PATRONES DE DISEÑO

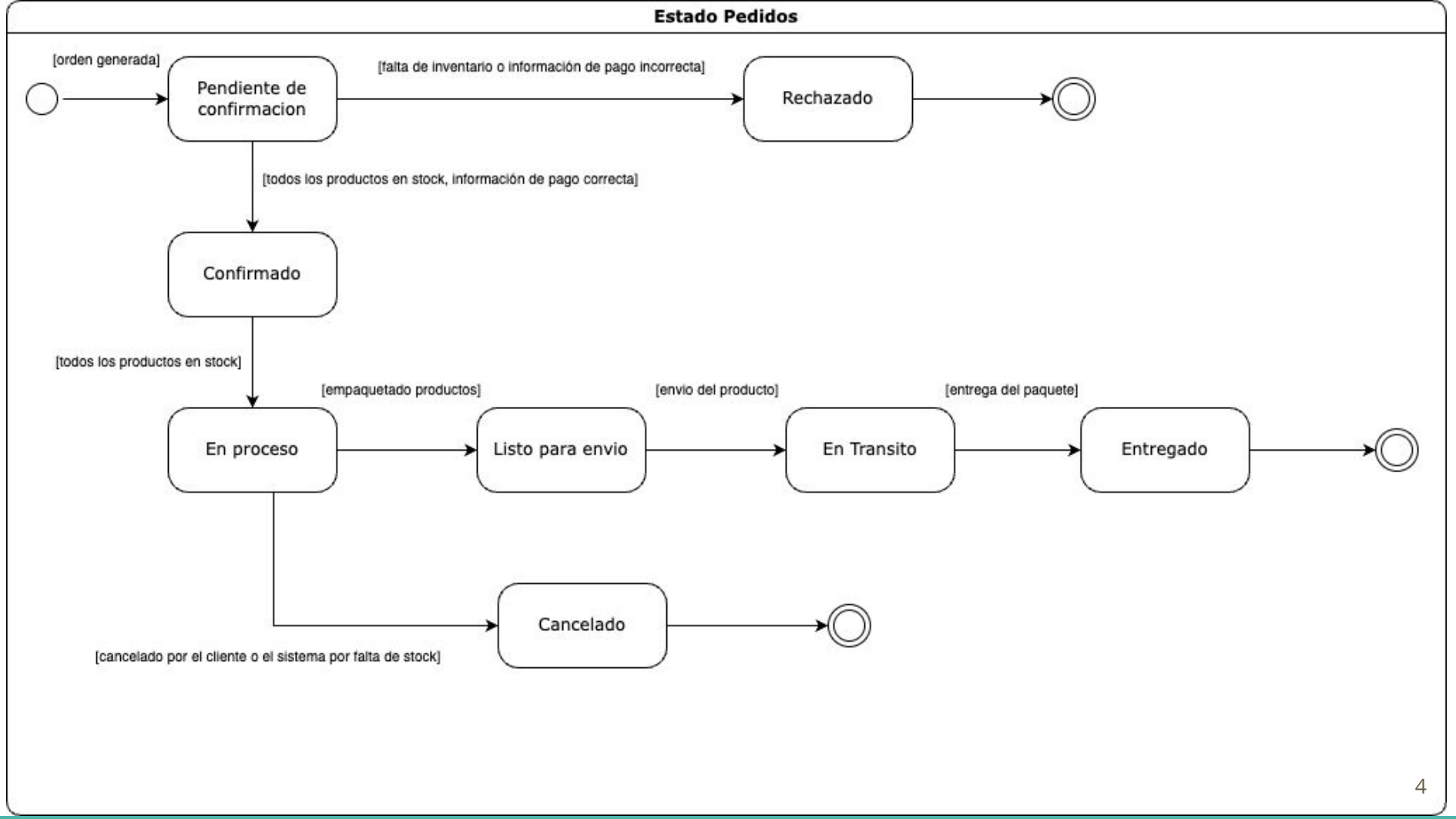
Seguimiento de estado de pedidos

Alumna : Valdiviezo Anahí

Problema

Necesidad de un sistema de seguimiento de estado de pedidos, este no solo es una conveniencia para los clientes, sino que también es esencial para la eficiencia operativa y la competitividad de una plataforma de comercio electrónico.

