**Entrega 1**

**Elaborado por**

Melissa Ortiz Pérez

Braian Camilo Piedrahita Rodríguez

Sebastián Quintero Osorio

**Asignatura**

Ingeniería de software dirigida por modelos

**Programa de Ingeniería de Sistemas y Computación**

**Universidad del Quindío**

**Armenia**

**2022**

**Tabla de contenido**

[**Introducción**](#_xlxxd6sz9nvy) **3**

[1.1 Glosario](#_c213qbj5t3gd) 3

[1.2 Documentos relacionados](#_w0epunby1tf0) 3

[**Enunciado o definición del problema**](#_6107f9wlfm6h) **3**

[**Dominio del Metamodelo**](#_godt4cdwmrj0) **3**

[**Metamodelo**](#_n8v8a6ajrrn2) **3**

[**Sintaxis Abstracta**](#_s6wt57wn3th8) **4**

[5.1 Diagrama de clases de la sintaxis abstracta](#_3oc34jq1j2an) 4

[5.2 Código fuente eugenia de la sintaxis abstracta](#_v21dqak230qx) 4

[5.3 Ejemplo de una producción de la sintaxis abstracta](#_wsmj01877fg0) 7

[5.4 Video de la sintaxis abstracta](#_br4gn6qamuqh) 9

[**Sintaxis Concreta**](#_ngzp0sfjuq8u) **10**

[6.1 Diagrama de clases de la sintaxis concreta](#_9l5tahaz4zjc) 10

[6.2 Código fuente eugenia de la sintaxis concreta](#_jp5dhuddq2pr) 10

[6.3 Ejemplo de una producción de la sintaxis concreta](#_mb9qagh85e75) 14

[6.4 Ejemplo de un diagrama del metamodelo (notación) relativo a la producción anterior](#_66xdceqbjuz4) 16

[6.5 Video de la sintaxis concreta](#_k3kj89l5lb6x) 17

[**Transformación de modelo a modelo (Concreto a Abstracto)**](#_mlnx6ykcv291) **17**

[**Transformación de modelo a texto (Abstracto a Reporte)**](#_2dsnk7gbo8n5) **18**

# Introducción

En este documento se describe brevemente la concepción y elaboración de un dsl para la representación y creación de diagramas de clase. El alcance del documento abarca, el contexto del problema, se define el dominio del metamodelo y el alcance que tendrá. Además, se enuncian las sintaxis abstracta y concreta, las cuales se enuncian brevemente, definiendo su alcance, el código fuente realizado en eugenia y ejemplificando con una producción basada en la creación de cualquier diagrama de clase. Por último se evidencian las transformaciones de modelo a modelo y de modelo a texto.

## 1.1 Glosario y Acrónimos

* **Metamodelo:** Es una estructura de entidades , asociaciones y restricciones que permite representar modelos conceptuales.
* **Diagrama de clases:** Un diagrama de clases en Lenguaje Unificado de Modelado es un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, operaciones, y las relaciones entre los objetos.
* **Sintaxis abstracta:** La sintaxis abstracta describe los conceptos y las relaciones existentes entre modelos.
* **Sintaxis concreta:** Representa de una forma simple los elementos de un metamodelo en base a los modelos mentales.
* **Eugenia:** Eugenia es una herramienta que simplifica el desarrollo de editores de modelos gráficos basados ​​en GMF mediante la generación automática de los *.gmfgraph* modelos *.gmftool* necesarios *.gmfmap* para el editor GMF a partir de un único metamodelo Ecore anotado.
* **Modelo a modelo (M2M):** Intercambio de información o comunicación en formato de datos entre dos máquinas remotas.
* **Modelo a texto (M2T):** Tienen por finalidad la generación de artefactos de tipo texto a partir de los modelos.
* **DSL:** Es un lenguaje creado de tal manera que permite especificar la solución utilizando directamente conceptos del dominio.

## 1.2 Documentos relacionados

A continuación se presentan los documentos relacionados

* Presentación “Desarrollo de metamodelos con EMF”

# Enunciado o definición del problema

En ingeniería de software, un diagrama de clases en Lenguaje Unificado de Modelado (UML) es un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, operaciones (o métodos), y las relaciones entre los objetos.

Como objetivo se quiere realizar un dsl, el cual permite crear diagramas de clases con todos sus componentes (clases, atributos, métodos, relaciones, etc…) y generar el respectivo código en el lenguaje destino. En este caso el lenguaje seleccionado fue Kotlin.

# Dominio del Metamodelo

Un metamodelo es una estructura de entidades (conceptos o clases), asociaciones y restricciones que permite representar modelos conceptuales. Cada entidad tiene un nombre único, un conjunto de atributos, un conjunto de asociaciones con otras entidades y un conjunto de restricciones. Un metamodelo se puede representar utilizando un subconjunto de los diagramas de clases de UML, con el entendido de que el objetivo en este caso no es hacer ningún tipo de implementación.

Es una simplificación de la realidad, obtenida a partir de la aplicación de una serie de abstracciones de la misma, por medio de la cual se puede organizar y entender su estructura, datos y dinámica.

Un modelo permite ocultar la complejidad de la realidad que se puede visualizar a través de los limitados sentidos del hombre, de tal forma que le permiten instanciar y crear estrategias adecuadas para manipular esa realidad al logro de los propósitos deseados.

