# Repaso de Java Semana II

## MisiónTIC 2022 - Ciclo 2

Jhon López, M.Sc. (c), Daniela Martín, M.Sc. (c), Carlos Sierra, M.Sc., Arles Rodríguez, Ph.D.

Jonatan Gómez Perdomo

jgomezpe@unal.edu.co

Research Group on Artificial Life – Grupo de investigación en vida artificial – (Alife)

Computer and System Department

Engineering School

Universidad Nacional de Colombia

# Contenido




# Agenda




# Enunciado

RedFlix, es un negocio nuevo que pretende ofrecer películas y series en línea. RedFlix desea que usted realice una primera implementación demo de su catálogo de películas y series. Para realizar esto se tiene el concepto de Catálogo donde se tienen Contenidos y un contenido puede ser Película ó Serie. Una Película tiene un título, un resumen y el año. Mientras que una Serie tiene título, número de temporadas y número de episodios.
Por ahora se requiere que se muestre a través de un programa como se pueden agregar, eliminar y visualizar los Contenidos almacenados en el Catálogo. (Pensemos por 3 minutos en definir claramente el problema)


# Entendiendo el Problema

Se pueden identificar todos los sustantivos pues estos muestran posibles entidades que pueden ser candidatos a clases. Se pueden idenficar porque tienen atributos o relaciones con otras clases (es un, tiene un, implementa) y realizan ciertas acciones.


# Determinación de posibles Clases y sus relaciones

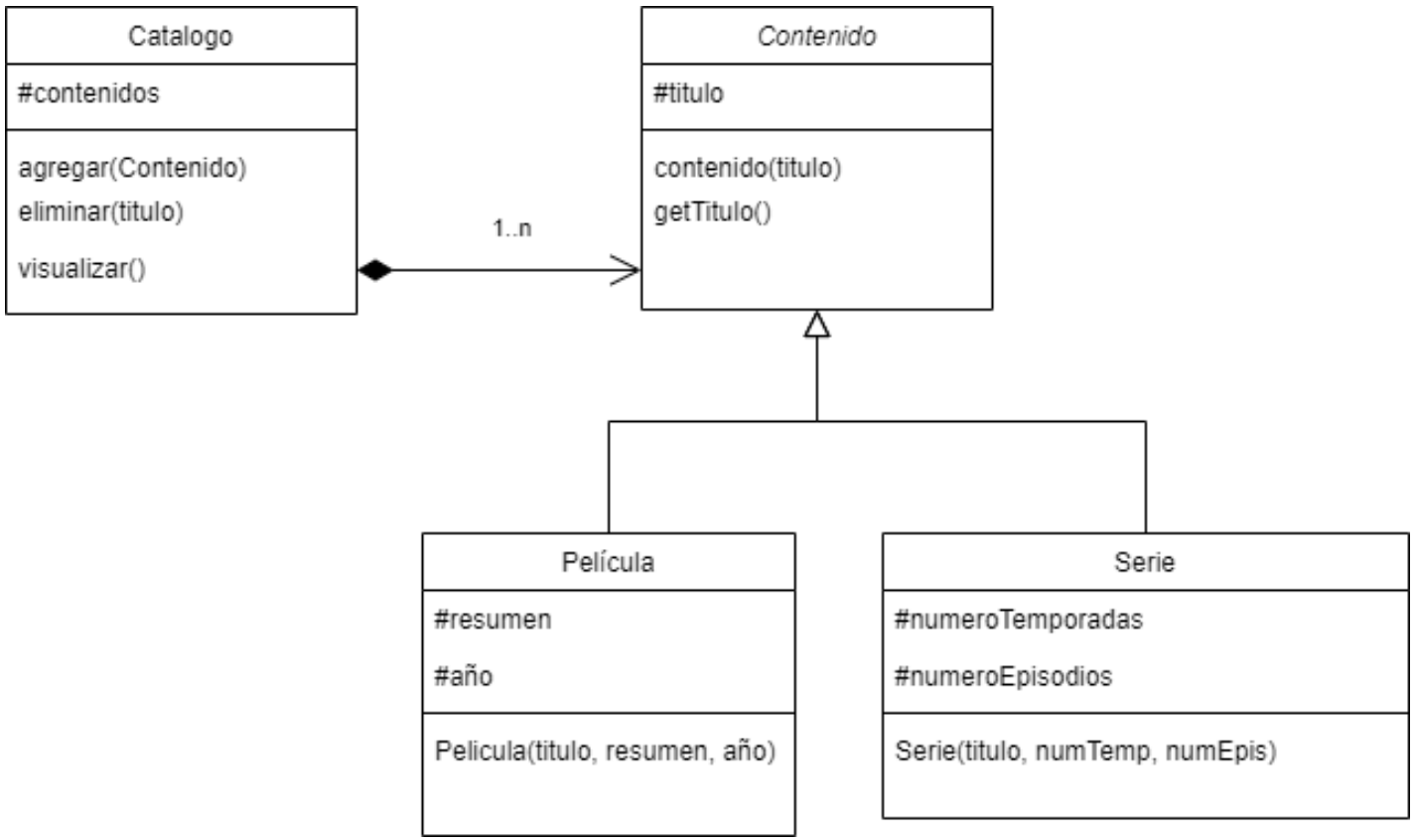
- ~~RedFlix~~
- ~~negocio~~
- **Película tiene título resumen y año**
- **Serie tiene título, número de temporadas y número de episodios**
- ~~concepto~~
- **Catálogo tiene contenidos**
- **Contenido puede ser una película o serie**


