estado Desayunar
Se cambia del estado Desayunar al estado Almorzar
Se cambia del estado Almorzar al estado Cenar
BUILD SUCCESSFUL (total time: 7 seconds)
,
Seleccione un tipo de simulador:
  1. Simulador con cola
  2. Simulador2
>>
2
Se cambia del estado Inicial al estado Almorzar
Se cambia del estado Almorzar al estado Desayunar
Se cambia del estado Desayunar al estado Cenar
BUILD SUCCESSFUL (total time: 5 seconds)
```

- C) Dentro de sus metas a corto plazo está volverse ingeniero en computación en un plazo **no mayor** a dos años, por lo cual, usted ya tiene planes de comprar un automóvil en el primer año de graduado y debe acondicionar su casa de habitación (o negociar con sus familiares) para hacer la remodelación necesaria y contar con un nuevo garaje en el que pueda guardar su vehículo. Como parte del plan, ha decidido poner en práctica todos sus conocimientos e implementar un dispositivo que le permita tener acceso a este sitio, y aprovechando, desarrollar otras características que le ofrezcan una serie de beneficios, pues es un hecho que va a pasar bastante tiempo admirando y cuidando su próxima adquisición.

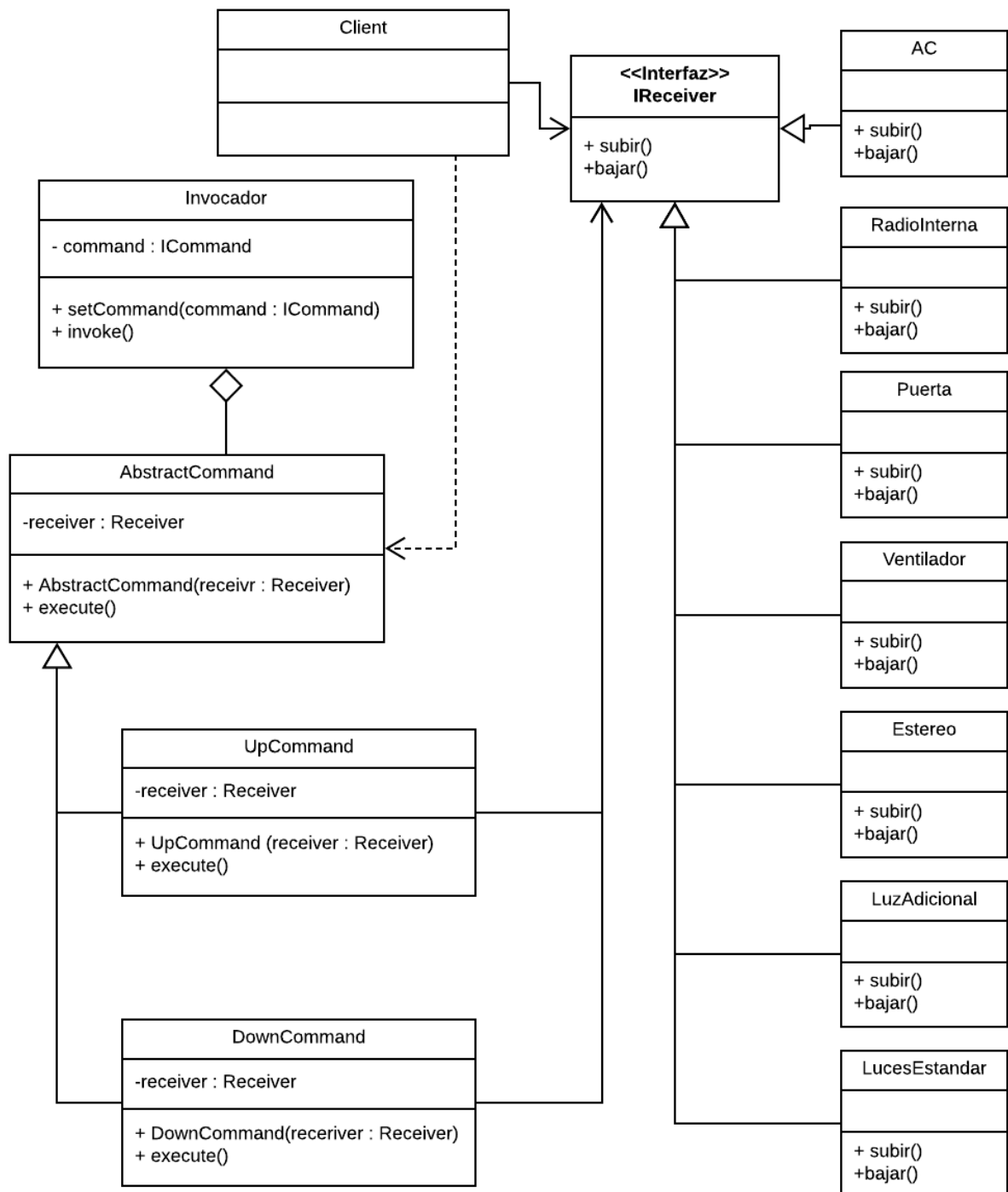
Por lo que ha incluirá en el nuevo garaje un sistema de ventilación (podría ser unos **ventiladores** o talvez un sistema de **AC**, talvez ambos), juegos de **luces adicionales** a la **estándar** por si debe revisar algo del vehículo con detenimiento por ejemplo en la parte inferior del vehículo, un **estéreo** que pueda hacer sonar la música que esté sonando en el **radio interno** del vehículo o bien habilitar otro **dispositivo para escuchar música**, entre otros y por supuesto, el sistema de manejo de la

**puerta** del garaje que le permita ingresar a él o salir sin abandonar su vehículo. Ya ha hecho los contactos con quien se requiera para el diseño físico de este dispositivo que le permitirá vía programación configurar los servicios que se dispondrán y el mecanismo para activar o desactivar cada uno de ellos considerando que en ciertos servicios se debe manipular o graduar las intensidades de funcionamiento de dicho servicio, por ejemplo, **subir** o **bajar** el volumen del estéreo, la intensidad de las luces, la temperatura del AC o la velocidad del ventilador, por ejemplo.

Proponga la utilización de un patrón adecuado que maneje esta situación, justifique su respuesta, construya el modelo en notación UML 2.0 que soporte esta necesidad y programe el funcionamiento de su propuesta de solución en un proyecto usando el lenguaje de programación Java en un paquete llamado **CasoC\_MiGarage\_Patron** (donde Patron es el nombre del patrón propuesto)

Patrón seleccionado (1 puntos)	Justificación (2 puntos)
Command	Es un command porque estandariza la acción de “subir” y “bajar” sin importar si la realiza el sistema de ventilación, las luces, el portón o la música.

Modelo UML 2.0 de implementación propuesta (3 puntos)



Screenshots del código generado y muestras de funcionamiento adecuado del patrón. (5 puntos)