El metamodelo se compone por:

* **Dominios:** Contiene los dominios en los que se divide el modelo.
* **Entidades:** Las entidades del modelo clasificadas por dominios.
* **Relaciones:** Las relaciones que existen entre las entidades. Se caracterizan por el nombre de la relación, y la cardinalidad de la misma.
* **Atributos:** Los atributos de las entidades y de las relaciones que van a formar parte del modelo. Un atributo en el metamodelo puede ser compartido por una o más de las entidades que forman el modelo.

El dominio del metamodelo es el dominio de la planeación de los recursos de un diagrama, específicamente, el metamodelo que se planteó intenta modelar los recursos que contiene un diagrama de clase para generar el respectivo código fuente, con la posibilidad que cualquier usuario pueda crear un diagrama de clases con sus diferentes atributos, clases y métodos.

# Metamodelo

El metamodelo *MBSDiagramaClases* tendrá la capacidad de representar clases, métodos, parámetros, atributos y paquetes, los cuales se pueden relacionar entre ellos por medio de un tipo de relación. El metamodelo cuenta con una parte parte abstracta la cual tiene la subjetividad del experto de negocio y se encarga de crear n cantidad de diagramas de clases. Por otro lado, se compone de la sintaxis concreta, por medio de esta se crea un editor gráfico y la notación que tendrá el dsl, diseñado para capturar los modelos mentales de los usuarios.

En cuanto al alcance, el dsl permitirá la creación de n cantidad de diagramas de clases, los cuales estarán conformados con métodos, clases, atributos, parámetros y paquetes, todos estos conectados por medio de relaciones, las cuales estarán diferenciadas por su tipo de relación. Un diagrama de clases deberá estar conformado mínimo por una clase.

# Sintaxis Abstracta

La sintaxis abstracta del metamodelo *MBSDiagramaClases*, se compone de todos los elementos que contiene un metamodelo, debe tener una estructura jerárquica que permita a los diagramas que permita crear sus propias clases , métodos, atributos y parámetros. La estructura en la que fue creada la sintaxis abstracta se modeló dependiendo al lenguaje destino.

## 5.1 Diagrama de clases de la sintaxis abstracta

La clase sintaxis abstracta contiene los diagramas de clase, la cual, contiene sus respectivos atributos y las clases de método, relación, atributo, parámetro, paquete y clase.

| diagrama de clases de la sintaxis abstracta |
| --- |
|  |

## 5.2 Código fuente eugenia de la sintaxis abstracta

El código fuente de la sintaxis abstracta se creó por medio de eugenia, el cual cuenta con todos los componentes que conforman un diagrama de clase, mencionados anteriormente.

| código fuente de la sintaxis abstracta |
| --- |
| @namespace(uri="abstracta", prefix="abstracta")  @gmf  package abstracta;  class ModelFactory {    attr String nombre;  attr String ruta;  val MBSPaquete[\*] listaPaquetes;  ref MBSClase[\*] listaTodasLasClases;  ref MBSPaquete[\*] listaTodosLosPaquetes;  }  class MBSPaquete {    attr String nombre;  attr String ruta;  val MBSClase[\*] listaClases;  val MBSPaquete[\*] listaPaquetes;  }  class MBSClase {    attr String nombre;  attr String estereotipo;  attr String modificadorAcceso;  attr boolean abstracta;  attr String descripcion;  attr String ruta;    val MBSAtributo[\*] atributos;  val MBSMetodo[\*] metodos;  val MBSRelacion [\*]listaRelaciones;  }  class MBSMetodo {    attr String nombre;  attr String modificadorAcceso;  attr String tipoRetorno;  attr String ruta;  attr String parametros;  }  class MBSAtributo {    attr String nombre;  attr String tipo;  attr String modificadorAcesso;  attr String ValorDefecto;  attr boolean constante;  attr String ruta;    }  class MBSRelacion {    ref MBSClase source;  ref MBSClase target;  attr String nombre;  attr String tipo;  attr String multiplicidadA;  attr String multiplicidadB;  attr String rolA;  attr String rolB;  attr Boolean navegabilidadA;  attr Boolean navegabilidadB;    } |

## 5.3 Ejemplo de una producción de la sintaxis abstracta

En la producción se ha fabricado un ejemplo de un diagrama de clases dirigido a un sistema de compra de productos por medio de un carrito virtual.

| ejemplo de una producción de la sintaxis abstracta |
| --- |
|  |

## 5.4 Video de la sintaxis abstracta

| video que ilustra la sintaxis abstracta |
| --- |
| [<Enlace al video>](https://youtu.be/CLJUCdVPs_4) |

# Sintaxis Concreta

La sintaxis concreta del metamodelo *MBSDiagramaClases* (notación) se elabora para representar de una forma simple, utilizando rectángulos de un color distintivo, los cuales contienen los atributos y métodos de las clases. Estos rectángulos están relacionados por medio de líneas continuas, las cuales representan la relación de una clase con otra.