# Diseño de la solución

Por ahora se requiere que se muestre a través de un **programa** como se pueden agregar, eliminar y visualizar los **Contenidos** almacenados en el **Catálogo**. (Pensemos por 3 minutos en definir cuales serían candidatos a clases).
Se puede tener en cuenta lo anterior para determinar un diagrama de clases con la estructura de la solución.


# Diagrama de Clases

Un posible diagrama de clases es el siguiente:



# Definición de la funcionalidad

Una vez se tiene el diagrama se puede ver que el Catálogo puede tener un arreglo ó ArrayList de contenidos en Java. La funcionalidad corresponderá a agregar, eliminar y visualizar títulos del catálogo.


# Codificación Clase Contenido

```
public class Contenido {
    protected String titulo;

    public Contenido(String titulo) {
        this.titulo = titulo;
    }

    public String getTitulo() {
        return titulo;
    }
}
```


# Codificación Clase Película

```
public class Pelicula extends Contenido {
    protected String resumen;
    protected String anio;

    public Pelicula(String titulo, String resumen, String anio) {
        super(titulo);
        this.resumen = resumen;
        this.anio = anio;
    }

    @Override
    public String toString() {
        return "Pelicula: {" + super.getTitulo() + ", resumen="
                + resumen + ", año=" + anio + "}";
    }
}
```


# Codificación clase Serie

```
public class Serie extends Contenido {
    private String temporadas;
    private String episodios;

    public Serie(String temporadas, String episodios, String titulo) {
        super(titulo);
        this.temporadas = temporadas;
        this.episodios = episodios;
    }

    @Override
    public String toString() {
       return "Serie: {" + super.getTitulo()
                + ", temporadas=" + temporadas +
                ", episodios=" + episodios + "}";
    }
}
```


# Codificación clase Catálogo

```
import java.util.ArrayList;

public class Catalogo {
    protected ArrayList<Contenido> contenidos = new ArrayList<>();

    public void addContenido(Contenido c) {
        contenidos.add(c);
    }

    public void eliminar(String nombre) {
        for (int i=0; i < contenidos.size(); i++) {
            if(contenidos.get(i).getTitulo().equalsIgnoreCase(nombre))
                contenidos.remove(i);
            }
        }
    }
```