Diagrama de estados


Estado Pedidos
[orden generada]
Pendiente de confirmacion
[falta de inventario o información de pago incorrecta]
Rechazado
[todos los productos en stock, información de pago correcta]
Confirmado
[todos los productos en stock]
[empaquetado productos]
[envio del producto]
[entrega del paquete]
En proceso
Listo para envio
En Transito
Entregado
Cancelado
[cancelado por el cliente o el sistema por falta de stock]

Importancia de los patrones de diseño

- **Contribuyen a la robustez al evitar errores comunes**
- **Facilitan la mantenibilidad al organizar el código de manera clara y promueven la flexibilidad al permitir adaptarse fácilmente a cambios**
- **Mejoran la calidad del software y facilitan su evolución a lo largo del tiempo.**

Patrones implementados

1. Singleton
2. State
3. Observer

UML inicial

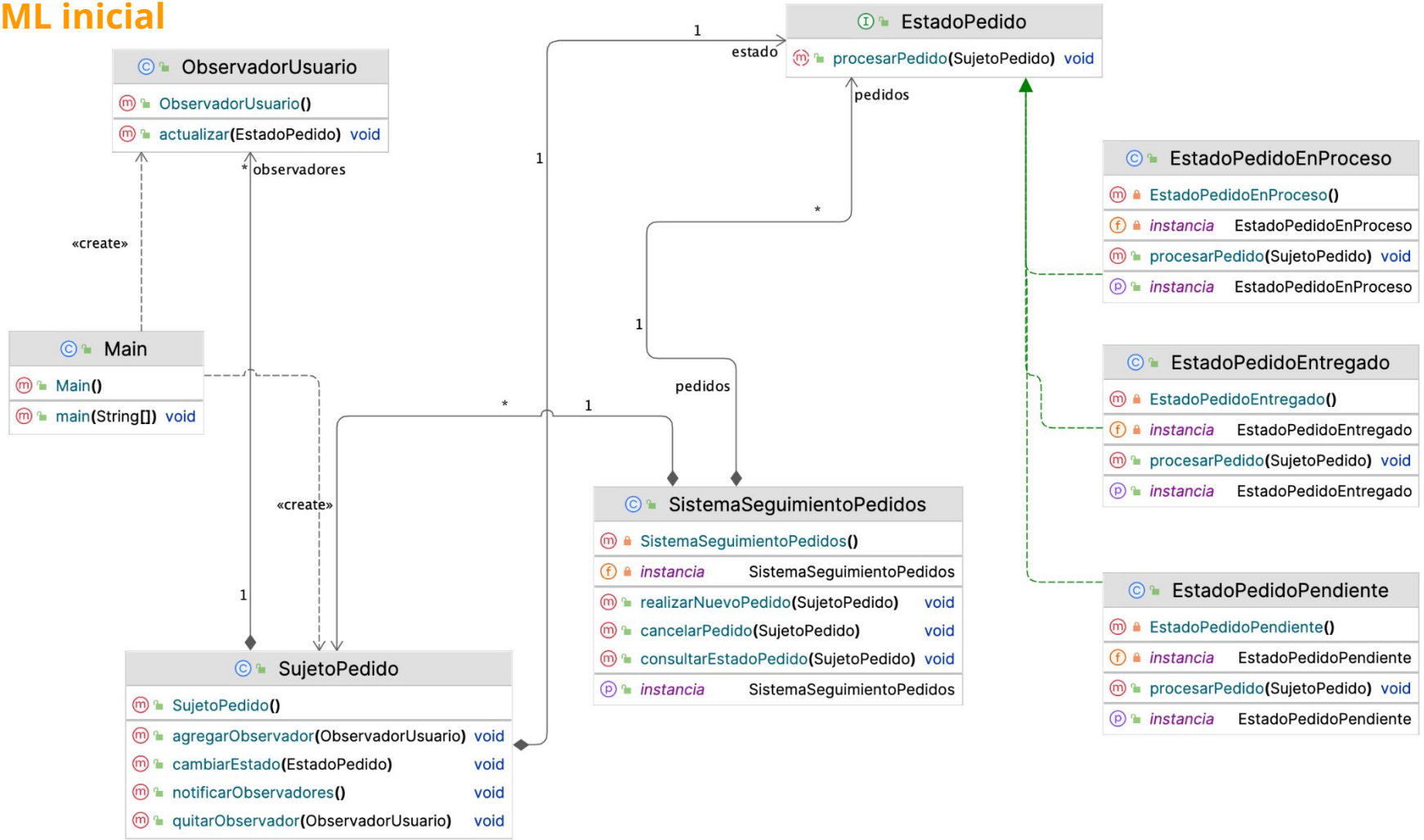

EstadoPedido
procesarPedido(SujetoPedido) void
ObservadorUsuario
ObservadorUsuario()
actualizar(EstadoPedido) void
EstadoPedidoEnProceso
EstadoPedidoEnProceso()
instancia EstadoPedidoEnProceso
procesarPedido(SujetoPedido) void
instancia EstadoPedidoEnProceso
EstadoPedidoEntregado
EstadoPedidoEntregado()
instancia EstadoPedidoEntregado
procesarPedido(SujetoPedido) void
instancia EstadoPedidoEntregado
EstadoPedidoPendiente
EstadoPedidoPendiente()
instancia EstadoPedidoPendiente
procesarPedido(SujetoPedido) void
instancia EstadoPedidoPendiente
Main
Main()
main(String[]) void
SistemaSeguimientoPedidos
SistemaSeguimientoPedidos()
instancia SistemaSeguimientoPedidos
realizarNuevoPedido(SujetoPedido) void
cancelarPedido(SujetoPedido) void
consultarEstadoPedido(SujetoPedido) void
instancia SistemaSeguimientoPedidos
SujetoPedido
SujetoPedido()
agregarObservador(ObservadorUsuario) void
cambiarEstado(EstadoPedido) void
notificarObservadores() void
quitarObservador(ObservadorUsuario) void
1
estado
pedidos
*
observadores
«create»
1
*
1
pedidos
*
1
«create»
1