## 6.1 Diagrama de clases de la sintaxis concreta

El modelo de clases de la sintaxis concreta mantiene una dependencia taxonómica entre los diagramas de clase y sus componentes (métodos, atributos, paquetes, clases, parámetros), por lo tanto, las clases MBSRelacion y MBSPaquete no pueden ser creadas de manera independiente, debe de existir un diagrama de clases. Además, si se desean crear las clases de MBSMetodo y MBSAtributo, debe de existir la clase MBSClase.

En el diagrama de clases se pueden observar las anotaciones de eugenia que dirigen la generación del diagrama de la notación del metamodelo *MBSDiagramaClases*. Las relaciones en el diagrama de clase no tienen la capacidad de mostrar dichas anotaciones.

| diagrama de clases de la sintaxis concreta |
| --- |
|  |
|  |

## 6.2 Código fuente eugenia de la sintaxis concreta

El código fuente en eugenia contiene los posibles usos de dicho lenguaje para describir la notación del metamodelo *MBSDiagramaClases*.

Adicionalmente, define la configuración de nodos y enlaces a utilizar.

| código fuente de la sintaxis concreta |
| --- |
| @namespace(uri="concreta", prefix="concreta")  @gmf  package concreta;  class ModelFactory{    attr String nombre;  attr String ruta;  val MBSDiagramaClases [\*] listaDiagramas;    }  @gmf.diagram  class MBSDiagramaClases {  attr String nombre;  attr String descripcion;  val MBSClase[\*] listaClases;  val MBSRelacion[\*] listaRelaciones;  val MBSPaquete[\*] listaPaquetes;  }  @gmf.node(label = "nombre")  class MBSPaquete {  attr String nombre;  attr String ruta;  }  @gmf.node(label = "nombre", label.layout= "center", size ="150,150", color ="240,255,255")  class MBSClase {  attr String nombre;  attr String estereotipo;  attr String modificadorAcceso;  attr boolean abstracta;  attr String descripcion;  attr String ruta;    @gmf.compartment(layout="list", collapsible="false")  val MBSAtributo[\*] atributos;  @gmf.compartment(layout="list", collapsible="false")  val MBSMetodo[\*] metodos;  }  @gmf.node(label = "nombre", color ="240,255,255", size ="100,15", margin = "0", border.color = "255,255,255")  class MBSMetodo {    attr String nombre;  attr String modificadorAcceso;  attr String tipoRetorno;  attr String ruta;  attr String parametros;  }  @gmf.node(label = "nombre", color ="240,255,255", size ="100,15", margin = "0", border.color = "255,255,255")  class MBSAtributo {    attr String nombre;  attr String tipo;  attr String modificadorAcesso;  attr String valorDefecto;  attr boolean constante;  attr String ruta;    }  @gmf.link(source="source", target="target", style="solid", width="2")  class MBSRelacion {    attr String nombre;  attr String tipo;  attr String multiplicidadA;  attr String multiplicidadB;  attr String rolA;  attr String rolB;  attr Boolean navegabilidadA;  attr Boolean navegabilidadB;  ref MBSClase source;  ref MBSClase target;  } |

## 6.3 Ejemplo de una producción de la sintaxis concreta

La producción de la sintaxis concreta presenta la configuración de 5 clases, las cuales representan, un usuario, un cliente, el carrito de compras, el pedido y los detalles que tiene el pedido. En la producción se puede evidenciar que existe una jerarquía en donde se puede observar que un diagrama de clases contiene n cantidad de clases y las clases contienen n cantidad de atributos y métodos.

| ejemplo de una producción de la sintaxis concreta |
| --- |
|  |

## 6.4 Ejemplo de un diagrama del metamodelo (notación) relativo a la producción anterior

El diagrama del metamodelo *MBSDiagramaClases* para el caso del servicio bibliotecario se puede observar en la siguiente figura. El metamodelo *MBSDiagramaClases* puede ser utilizado como un control visual para determinar las clases que componen un diagrama. La decoración utilizada en la relación es igual para todas, su factor diferencial es el atributo *tipo*, el cual hace parte de la clase MBSRelacion. La notación utiliza rectángulos para las clases.

El diseño de la notación concreta está centrado en capturar los modelos mentales de los expertos del dominio para los cuales está diseñada. Su expresividad será juzgada, entre varios factores, por la capacidad de la notación de capturar la información del dominio utilizando al máximo el poder de la abstracción sin que por ello se pasen por alto los detalles más relevantes y su respectiva información requerida para realizar una operación concreta.

| ejemplo de un diagrama del metamodelo |
| --- |
|  |

## 6.5 Video de la sintaxis concreta

| video que ilustra la sintaxis abstracta |
| --- |
| [<Enlace al video>](https://youtu.be/9ncrWVEljxM) |

# Transformación de modelo a modelo (Concreto a Abstracto)

Para la transformación de modelo a modelo se creó un plug-in para el proyecto en donde se crearon las respectivas clases para cargar los archivos que se usarían para la transformación (abstracta y concreta), esta clase sería la ModelFactoryModel.java que carga ambos archivos y una vez que se crean los paquetes, clases, métodos y atributos, se guardan en el archivo abstracto. La clase usada para la transformación del modelo concreto a abstracto es la clase TransformacionM2M.java que es la contiene todos los métodos que cargan el model factory concreto y de los diagramas de clase almacenados hace las respectivas conversiones para generar la estructura abstracta.