# Codificación clase Catálogo

```
public void visualizar() {
    for (Contenido c : contenidos) {
        System.out.println(c);
    }
}

 public static void main(String[] args) {
    Catalogo c = new Catalogo();

    Pelicula peli = new Pelicula("Batman 2020", "aparece de nuevo", "1996");
    Pelicula peli2 = new Pelicula("Inception", "Is about dreams", "2010");
    Serie serie1 = new Serie("3", "186", "How I meet your mother");
    Pelicula peli3 = new Pelicula("Superman", "Hero", "1996");

    c.addContenido(peli);
    c.addContenido(peli2);
    c.addContenido(serie1);
    c.addContenido(peli3);

    System.out.println("-----------");
    c.visualizar();

    System.out.println("Eliminar a Batman");
    c.eliminar("Batman 2020");
    c.visualizar();
}
}
```


# Agenda




# Enunciado

El año 2020 ha sido un año difícil para los niños, muchos debieron pasar el 31 de octubre en sus casas, en misionTIC no queremos dejar pasar su día de celebración desapercibido por eso vamos hacer una simulación con la mejor fiesta de disfraces de la historia, queremos que los estudiantes nos ayuden a hacerla cada vez mejor. Será una fiesta con niños como invitados, tendremos menus variados de comida y cada niño se podrá disfrazar de su personaje favorito, se desea contar con la mayor variedad de disfraces para el concurso que se hace a media noche donde se premia cada categoría (Monstruo, personaje de comic, personaje de anime y otros)