Diagrama UML

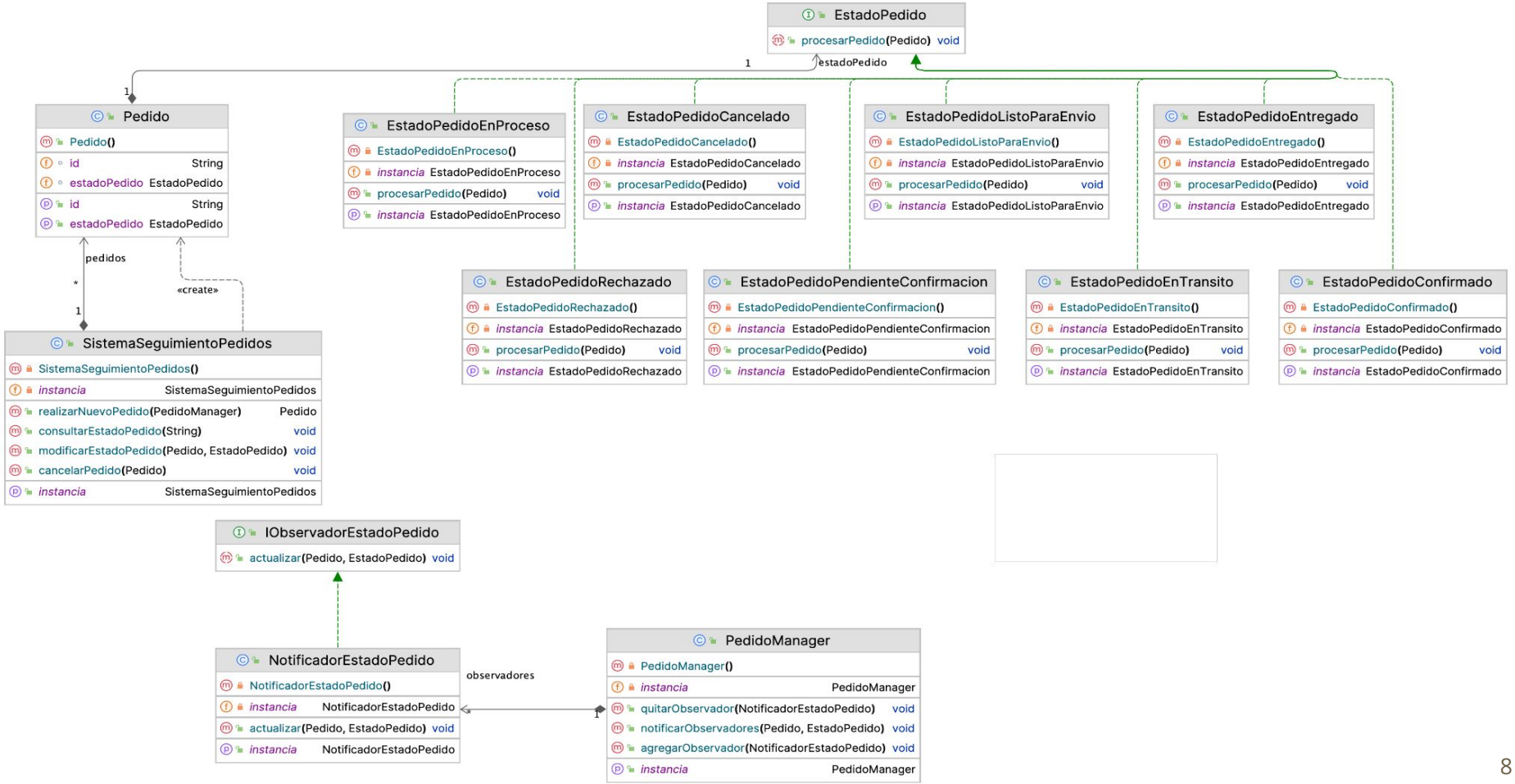
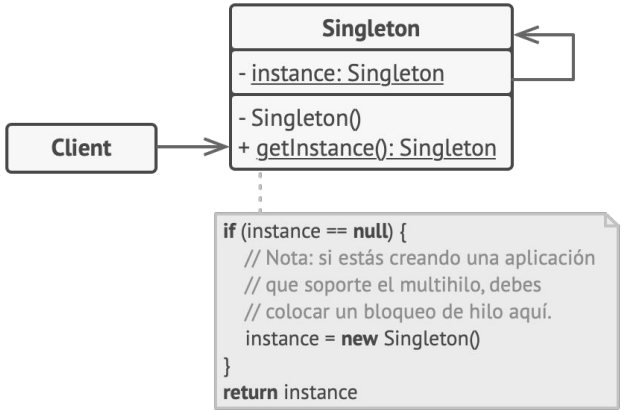


Diagrama UML

Singleton

Es un patrón de diseño creacional que nos permite asegurarnos de que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia.