| código que realiza la transformación concreto a abstracto |
| --- |
| public class TransformacionM2M {  private ModelFactory modelFactoryConcreta;  private abstracta.ModelFactory modelFactoryAbstracta;    public TransformacionM2M(ModelFactory modelFactoryConcreta, abstracta.ModelFactory modelFactoryAbstracta) {  super();  this.modelFactoryConcreta = modelFactoryConcreta;  this.modelFactoryAbstracta = modelFactoryAbstracta;  }    public String transformarM2M() {  String mensaje = "Se ha realizado la transformacion M2M";    if(modelFactoryAbstracta.getListaPaquetes().size()>0) {    if(modelFactoryAbstracta.getListaPaquetes().get(0).getListaClases().size()>0) {    modelFactoryAbstracta.getListaPaquetes().get(0).getListaClases().clear();  }    modelFactoryAbstracta.getListaPaquetes().get(0).getListaPaquetes().clear();  }    if(modelFactoryAbstracta.getListaTodasLasClases().size()>0) {    modelFactoryAbstracta.getListaTodasLasClases().clear();  }    if(modelFactoryAbstracta.getListaTodosLosPaquetes().size()>0) {    modelFactoryAbstracta.getListaTodosLosPaquetes().clear();  }      for (MBSDiagramaClases diagramaConcreta : modelFactoryConcreta.getListaDiagramas()){    //crear los paquetes  for(MBSPaquete paquete : diagramaConcreta.getListaPaquetes()){    System.out.println("Paquete: "+paquete.getNombre());  crearPaquete(paquete);  }    //Crear clases  for(MBSClase clase : diagramaConcreta.getListaClases()){    System.out.println("Clase: "+clase.getNombre());  crearClase(clase);  }    //Crear atributos  for(MBSClase clase : diagramaConcreta.getListaClases()){    if(!clase.getAtributos().isEmpty()) {    for(MBSAtributo atributo : clase.getAtributos()) {    System.out.println("Atributo: "+atributo.getNombre());  crearAtributo(clase.getNombre(), clase.getRuta(), atributo);  }  }  }    //Crear metodos  for(MBSClase clase : diagramaConcreta.getListaClases()){    if(!clase.getMetodos().isEmpty()) {    for(MBSMetodo metodo : clase.getMetodos()) {    System.out.println("Metodo: "+metodo.getNombre());  crearMetodo(clase.getNombre(), clase.getRuta(), metodo);  }  }  }    //Crear relaciones  for(MBSRelacion relacion : diagramaConcreta.getListaRelaciones()) {    System.out.println("Relacion: "+relacion.getNombre());  crearRelacion(relacion);  }      }    return mensaje;  }    private void crearPaquete(MBSPaquete paquete) {    String ruta = "";    if(paquete.getRuta()==null){    ruta = paquete.getNombre();  }else {    ruta = paquete.getRuta()+paquete.getNombre();  }    String[] split = ruta.split("/");  abstracta.MBSPaquete paqueteParent = null;    String nuevaRuta="";    for(int i=0;i<split.length;i++){    String nombrePaquete = split[i];  paqueteParent = obtenerPaqueteAbstracta(nombrePaquete, nuevaRuta, paqueteParent);  nuevaRuta += split[i]+"/";  }    }    private void crearClase(MBSClase clase){    String ruta = clase.getRuta();  abstracta.MBSPaquete paqueteAbstracta = buscarPaqueteClase(ruta);  abstracta.MBSClase claseAbstracta = obtenerClaseAbstracta(ruta, clase.getNombre(), paqueteAbstracta);    if(claseAbstracta==null) {    abstracta.MBSClase nuevaClase = AbstractaFactory.eINSTANCE.createMBSClase();  nuevaClase.setNombre(clase.getNombre());  nuevaClase.setEstereotipo(clase.getEstereotipo());  nuevaClase.setModificadorAcceso(clase.getModificadorAcceso());  nuevaClase.setDescripcion(clase.getDescripcion());  nuevaClase.setAbstracta(clase.isAbstracta());  nuevaClase.setRuta(clase.getRuta());  modelFactoryAbstracta.getListaTodasLasClases().add(nuevaClase);  paqueteAbstracta.getListaClases().add(nuevaClase);  }    }    private void crearAtributo(String nombreClase, String rutaClase, MBSAtributo atributo) {    abstracta.MBSPaquete paqueteAbstracta = buscarPaqueteClase(rutaClase);  abstracta.MBSClase claseAbstracta = obtenerClaseAbstracta(rutaClase, nombreClase, paqueteAbstracta);    if(claseAbstracta != null){    abstracta.MBSAtributo atributoAbstracta = obtenerAtributoAbstracta(atributo.getRuta(), atributo.getNombre(), claseAbstracta);    if(atributoAbstracta == null) {    abstracta.MBSAtributo atributoNuevo = AbstractaFactory.eINSTANCE.createMBSAtributo();  atributoNuevo.setNombre(atributo.getNombre());  atributoNuevo.setModificadorAcesso(atributo.getModificadorAcesso());  atributoNuevo.setRuta(atributo.getRuta());  atributoNuevo.setTipo(atributo.getTipo());  atributoNuevo.setConstante(atributo.isConstante());  atributoNuevo.setValorDefecto(atributo.getValorDefecto());  claseAbstracta.getAtributos().add(atributoNuevo);  }  }  }    private void crearMetodo(String nombreClase, String rutaClase, MBSMetodo metodo) {    abstracta.MBSPaquete paqueteAbstracta = buscarPaqueteClase(rutaClase);  abstracta.MBSClase claseAbstracta = obtenerClaseAbstracta(rutaClase, nombreClase, paqueteAbstracta);    if(claseAbstracta != null) {    abstracta.MBSMetodo metodoAbstracta = obtenerMetodoAbstracta(metodo.getRuta(), metodo.getNombre(), claseAbstracta);    if(metodoAbstracta == null) {    abstracta.MBSMetodo metodoNuevo = AbstractaFactory.eINSTANCE.createMBSMetodo();  metodoNuevo.setNombre(metodo.getNombre());  metodoNuevo.setRuta(metodo.getRuta());  metodoNuevo.setModificadorAcceso(metodo.getModificadorAcceso());  metodoNuevo.setParametros(metodo.getParametros());  