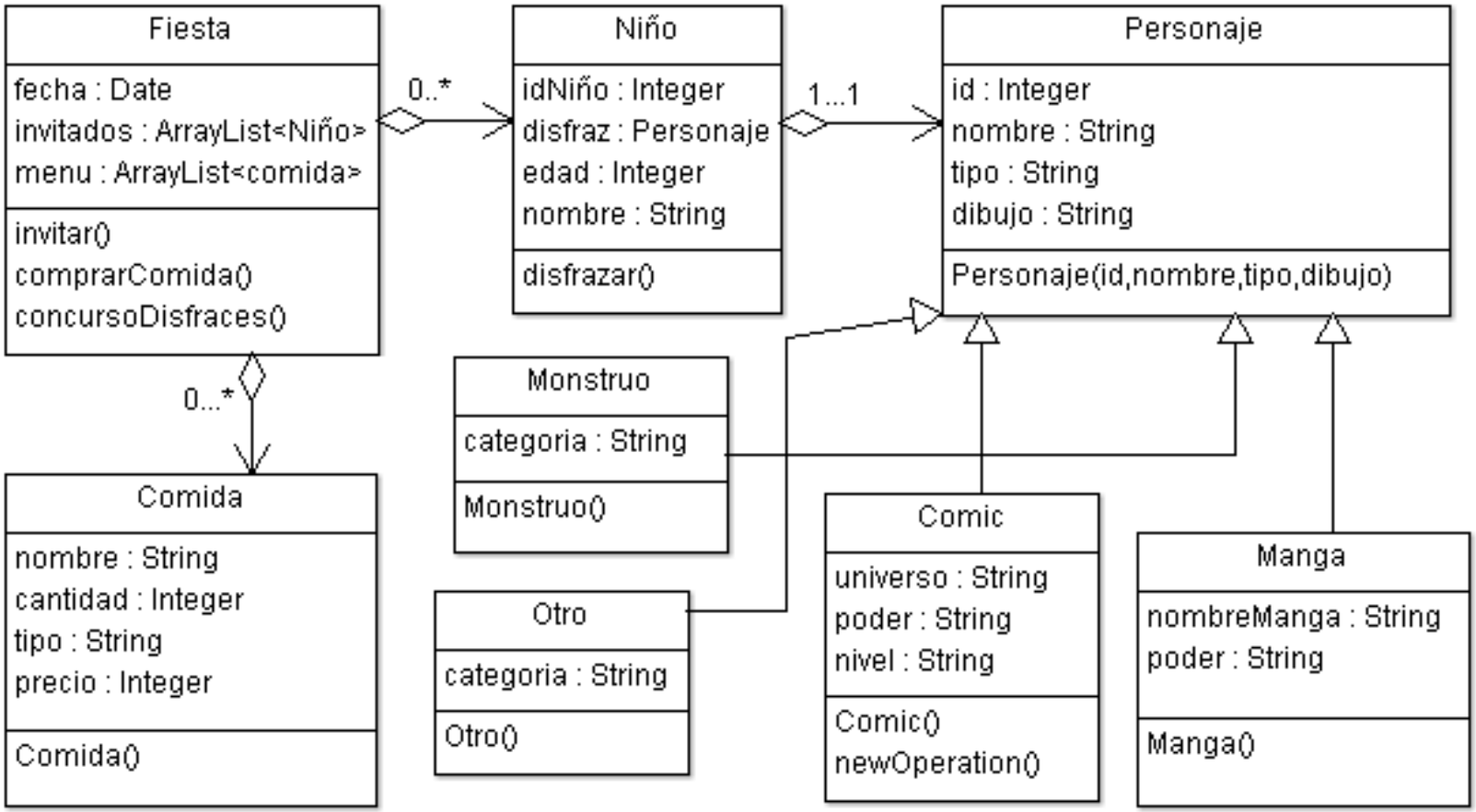

# Entendiendo el Problema

El enunciado brinda gran cantidad de información que sirve para modelar el problema, los sustantivos son indicativos de las clases que podemos usar pero a veces lo dificil es saber como relacionar las clases. Se sugiere organizar los sustantivos por categorias y revisar las posibles formas de conectarlas, al final solo una de esas posibilidades tendrá mas sentido.
Los verbos nos brindan información de las posibles funcionalidades que puede tener el programa, será dificil en qué clase se puede realizar la acción pero será donde tenga mayor sentido y coherencia.
**Clave:** Cada clase suele especializarse en unas tareas específicas por lo que entre mas precisa sea su funcionalidad es mejor


# Especificando el Problema

- ~~Disfraz~~
- **Fiesta** tiene invitados(colección de niños) y menus(colección de comida)
- **Personaje** tiene nombre tipo
- **Las categorias de disfraz son clases hijas de personaje**
- ~~Menu~~
- **Niño**
- **Comida**


# Diagrama de Clases



# Codificación

```
public class Personaje {
    private int idPersonaje;
    private String nombre;
    private String tipo;
    private String dibujo;

    public Personaje() {
        idPersonaje=0;
        nombre="Sin nombre";
        tipo="Monstruo";
        String dibujo="";
    }

    public Personaje(int idPersonaje, String nombre, String tipo, String di
        this.idPersonaje = idPersonaje;
        this.nombre = nombre;
        this.tipo = tipo;
        this.dibujo = dibujo;
    }
```


# Codificación

```
public class Niño {

    private int idNiño;
    private String nombre;
    private int edad;
    private Personaje disfraz;

    public Niño() {
        idNiño=0;
        nombre="NN";
        edad=0;
        disfraz= new Personaje();
    }

    public Niño(int idNiño, String nombre, int edad, Personaje disfraz)
        this.idNiño = idNiño;
        this.nombre = nombre;
        this.edad = edad;
        this.disfraz = disfraz;
    }
}
```


# Codificación

```
public class Comida {
    private int idComida;
    private String nombre;
    private int cantidad;
    private int precio;

    public Comida() {
        idComida=0;
        nombre="Dulces";
        cantidad=100;
        precio=100;
    }

    public Comida(int idComida, String nombre, int cantidad, int precio) {
        this.idComida = idComida;
        this.nombre = nombre;
        this.cantidad = cantidad;
        this.precio = precio;
```


# Codificación

```
public class Fiesta {

    private Calendar fecha;
    private ArrayList<Niño> invitados;
    private ArrayList<Comida> menu;

    public Fiesta() {
        fecha = Calendar.getInstance();
        invitados = new ArrayList();
        menu = new ArrayList();
    }

    public void invitar(Niño n){
        invitados.add(n);
    }

    public void comprarComida(Comida c){
        menu.add(c);
    }
```