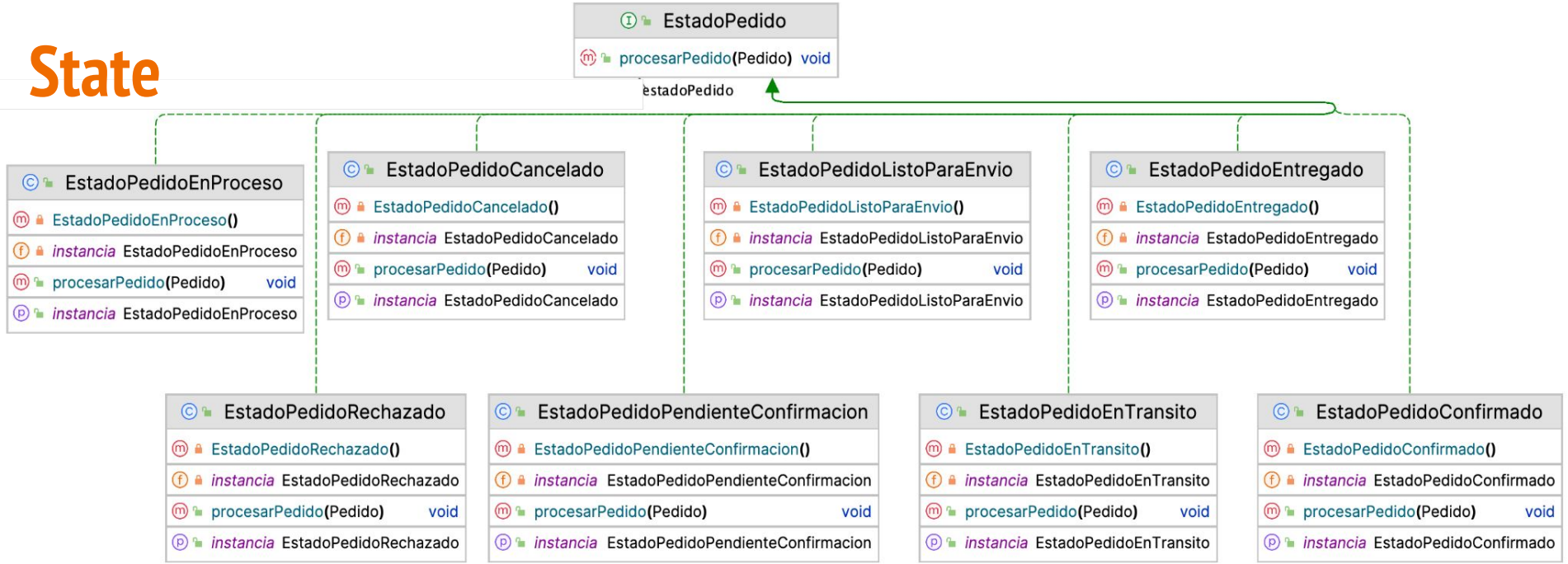



```
public class SistemaSeguimientoPedidos {
    private static SistemaSeguimientoPedidos instancia;

    private SistemaSeguimientoPedidos() {
    }

    public static SistemaSeguimientoPedidos getInstancia() {
        if (instancia == null) {
            instancia = new SistemaSeguimientoPedidos();
        }
        return instancia;
    }
```

State



Pertenece a la categoría de patrones de comportamiento.

Este patrón se utiliza cuando un objeto debe cambiar su comportamiento cuando su estado interno cambia.

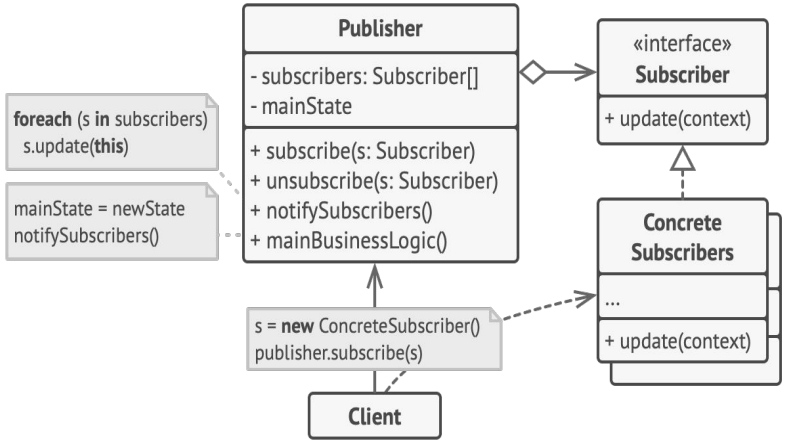
En lugar de tener múltiples condicionales que verifican el estado actual.

El patrón State encapsula los diferentes estados en clases separadas y delega el comportamiento a estos objetos de estado.

Observer

Pertenece a la categoría de patrones de comportamiento.