metodoNuevo.setTipoRetorno(metodo.getTipoRetorno());  claseAbstracta.getMetodos().add(metodoNuevo);  }  }  }    private void crearRelacion(MBSRelacion relacion) {    MBSClase sourceConcreta = relacion.getSource();  MBSClase targetConcreta = relacion.getTarget();    abstracta.MBSPaquete paqueteAbstractaSource = buscarPaqueteClase(sourceConcreta.getRuta());  abstracta.MBSPaquete paqueteAbstractaTarget = buscarPaqueteClase(targetConcreta.getRuta());  abstracta.MBSClase claseAbstractaSource = obtenerClaseAbstracta(sourceConcreta.getRuta(), sourceConcreta.getNombre(), paqueteAbstractaSource);  abstracta.MBSClase claseAbstractaTarget = obtenerClaseAbstracta(targetConcreta.getRuta(), targetConcreta.getNombre(), paqueteAbstractaTarget);    abstracta.MBSRelacion relacionSource = AbstractaFactory.eINSTANCE.createMBSRelacion();  relacionSource.setNombre(relacion.getNombre());  relacionSource.setMultiplicidadA(relacion.getMultiplicidadA());  relacionSource.setMultiplicidadB(relacion.getMultiplicidadB());  relacionSource.setNavegabilidadA(relacion.getNavegabilidadA());  relacionSource.setNavegabilidadB(relacion.getNavegabilidadB());  relacionSource.setRolA(relacion.getRolA());  relacionSource.setRolB(relacion.getRolB());  relacionSource.setTipo(relacion.getTipo());  relacionSource.setSource(claseAbstractaSource);  relacionSource.setTarget(claseAbstractaTarget);  claseAbstractaSource.getListaRelaciones().add(relacionSource);    if(relacion.getNavegabilidadB()==true) {    abstracta.MBSRelacion relacionTarget = AbstractaFactory.eINSTANCE.createMBSRelacion();  relacionTarget.setNombre(relacion.getNombre());  relacionTarget.setMultiplicidadA(relacion.getMultiplicidadB());  relacionTarget.setMultiplicidadB(relacion.getMultiplicidadA());  relacionTarget.setNavegabilidadA(relacion.getNavegabilidadB());  relacionTarget.setNavegabilidadB(relacion.getNavegabilidadA());  relacionTarget.setRolA(relacion.getRolB());  relacionTarget.setRolB(relacion.getRolA());  relacionTarget.setTipo(relacion.getTipo());  relacionTarget.setSource(claseAbstractaTarget);  relacionTarget.setTarget(claseAbstractaSource);  claseAbstractaTarget.getListaRelaciones().add(relacionTarget);  }    }    private abstracta.MBSPaquete buscarPaqueteClase(String ruta){    abstracta.MBSPaquete MBSPaquete = null;    for(abstracta.MBSPaquete paquete : modelFactoryAbstracta.getListaPaquetes()) {    if(paquete.getNombre().equals(ruta)){    return paquete;  }else {    MBSPaquete = buscarPaquete(ruta, paquete);  }    if(MBSPaquete!=null){    return MBSPaquete;  }    }  return null;  }    private abstracta.MBSPaquete buscarPaquete(String ruta, abstracta.MBSPaquete paquetePadre){    for (abstracta.MBSPaquete p : paquetePadre.getListaPaquetes()){    if((p.getRuta()+p.getNombre()).equals(ruta)){    return p;  }    if(!p.getListaPaquetes().isEmpty()){    abstracta.MBSPaquete paquete = buscarPaquete(ruta, p);    if(paquete!=null) {  return paquete;  }  }  }    return null;  }  private abstracta.MBSPaquete obtenerPaqueteAbstracta(String nombrePaquete, String nuevaRuta,  abstracta.MBSPaquete paqueteParent) {    if(paqueteParent == null) {    for (int i = 0; i < modelFactoryAbstracta.getListaPaquetes().size(); i++) {    if(modelFactoryAbstracta.getListaPaquetes().get(i).getNombre().equals(nombrePaquete)) {    return modelFactoryAbstracta.getListaPaquetes().get(i);  }  }    abstracta.MBSPaquete nuevoPackage = AbstractaFactory.eINSTANCE.createMBSPaquete();  nuevoPackage.setNombre(nombrePaquete);  nuevoPackage.setRuta(nuevaRuta);  modelFactoryAbstracta.getListaPaquetes().add(nuevoPackage);  modelFactoryAbstracta.getListaTodosLosPaquetes().add(nuevoPackage);  return nuevoPackage;  }else{    for(int i = 0; i < paqueteParent.getListaPaquetes().size(); i++){    if(paqueteParent.getListaPaquetes().get(i).getNombre().equals(nombrePaquete)) {    if(paqueteParent.getListaPaquetes().get(i).getRuta().equals(nuevaRuta)){    return paqueteParent.getListaPaquetes().get(i);  }  }  }  }    abstracta.MBSPaquete nuevoPackage = AbstractaFactory.eINSTANCE.createMBSPaquete();  nuevoPackage.setNombre(nombrePaquete);  nuevoPackage.setRuta(nuevaRuta);  paqueteParent.getListaPaquetes().add(nuevoPackage);  modelFactoryAbstracta.getListaTodosLosPaquetes().add(nuevoPackage);    return nuevoPackage;  }      private abstracta.MBSClase obtenerClaseAbstracta(String ruta, String nombre, abstracta.MBSPaquete MBSPaquete) {    if(MBSPaquete != null) {    for(abstracta.MBSClase c : MBSPaquete.getListaClases()) {    if(c.getNombre().equals(nombre) && c.getRuta().equals(ruta)) {    return c;  }  }  }    return null;  }    private abstracta.MBSAtributo obtenerAtributoAbstracta(String ruta, String nombre, abstracta.MBSClase MBSClase){    if(MBSClase != null) {    for(abstracta.MBSAtributo a : MBSClase.getAtributos()) {    if(a.getNombre().equals(nombre) && a.getRuta().equals(ruta)) {    return a;  }  }  }    return null;  }    private abstracta.MBSMetodo obtenerMetodoAbstracta(String ruta, String nombre, abstracta.MBSClase MBSClase){    if(MBSClase != null) {    for(abstracta.MBSMetodo m : MBSClase.getMetodos()) {    if(m.getNombre().equals(nombre) && m.getRuta().equals(nombre)) {    return m;  }  }  }    return null;  }  } |