# Agenda




# Enunciado

Las clases de los ciclos de MisiónTIC 2022 han sido un poco extenuantes, se nota que los profesores se tomaron a pecho la palabra *intensivo*. Aunque los estudiantes han dado uso a distintas redes sociales como alternativa a la comunicación fuera de clase, usted considera que los mensajes de chats son demasiados, y le cuesta mantenerse actualizado. Así, con lo aprendido en el curso, usted considera que puede crear una red social sencilla, la cual pueda tener usuarios, grupos (compuestos por usuarios), y en donde solo se puedan colocar estados, comentarios en los estados, y un ajuste para recibir notificaciones de nuevos estados. (Pensemos por 3 minutos en definir claramente el problema)


# Entendiendo el Problema

El enunciado brinda gran cantidad de información que sirve para modelar el problema. Los sustantivos son indicativos de las clases se deben construir, incluyendo los atributos y comportamientos que se necesitan, a veces lo difícil es saber cómo relacionar las clases. Se sugiere organizar los sustantivos por categorías y revisar las posibles formas de conectarlas (recuerde usar oraciones de conexión), al final solo una de esas posibilidades tendrá mas sentido.
Una vez logre comprender patrones, también podrá evaluar la opción de usar patrones de diseño para mejorar la calidad de su solución.

**Clave:** Pensar en cómo deberían verse al final las cosas es vital para identificar qué patrones de diseño pueden ser los más apropiados para resolver el problema.


# Especificando el Problema

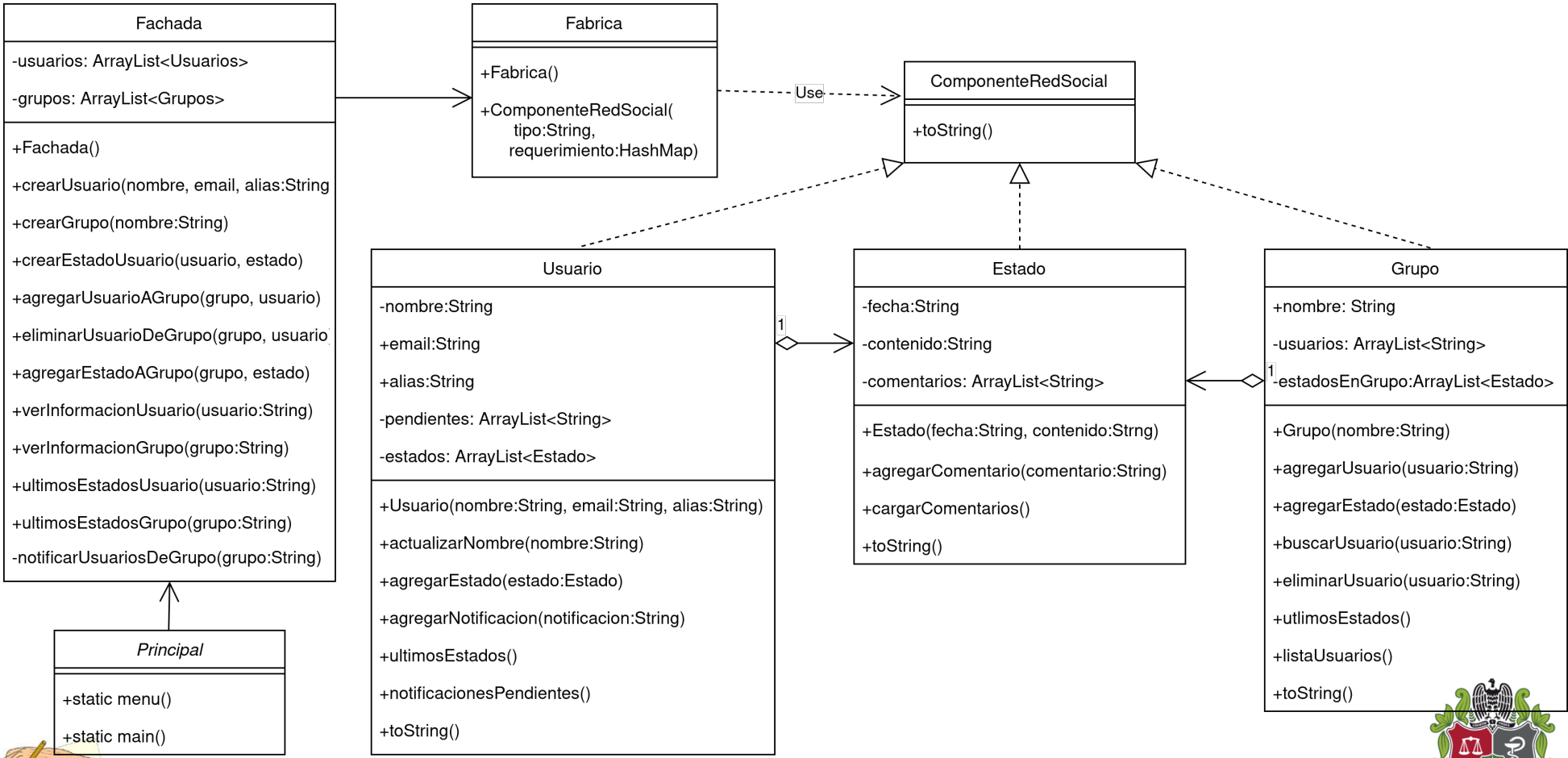
En este punto es bueno pensar un poco en las posibles *User Stories* de este miniproyecto:

