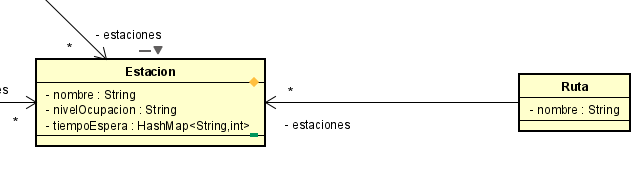
**Taller último Corte**

Nombres: Daniel Diaz y Miguel Motta

Servicios:

# DISEÑO ESTRUCTURAL

**En la relación de Ruta y Estación:**



Estaciones sería una LinkedList, porque se tendría una lista ordenada, dónde una estación conoce la anterior y la siguiente.

**En la relación de Troncal y Estación:**

Diagrama

Descripción generada automáticamente

Estaciones sería una LinkedList, porque se tendría una lista ordenada, dónde una estación conoce la anterior y la siguiente.

**En la relación de Sistema y ruta:**



En esta relación se implementaría un TreeMap como colección, debido a que se realizan varias consultas frecuentes por medio del nombre (llave), en donde consultar en un TreeMap es eficiente. Además, un TreeMap nos permite almacenar datos de manera jerárquica, es decir, en orden.

**Diagrama

Descripción generada automáticamenteEn la relación de Sistema y Troncal:**

Troncales sería un ArrayList<Troncal>, puesto que no hay requerimiento de búsquedas y mantendría flexibilidad del tamaño del arreglo.

**En la relación de Sistema y Estación:**

Diagrama

Descripción generada automáticamenteEn sistema a estaciones implementaríamos un HashMap<> debido que se hacen consultas frecuentes, lo cuál es eficiente en un HashMap, y además, no debe estar ordenado.

# DISEÑO DE COMPORTAMIENTO [en el Diagrama Taller 3.astah y src/domain/\*.java]

# PATRONES

Consulten el patrón estado para completar 2. 3. 4.

1. Nombre:

State

2. Problema: ¿Cuál es el problema que busca resolver?

Resuelve el problema de gestionar el comportamiento de un objeto de manera que su comportamiento pueda cambiar dinámicamente según su estado interno. En otras palabras, este patrón permite que un objeto altere su comportamiento cuando su estado interno cambia.

3. Solución: Definan la solución presentando la estructura propuesta

Diagrama

Descripción generada automáticamente

El **patrón de diseño State** sugiere que crees nuevas clases para todos los posibles estados de un objeto y extraigas todos los comportamientos específicos del estado en estas clases.

En lugar de implementar todos los comportamientos por sí mismo, el objeto original, llamado **contexto**, almacena una referencia a uno de los objetos de estado que representa su estado actual y delega todo el trabajo relacionado con el estado a ese objeto .

4. Ejemplo: Presenten un ejemplo de uso

Para este ejemplo vamos a hacer uso de un banco con personas en una fila. Abran 3 estados para la fila, los cuales serán abierto, cerrado y suspendido. Dependiendo del estado de la fila la persona realizará una acción diferente.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

El método atende() lo realiza la clase de ventanilla.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media

La ventanilla tiene un comportamiento de acuerdo con el estado en el que esté. La ventanilla delega este método al estado en el que esté.

Texto

Descripción generada automáticamente con confianza media

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

**Punto 2:**

1. Nombre:

Composición clase hija

2. Problema: ¿Cuál es el problema que busca resolver?

Cuando se crea una clase, pero no se piensa en extensión se construyen sus métodos manteniendo un margen de extensión.

Pero cuando se tiene una clase de la que se quiere extender, compartiendo atributos y comportamientos, lo que hay que hacer es una herencia, pero si son comportamientos similares, pero con características muy diferentes, es necesario crear una super clase, de la que ambos heredan, normalmente con métodos abstractos. Con este patrón nos adelantamos a ese proceso, permitiendo extensibilidad de tipo para cualquier clase.

3. Solución: Definan la solución presentando la estructura propuesta

Para todo objeto que se cree, se piensa en un diseño de extensión, cuando se crea el objeto, se crea una superclase de ese objeto, y luego se crea la subclase de ese objeto como Objeto normal.

Diagrama

Descripción generada automáticamente

4. Ejemplo: Presenten un ejemplo de uso

// Superclase de Composición

public abstract class ComposicionSuperclase {

    public void metodoComun() {

        System.out.println("Este es un método común para todas las subclases.");

    }

    public abstract void metodoEspecifico();

}

// Subclase de Composición

class ObjetoNormal extends ComposicionSuperclase {

    private String nombre;

    public ObjetoNormal(String nombre) {

        this.nombre = nombre;

    }

    @Override

    public void metodoEspecifico() {

        System.out.println("Implementación específica para " + nombre);

    }

}

// Esta abierto a la posibilidad de extender mas objetos sin necesidad de especificar nuevamente una clase madre

// Ejemplo de Uso

public class EjemploComposicionClaseHija {

    public static void main(String[] args) {

        // Crear una instancia de ObjetoNormal

        ObjetoNormal objeto1 = new ObjetoNormal("Objeto1");

        // Llamar al método común desde la superclase de composición

        objeto1.metodoComun();

        // Llamar al método específico implementado en la subclase

        objeto1.metodoEspecifico();

        // Crear otra instancia de ObjetoNormal

        ObjetoNormal objeto2 = new ObjetoNormal("Objeto2");

        // Llamar al método común desde la superclase de composición

        objeto2.metodoComun();

        // Llamar al método específico implementado en la subclase

        objeto2.metodoEspecifico();

    }

}