## 

| video que ilustra la transformación |
| --- |
| [<Enlace al video>](https://youtu.be/f0GVa1eIx0k) |

# Transformación de modelo a texto (Abstracto a Reporte)

Para la transformación de modelo a texto, se creó una clase nueva en el plug-in mencionado en la parte de la transformación de modelo a modelo, dicha clase se llama TransformacionM2T.java, que toma el model factory de la parte abstracta y crea el código de las clases creadas y las guarda en los paquetes que están en las rutas especificadas en el modelo.

| código que realiza la transformación abstracto a reporte |
| --- |
| public class TransformacionM2T {    private abstracta.ModelFactory modelFactoryAbstracta;    public TransformacionM2T(abstracta.ModelFactory modelFactoryAbstracta) {  super();  this.modelFactoryAbstracta = modelFactoryAbstracta;  }    public String transformarM2T() {    String pathRaiz = "";  DirectoryDialog fd = new DirectoryDialog(new Shell(), SWT.SELECTED);  fd.setText("Generacion de codigo");  pathRaiz = fd.open();    for (abstracta.MBSClase clase : modelFactoryAbstracta.getListaTodasLasClases()){    StringBuilder textoCodigo = new StringBuilder();  generarClase(clase, textoCodigo);  guardarArchivo(textoCodigo.toString(), pathRaiz+"/"+clase.getRuta(), clase.getNombre());  }    return "Se ha generado el codigo de su proyecto";  }  private void generarClase(abstracta.MBSClase clase, StringBuilder textoCodigo) {    //Agregar la declaracion del paquete  if(!clase.getRuta().equals("")) {    String[] split = clase.getRuta().split("/");  String ruta="";    for(int i=0;i<split.length;i++) {    ruta+=split[i];    if(i!=split.length-1) {  ruta+=".";  }  }    textoCodigo.append("package "+ruta+"\n\n");  }    //Agregar el encabezado de la clase  agregarEncabezado(clase, textoCodigo);    //Agregar los atributos de la clase  agregarAtributos(clase, textoCodigo);    //Agregar el constructor de la clase  agregarConstructor(clase, textoCodigo);    //Agregar los set y get de los atributos  agregarSetyGet(clase, textoCodigo);    //Agregar los metodos de la clase  agregarMetodos(clase, textoCodigo);    textoCodigo.append("}");    }  private void agregarMetodos(abstracta.MBSClase clase, StringBuilder textoCodigo) {    for (abstracta.MBSMetodo metodo : clase.getMetodos()) {    if(metodo.getTipoRetorno().equals("") || metodo.getTipoRetorno().equals("void")) {    textoCodigo.append("\t"+metodo.getModificadorAcceso()+" fun "+metodo.getNombre()+"("+metodo.getParametros()+"){\n");  textoCodigo.append("\t\t"+"// TODO: Autogenerado\n");  textoCodigo.append("\t}\n\n");  }else {    textoCodigo.append("\t"+metodo.getModificadorAcceso()+" fun "+metodo.getNombre()+"("+metodo.getParametros()+")"+": "+metodo.getTipoRetorno()+" {\n");    if(metodo.getTipoRetorno().equals("String")) {    textoCodigo.append("\t\t"+"// TODO: Autogenerado\n");  textoCodigo.append("\t\t"+"return "+"\"\""+"\n");  }else if(metodo.getTipoRetorno().equals("Int")) {    textoCodigo.append("\t\t"+"// TODO: Autogenerado\n");  textoCodigo.append("\t\t"+"return 0"+"\n");  }else if(metodo.getTipoRetorno().equals("Float") || metodo.getTipoRetorno().equals("Double")) {    textoCodigo.append("\t\t"+"// TODO: Autogenerado\n");  textoCodigo.append("\t\t"+"return 0.0"+"\n");  }else if(metodo.getTipoRetorno().equals("Char")) {    textoCodigo.append("\t\t"+"// TODO: Autogenerado\n");  textoCodigo.append("\t\t"+"return "+"\'\'"+"\n");  }else if(metodo.getTipoRetorno().equals("Boolean")) {    textoCodigo.append("\t\t"+"// TODO: Autogenerado\n");  textoCodigo.append("\t\t"+"return True"+"\n");  }else {    textoCodigo.append("\t\t"+"// TODO: Autogenerado\n");  textoCodigo.append("\t\t"+"return null"+"\n");  }    textoCodigo.append("\t}\n\n");  }    }    }  private void agregarSetyGet(abstracta.MBSClase clase, StringBuilder textoCodigo) {    for (abstracta.MBSAtributo atributo : clase.getAtributos()) {    textoCodigo.append("\t"+"public fun "+"get"+atributo.getNombre()+"(): "+atributo.getTipo()+"{\n");  textoCodigo.append("\t\t"+"return "+atributo.getNombre()+"\n");  textoCodigo.append("\t"+"}\n\n");    textoCodigo.append("\t"+"public fun "+"set"+atributo.getNombre()+"("+atributo.getNombre()+": "+atributo.getTipo()+"){\n");  textoCodigo.append("\t\t"+"this."