- Yo como usuario, quiero crear un estado, para compartir información en la red social.
- Yo como usuario, quiero crear un grupo, para compartir y discutir de contenido sobre un tema en particular.
- Yo como usuario, quiero afiliarme o desafiliarme de un grupo, para controlar el contacto con grupos de trabajo.
- Yo como usuario, quiero ver mis notificaciones sin leer, para estar al tanto de lo que está pasando en la red.


# Determinación de Posibles Clases y sus Relaciones

- ~~Usuario~~
- ~~Grupo~~
- Un **usuario** puede pertener a uno o varios grupos. A su vez, un **grupo** puede estar compuesto por uno o varios usuarios.
- ~~Estado~~
- Un **usuario** puede escribir uno o varios estados. A su vez, un **estado** puede pertenecer a un usuario.
- Un **grupo** puede tener uno o varios estados. A su vez, un **estado** puede pertenecer a un grupo.


# Diagrama de Clases



# Codificación

A continuación se explicarán las clases utilizadas en el proyecto. De igual manera, en este repositorio de GitHub podrá encontrar el código fuente.

Interface para implementar el Patrón de Diseño Factory:

```
public interface ComponenteRedSocial {
    public String toString();
}
```


# Codificación

Clase **Usuario**:

```
import java.util.ArrayList;

public class Usuario implements ComponenteRedSocial{

    private String nombre = "";
    public String email = ""; // este debe ser único e irrepetible
    public String alias = ""; // este debe ser único e irrepetible
    private ArrayList<String> pendientes = null;
    private ArrayList<Estado> estados = null;

    public Usuario(String nombre, String email, String alias){
        this.nombre = nombre;
        this.email = email;
        this.alias = alias;
        this.pendientes = new ArrayList<>();
        this.estados = new ArrayList<>();
    }
```


# Codificación

Clase **Usuario**:

```
public void actualizarNombre(String nombre){
    this.nombre = nombre;
}

@Override
public String toString(){
    String usuario = "El usuario tiene por nombre " + this.nombre +
            ",con el email " + this.email + " y con el alias " +
            this.alias + ". Ha escrito " + this.estados.size() +
            " estados, y tiene " + this.pendientes.size() +
            " notificaciones por leer.";
    return usuario;
}
```


# Codificación

Clase **Usuario**:

```
public String agregarEstado(Estado nuevoEstado){
    try{
        this.estados.add(nuevoEstado);
        return "Proceso Exitoso";
    }catch(Exception ex){
        ex.printStackTrace();
        return "Proceso No Exitoso.";
    }
}

public Estado[] ultimosEstados(){
    int cantidad = (this.estados.size() < 5) ? this.estados.size() : 5;
    Estado[] estados_ = new Estado[cantidad];
    for(int i = 0; i < cantidad; i++){
        estados_[i] = this.estados.get(this.estados.size() - 1 - i);
    }
    return estados_;
}
```