Este patrón define una relación de uno a muchos entre objetos, de modo que cuando un objeto cambia de estado, todos sus dependientes, llamados observadores, son notificados y actualizados automáticamente.



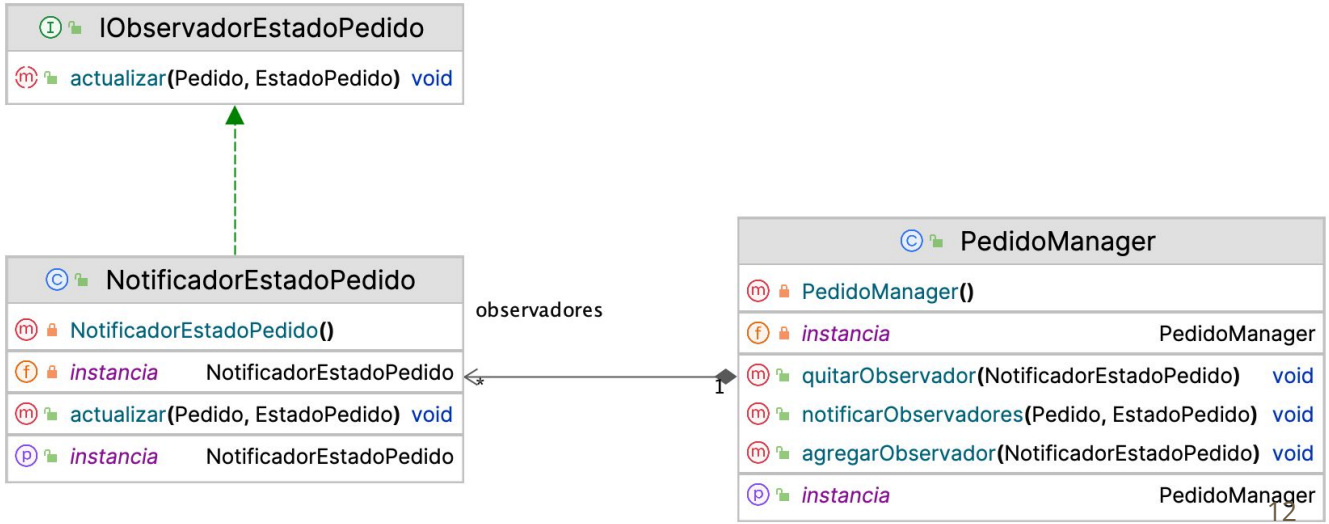
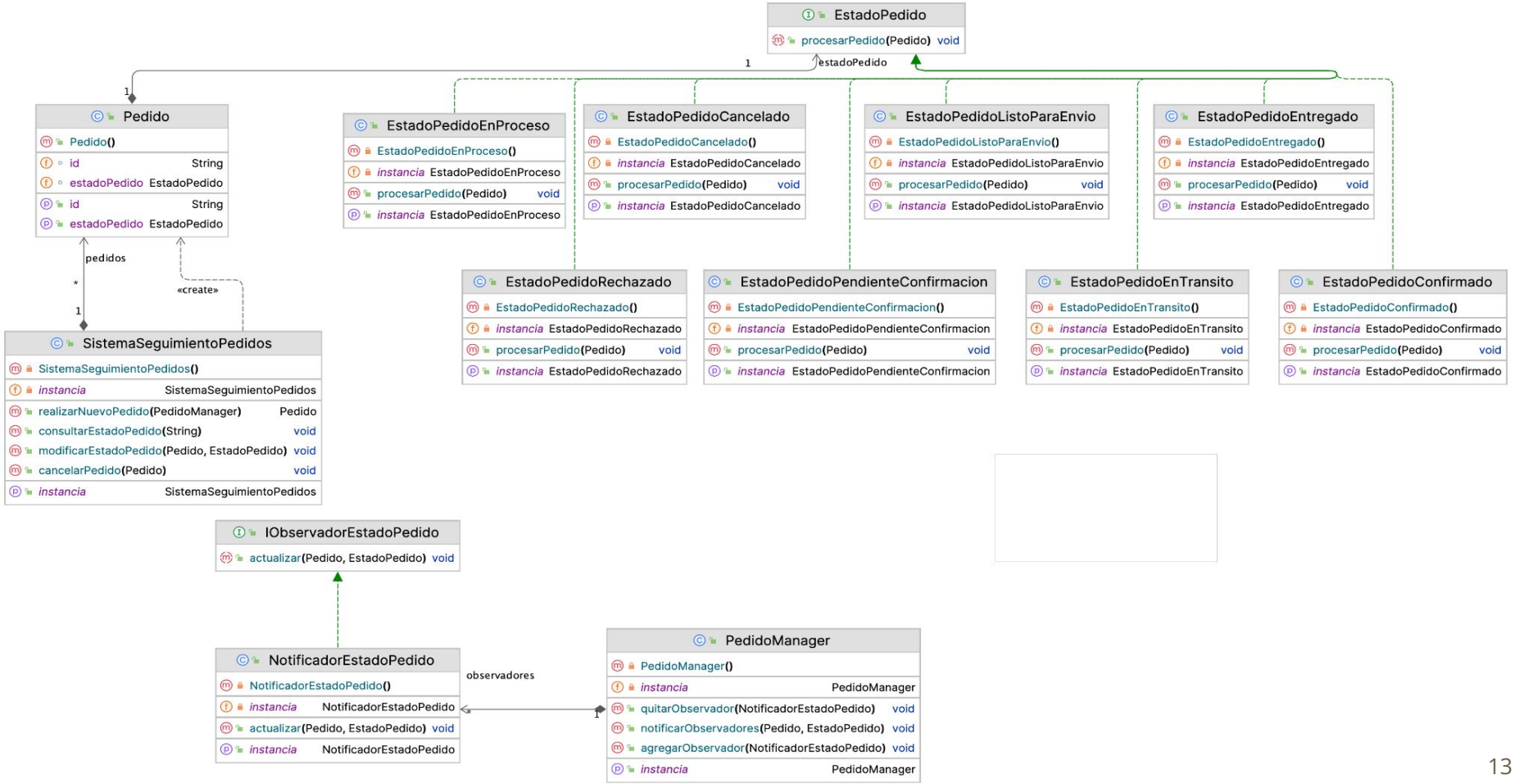