+atributo.getNombre()+" = "+atributo.getNombre()+"\n");  textoCodigo.append("\t"+"}\n\n");  }    for(abstracta.MBSRelacion relacion : clase.getListaRelaciones()) {    if(!relacion.getTipo().equals("herencia")) {    if(relacion.getMultiplicidadB().equals("\*")) {  textoCodigo.append("\t"+"public fun "+"get"+relacion.getRolB()+"():"+" List<"+relacion.getTarget().getNombre()+">"+"{\n");  textoCodigo.append("\t\t"+"return "+relacion.getRolB()+"\n");  textoCodigo.append("\t"+"}\n\n");    textoCodigo.append("\t"+"public fun "+"set"+relacion.getRolB()+"("+relacion.getRolB()+": List<"+relacion.getTarget().getNombre()+">"+")"+"{\n");  textoCodigo.append("\t\t"+"this."+relacion.getRolB()+" = "+relacion.getRolB()+"\n");  textoCodigo.append("\t"+"}\n\n");  }else {  textoCodigo.append("\t"+"public fun "+"get"+relacion.getRolB()+"(): "+relacion.getTarget().getNombre()+"{\n");  textoCodigo.append("\t\t"+"return "+relacion.getRolB()+"\n");  textoCodigo.append("\t}\n\n");    textoCodigo.append("\t"+"public fun "+"set"+relacion.getRolB()+"("+relacion.getRolB()+": "+relacion.getTarget().getNombre()+")"+"{\n");  textoCodigo.append("\t\t"+"this."+relacion.getRolB()+" = "+relacion.getRolB()+"\n");  textoCodigo.append("\t}\n\n");  }  }  }    }  private void agregarConstructor(abstracta.MBSClase clase, StringBuilder textoCodigo) {    abstracta.MBSRelacion herencia=obtenerHerencia(clase);    textoCodigo.append("\t"+"constructor(");    if(herencia!=null) {    abstracta.MBSClase claseTarget = herencia.getTarget();    for(int i=0;i<claseTarget.getAtributos().size();i++) {    textoCodigo.append(claseTarget.getAtributos().get(i).getNombre()+": "+claseTarget.getAtributos().get(i).getTipo()+", ");  }    for(int i=0;i<clase.getListaRelaciones().size();i++) {    if(!clase.getListaRelaciones().get(i).getTipo().equals("herencia")) {    if(clase.getListaRelaciones().get(i).getMultiplicidadB().equals("\*")) {    textoCodigo.append(clase.getListaRelaciones().get(i).getRolB()+": List<"+clase.getListaRelaciones().get(i).getTarget().getNombre()+">, ");    }else {    textoCodigo.append(clase.getListaRelaciones().get(i).getRolB()+": "+clase.getListaRelaciones().get(i).getTarget().getNombre()+", ");  }  }  }    for(int i=0;i<clase.getAtributos().size();i++) {    textoCodigo.append(clase.getAtributos().get(i).getNombre()+": "+clase.getAtributos().get(i).getTipo());    if(i!=clase.getAtributos().size()-1) {  textoCodigo.append(", ");  }  }    textoCodigo.append("){\n");  textoCodigo.append("\t\t"+"super(");    for(int i=0;i<claseTarget.getAtributos().size();i++) {    textoCodigo.append(claseTarget.getAtributos().get(i).getNombre());    if(i!=claseTarget.getAtributos().size()-1) {  textoCodigo.append(", ");  }    if(i==claseTarget.getAtributos().size()-1) {  textoCodigo.append(")\n");  }  }    for(int i=0;i<clase.getListaRelaciones().size();i++) {    if(!clase.getListaRelaciones().get(i).getTipo().equals("herencia")) {    textoCodigo.append("\t\t"+"this."+clase.getListaRelaciones().get(i).getRolB()+" = "+clase.getListaRelaciones().get(i).getRolB()+"\n");  }  }    for(int i=0;i<clase.getAtributos().size();i++) {    textoCodigo.append("\t\t"+"this."