# Codificación

Clase **Usuario**:

```
public String agregarNotificacion(String nuevaNotificacion){
    try{
        this.pendientes.add(nuevaNotificacion);
        return "Proceso Exitoso";
    }
    catch(Exception ex){
        ex.printStackTrace();
        return "Proceso No Exitoso.";
    }
}

public String verNotificacionesPendientes(){
    String respuesta = "";
    for(String notificacion: this.pendientes){
        respuesta += notificacion + "\n";
    }
    this.pendientes.clear();
    return respuesta;
}
```


# Codificación

Clase **Grupo**:

```
import java.util.ArrayList;

public class Grupo implements ComponenteRedSocial{

    public String nombre = ""; //este debe ser único e irrepetible
    private ArrayList<String> usuarios = null;
    private ArrayList<Estado> estadosEnGrupo = null;

    public Grupo(String nombre){
        this.nombre = nombre;
        this.usuarios = new ArrayList<>();
        this.estadosEnGrupo = new ArrayList<>();
    }
```


# Codificación

## Clase **Grupo**:

```
public String agregarUsuario(String nuevoUsuario){
    try{
        this.usuarios.add(nuevoUsuario);
        return "Proceso Exitoso";
    } catch(Exception ex){
        ex.printStackTrace();
        return "Proceso No Exitoso";
    }
}

public boolean buscarUsuario(String usuario){
    try{
        return this.usuarios.contains(usuario);
    }catch(Exception ex){
        ex.printStackTrace();
        return false;
    }
}

public String eliminarUsuario(String usuario){
    try{
        this.usuarios.remove(usuario);
        return "Proceso Exitoso";
    }catch(Exception ex){
        ex.printStackTrace();
        return "Proceso No Exitoso";
    }
```


# Codificación

Clase **Grupo**:

```
public String agregarEstado(Estado nuevoEstado){
    try{
        this.estadosEnGrupo.add(nuevoEstado);
        return "Proceso Exitoso";
    }catch(Exception ex){
        ex.printStackTrace();
        return "Proceso No Exitoso.";
    }
}


public Estado[] ultimosEstados(){
    int cantidad = (this.estadosEnGrupo.size() < 10) ? this.estadosEnGr
    Estado[] estados = new Estado[cantidad];
    for(int i = 0; i < cantidad; i++){
        estados[i] = this.estadosEnGrupo.get(cantidad - 1 - i);
    }
    return estados;
}
```


# Codificación

Clase **Estado**:

```
import java.util.ArrayList;

public class Estado implements ComponenteRedSocial{

    public String fecha = "";
    public String contenido = "";
    private ArrayList<String> comentarios = null;

    public Estado(String fecha, String contenido){
        this.fecha = fecha;
        this.contenido = contenido;
        this.comentarios = new ArrayList<>();
    }
```


# Codificación

Clase **Estado**:

```
public String agregarComentario(String nuevoComentario){
    try{
        this.comentarios.add(nuevoComentario);
        return "Proceso Exitoso";
     }catch(Exception ex){
        return "Proceso No Exitoso";
     }
}

public String cargarComentarios(){
     String comentariosTotal = "";
     for(String comentario: this.comentarios){
        comentariosTotal += comentario + "";
     }
     return comentariosTotal;
}
```


# Codificación

## Clase **Fabrica**:

```
import java.util.HashMap;

public class Fabrica {
    public Fabrica(){}

    public ComponenteRedSocial crearComponenteRedSocial(String tipo, HashMap<String, String> requerimiento){
        ComponenteRedSocial respuesta = null;
        switch(tipo){
            case "Usuario":
                String nombre = requerimiento.get("nombre");
                String email = requerimiento.get("email");
                String alias = requerimiento.get("alias");
                respuesta = new Usuario(nombre, email, alias);
                break;
            case "Grupo":
                String nombreGrupo = requerimiento.get("nombre");
                respuesta = new Grupo(nombreGrupo);
                break;
            case "Estado":
                String fecha = requerimiento.get("fecha");
                String contenido = requerimiento.get("contenido");
                respuesta = new Estado(fecha, contenido);
                break;
        }
        return respuesta;
    }
}
```