Diagrama UML



Main

```
public class Main {

    public static void main(String[] args) {

        SistemaSeguimientoPedidos sistema = SistemaSeguimientoPedidos.getInstancia();

        PedidoManager pedidoManager = PedidoManager.getInstancia();

        NotificadorEstadoPedido observador = NotificadorEstadoPedido.getInstancia();

        pedidoManager.agregarObservador(observador);


        Pedido pedido = sistema.realizarNuevoPedido(pedidoManager);

        sistema.consultarEstadoPedido(pedido.getId());

        sistema.modificarEstadoPedido(pedido, EstadoPedidoEnProceso.getInstancia());

        sistema.cancelarPedido(pedido);

    }

}
```

Impresion en Consola

```
----- Nuevo pedido realizado -----

El pedido 125 está pendiente de confirmación

----- Estado actual del pedido: EstadoPedidoPendienteConfirmacion

El pedido125ha sido confirmado y está pendiente de procesamiento

El pedido 125 está en proceso de preparación

----- Pedido cancelado

El pedido 125 ha sido cancelado

---- Pedido no encontrado
```

Pedido

```java
public class Pedido {

    String id;

    EstadoPedido estadoPedido;

    public Pedido() {

        this.id = String.valueOf((int) (Math.random() * 1000));

    }

    public String getId() {

        return id;

    }

    public EstadoPedido getEstadoPedido() {

        return estadoPedido;

    }

    public void setEstadoPedido(EstadoPedido pedido) {

        this.estadoPedido = pedido;

    }

}
```