```
public class Invocador {

    private AbstractCommand command;

    public void setCommand(AbstractCommand command) {
        this.command = command;
    }

    public void invoke() {
        command.execute();
    }
}

public abstract class AbstractCommand {

    protected IReceiver receiver;

    public AbstractCommand(IReceiver receiver) {
        this.receiver = receiver;
    }

    public abstract void execute();
}

public class UpCommand extends AbstractCommand {

    public UpCommand(IReceiver receiver) {
        super(receiver);
    }

    @Override
    public void execute() {
        receiver.subir();
    }
}
```

```

public class DownCommand extends AbstractCommand{

    public DownCommand(IReceiver receiver){
        super(receiver);
    }

    @Override
    public void execute() {
        receiver.bajar();
    }

}

public interface IReceiver {
    public void subir();
    public void bajar();
}

public class AC implements IReceiver {

    private int temperatura = 21;

    @Override
    public void subir() {
        if(temperatura < 30)
            temperatura++;
        System.out.println("Se subió la temperatura a: "+temperatura+" grados");
    }

    @Override
    public void bajar() {
        if(temperatura > -5)
            temperatura--;
        System.out.println("Se bajó la temperatura a: "+temperatura+" grados");
    }

}

```

```
public class Estereo implements IReceiver {

    private int volumen = 20;

    @Override
    public void subir() {
        if(volumen < 100)
            volumen++;
        System.out.println("Se subió el volumen a: "+volumen);
    }