+clase.getAtributos().get(i).getNombre()+" = "+clase.getAtributos().get(i).getNombre()+"\n");  }    textoCodigo.append("\t"+"}\n\n");    }else {    for(int i=0;i<clase.getListaRelaciones().size();i++) {    if(!clase.getListaRelaciones().get(i).getTipo().equals("herencia")) {    if(clase.getListaRelaciones().get(i).getMultiplicidadB().equals("\*")) {    textoCodigo.append(clase.getListaRelaciones().get(i).getRolB()+": List<"+clase.getListaRelaciones().get(i).getTarget().getNombre()+">, ");    }else {    textoCodigo.append(clase.getListaRelaciones().get(i).getRolB()+": "+clase.getListaRelaciones().get(i).getTarget().getNombre()+", ");  }  }  }    for(int i=0;i<clase.getAtributos().size();i++) {    textoCodigo.append(clase.getAtributos().get(i).getNombre()+": "+clase.getAtributos().get(i).getTipo());    if(i!=clase.getAtributos().size()-1) {  textoCodigo.append(", ");  }  }    textoCodigo.append("){\n");  textoCodigo.append("\t\t"+"super()\n");    for(int i=0;i<clase.getListaRelaciones().size();i++) {    if(!clase.getListaRelaciones().get(i).getTipo().equals("herencia")) {    textoCodigo.append("\t\t"+"this."+clase.getListaRelaciones().get(i).getRolB()+" = "+clase.getListaRelaciones().get(i).getRolB()+"\n");  }  }    for(int i=0;i<clase.getAtributos().size();i++) {    textoCodigo.append("\t\t"+"this."+clase.getAtributos().get(i).getNombre()+" = "+clase.getAtributos().get(i).getNombre()+"\n");  }    textoCodigo.append("\t"+"}\n\n");  }    }  private void agregarAtributos(abstracta.MBSClase clase, StringBuilder textoCodigo) {    for (abstracta.MBSAtributo atributo : clase.getAtributos()) {  textoCodigo.append("\t"+atributo.getModificadorAcesso()+" val "+atributo.getNombre()+": "+atributo.getTipo() +"\n");  }    for (abstracta.MBSRelacion relacion : clase.getListaRelaciones()) {  if(!relacion.getTipo().equals("herencia")) {    if(relacion.getMultiplicidadB().equals("\*")) {    textoCodigo.append("\t"+"private val "+relacion.getRolB()+": List<"+relacion.getTarget().getNombre()+">"+"\n");  }else {    textoCodigo.append("\t"+"private val "+relacion.getRolB()+": "+relacion.getTarget().getNombre()+"\n");  }    }  }    textoCodigo.append("\n");    }  private void agregarEncabezado(abstracta.MBSClase clase, StringBuilder textoCodigo) {    if(clase.getListaRelaciones().size()>0) {    abstracta.MBSRelacion herencia = obtenerHerencia(clase);    if(herencia!=null) {    abstracta.MBSClase claseTarget = herencia.getTarget();  textoCodigo.append(clase.getModificadorAcceso()+" class "+clase.getNombre()+" : "+claseTarget.getNombre()+" {\n\n");    }else {    textoCodigo.append(clase.getModificadorAcceso()+" class "+clase.getNombre()+" {\n\n");  }    }else {    textoCodigo.append(clase.getModificadorAcceso()+" class "+clase.getNombre()+" {\n\n");  }    }    private abstracta.MBSRelacion obtenerHerencia(abstracta.MBSClase clase){    for (abstracta.MBSRelacion relacion : clase.getListaRelaciones()) {    if(relacion.getTipo().equals("herencia")) {    return relacion;  }  }    return null;  }    private void guardarArchivo(String cadena, String ruta, String nombre) {  try  {  //Crear un objeto File se encarga de crear o abrir acceso a un archivo que se especifica en su constructor    File archivo=new File(ruta);  System.out.println("");  if(archivo.exists() == false) {  archivo.mkdirs();  }  //Crear objeto FileWriter que sera el que nos ayude a escribir sobre archivo  FileWriter escribir=new FileWriter(archivo+"/"+nombre+".kt",true);  //Escribimos en el archivo con el metodo write  escribir.write(cadena);  //Cerramos la conexion  escribir.close();  }  //Si existe un problema al escribir cae aqui  catch(Exception e)  {  System.out.println("Error al Guardar");  }  }  } |

## 

| video que ilustra la transformación |
| --- |
| [<Enlace al video>](https://youtu.be/A1HKweQulqI) |