EstadoPedidoEntregado

```
public class EstadoPedidoEntregado implements EstadoPedido {
    private static EstadoPedidoEntregado instancia;

    private EstadoPedidoEntregado() {
    }
    public static EstadoPedidoEntregado getInstancia() {
        if (instancia == null) {
            instancia = new EstadoPedidoEntregado();
        }
        return instancia;
    }
    @Override
    public void procesarPedido(Pedido pedido) {
        System.out.println("El pedido " + pedido.getId() + " ha sido entregado con éxito");
    }
}
```

PedidoManager

```
public class PedidoManager {
    private static PedidoManager instancia;
    private List<NotificadorEstadoPedido> observadores = new ArrayList<>();
    private PedidoManager() {}
    public static PedidoManager getInstancia() {
        if (instancia == null) {
            instancia = new PedidoManager();}
        return instancia;}
    public void agregarObservador(NotificadorEstadoPedido observador) {
        observadores.add(observador);}
    public void quitarObservador(NotificadorEstadoPedido observador) {
        observadores.remove(observador);}
    public void notificarObservadores(Pedido pedido, EstadoPedido nuevoEstado) {
        for (NotificadorEstadoPedido observador : observadores) {
            observador.actualizar(pedido, nuevoEstado);}}
}
```

NotificadorEstadoPedido

```
public class NotificadorEstadoPedido implements IObservadorEstadoPedido {


    private static NotificadorEstadoPedido instancia;
    private NotificadorEstadoPedido() {
    }
    public static NotificadorEstadoPedido getInstancia() {
        if (instancia == null) {
            instancia = new NotificadorEstadoPedido();
        }
        return instancia;
    }
    @Override
    public void actualizar(Pedido pedido, EstadoPedido nuevoEstado) {
        //Actualiza el estado del pedido en el sistema
        nuevoEstado.procesarPedido(pedido);
    }}
```

SistemaSeguimientoPedidos

```
public class SistemaSeguimientoPedidos {

   private static SistemaSeguimientoPedidos instancia;

   private Map<String, Pedido> pedidos = new HashMap<>();

   private SistemaSeguimientoPedidos() {}

   public static SistemaSeguimientoPedidos getInstancia() {

       if (instancia == null) {

           instancia = new SistemaSeguimientoPedidos();}

       return instancia;}

   public Pedido realizarNuevoPedido(PedidoManager pedidoManager) {

       Pedido pedido = new Pedido();

       System.out.println("----- Nuevo pedido realizado -----");

       EstadoPedido estadoInicial = EstadoPedidoPendienteConfirmacion.getInstancia();

       pedido.setEstadoPedido(estadoInicial);

       pedidoManager.notificarObservadores(pedido, estadoInicial);

       pedidos.put(pedido.getId(), pedido);

       return pedido;}
```

```
public void consultarEstadoPedido(String pedidoID) {

        // Lógica para consultar el estado del pedido

        if (pedidos.containsKey(pedidoID)) {

            Pedido pedido = pedidos.get(pedidoID);

            System.out.println("----- Estado actual del pedido: " +
pedido.getEstadoPedido().getClass().getSimpleName());

        } else { System.out.println("---- Pedido no encontrado");}

    }

    public void cancelarPedido(Pedido pedido) {

        // Lógica para cancelar un pedido

        if (pedidos.containsKey(pedido.getId())) {

            EstadoPedido estadoCancelado = EstadoPedidoCancelado.getInstancia();

            PedidoManager.getInstancia().notificarObservadores(pedido, estadoCancelado);

            pedidos.remove(pedido.getId());

            System.out.println("----- Pedido cancelado");

        } else { System.out.println("----- Pedido no encontrado");}}
```

SistemaSeguimientoPedidos

```
public void modificarEstadoPedido(Pedido pedido, EstadoPedido nuevoEstado) {
        if (pedidos.containsKey(pedido.getId())) {
            pedido.setEstadoPedido(nuevoEstado);
            PedidoManager.getInstancia().notificarObservadores(pedido, nuevoEstado);
        } else {
            System.out.println("----- Pedido no encontrado");
        }
    }
}
```

Diagrama UML

Tecnologías utilizadas

IntelliJ IDEA

GRACIAS!

Seguimiento de estado de pedidos