    @Override
    public void bajar() {
        if(volumen > 0)
            volumen--;
        System.out.println("Se bajó el volumen a: "+volumen);
    }
}

public class LuzAdicional implements IReceiver {

    private int intensidad = 5;

    @Override
    public void subir() {
        if(intensidad < 10)
            intensidad++;
        System.out.println("Se subió la intensidad de la luz a: "+intensidad);
    }

    @Override
    public void bajar() {
        if(intensidad > 0)
            intensidad--;
        System.out.println("Se bajó la intensidad de la luz a: "+intensidad);
    }
}
```



```
public class Puerta implements IReceiver {

    private boolean cerrada;

    @Override
    public void subir() {
        cerrada = false;
        System.out.println("La puerta del garage está : "+(cerrada ? "cerrada" : "abierta"));
    }

    @Override
    public void bajar() {
        cerrada = true;
        System.out.println("La puerta del garage está : "+(cerrada ? "cerrada" : "abierta"));
    }
}

public class LuzEstandar implements IReceiver {

    private int intencidad = 5;

    @Override
    public void subir() {
        if(intencidad < 10)
            intencidad++;
        System.out.println("Se subió la intencidad de la luz a: "+intencidad);
    }

    @Override
    public void bajar() {
        if(intencidad > 0)
            intencidad--;
        System.out.println("Se bajó la intencidad de la luz a: "+intencidad);
    }
}
```

```
public class RadioInterna implements IReceiver{

    private int volumen = 10;

    @Override
    public void subir() {
        if(volumen < 50){
            volumen++;
        }
        System.out.println("El volumen de la radio subió a nivel: "+volumen);
    }

    @Override
    public void bajar() {
        if(volumen>0){
            volumen--;
        }
        System.out.println("El volumen de la radio bajó a nivel: "+volumen);
    }

}

public class Ventilador implements IReceiver {

    private int velocidad = 3;

    @Override
    public void subir() {
        if(velocidad < 5)
            velocidad++;
        System.out.println("Se subió la velocidad del ventilador a: "+velocidad);
    }

    @Override
    public void bajar() {
        if(velocidad > 0)
            velocidad--;
        System.out.println("Se bajó la velocidad del ventilador a: "+velocidad);
    }

}
```

Muestras de funcionamiento:

```
public static void main (String[] args){
    //Crea los recibidores
    IReceiver ventilador = new Ventilador();
    IReceiver luz = new LuzEstandar();
    IReceiver ac = new AC();
    IReceiver estereo = new Estereo();

    //Crea los comandos con los recibidores respectivos
    AbstractCommand subirVentilador = new UpCommand(ventilador);
    AbstractCommand bajarVentilador = new DownCommand(ventilador);
    AbstractCommand subirLuz = new UpCommand(luz);
    AbstractCommand bajarLuz = new DownCommand(luz);
    AbstractCommand subirAC = new UpCommand(ac);
    AbstractCommand bajarAC = new DownCommand(ac);
    AbstractCommand subirEstereo = new UpCommand(estereo);
    AbstractCommand bajarEstereo = new DownCommand(estereo);

    //Crea el invocador
    Invocador invoker = new Invocador();

    //Asigna el comando al invocador y ejecuta
    invoker.setCommand(subirVentilador);
    invoker.invoke();
    invoker.setCommand(bajarVentilador);
    invoker.invoke();
    invoker.setCommand(subirLuz);
    invoker.invoke();
    invoker.setCommand(bajarLuz);
    invoker.invoke();
    invoker.setCommand(subirAC);
    invoker.invoke();
    invoker.setCommand(bajarAC);
    invoker.invoke();
    invoker.setCommand(subirEstereo);
    invoker.invoke();
    invoker.setCommand(bajarEstereo);
    invoker.invoke();
}
```

Se subió la velocidad del ventilador a: 4

Se bajó la velocidad del ventilador a: 3

Se subió la intensidad de la luz a: 6

Se bajó la intensidad de la luz a: 5

Se subió la temperatura a: 22 grados

Se bajó la temperatura a: 21 grados

Se subió el volumen a: 21

Se bajó el volumen a: 20

BUILD SUCCESSFUL (total time: 0 seconds)