Crear una aplicación JPA, DAO con una vista de terminal.

Requerimiento: Crear una base de datos con las tablas llamadas Cliente y Correo, agregar las funcionalidades GRUD a cada tabla, la relación entre tablas es: un Cliente puede tener muchos correos.

Esquema de la base de datos

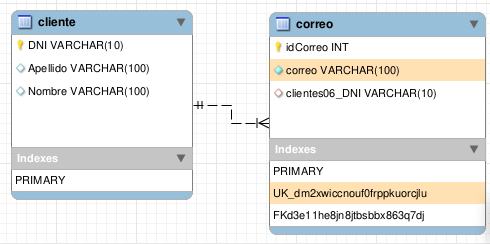
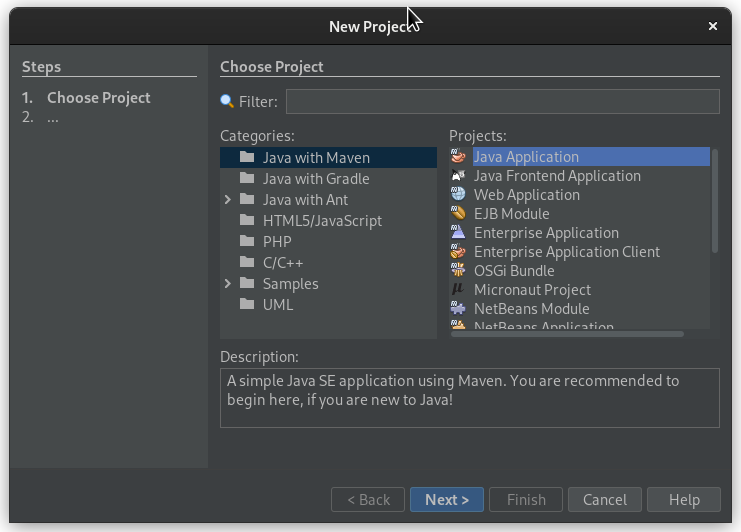


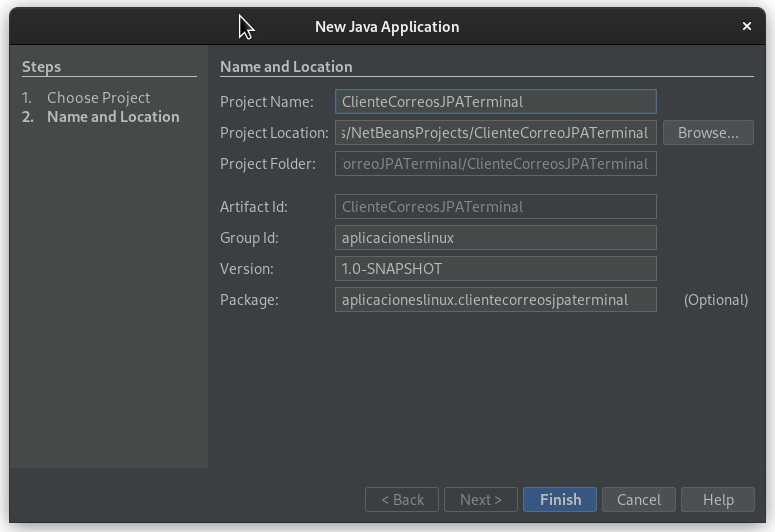
Diagrama de clases

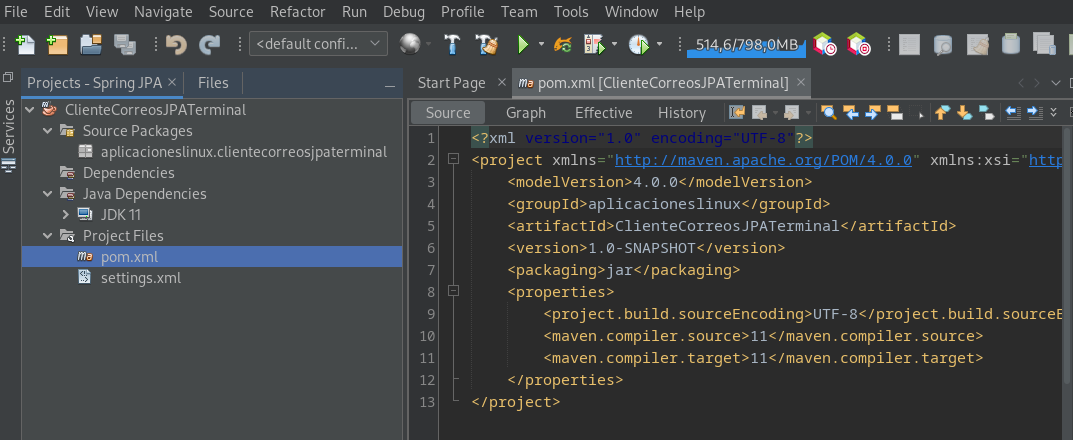


Paso N.º 01 – Crear el proyecto POM.xml

Crear un nuevo proyecto en NetBeans







El archivo POM.xml creado es muy básico, hay que agregar las dependencia faltantes y la configuración de las estructuras de paquetes para los test y para localizar la clase que contiene el método main().

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>aplicacioneslinux</groupId>

<artifactId>ClienteCorreosJPATerminal</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

</properties>

</project>

Este es el POM.xml que incluye las ordenes básicas para compilar, crear el archivo jar y encontrar el método main() para ejecutar. Esta configuración en el archivo pom.xml es usada para crear el archivo de manifiesto en el directorio META-INF.

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>AplicacionesLinux</groupId>

<artifactId>ClienteCorreosJPATerminal</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

</properties>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<release>11</release>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-jar-plugin</artifactId>

<version>3.2.0</version>

<configuration>

<archive>

<manifest>

<addClasspath>true</addClasspath>

<classpathPrefix>lib/</classpathPrefix>

<mainClass>aplicacioneslinux.clientecorreosjpaterminal.MainClass</mainClass>

</manifest>

</archive>

</configuration>

</plugin>

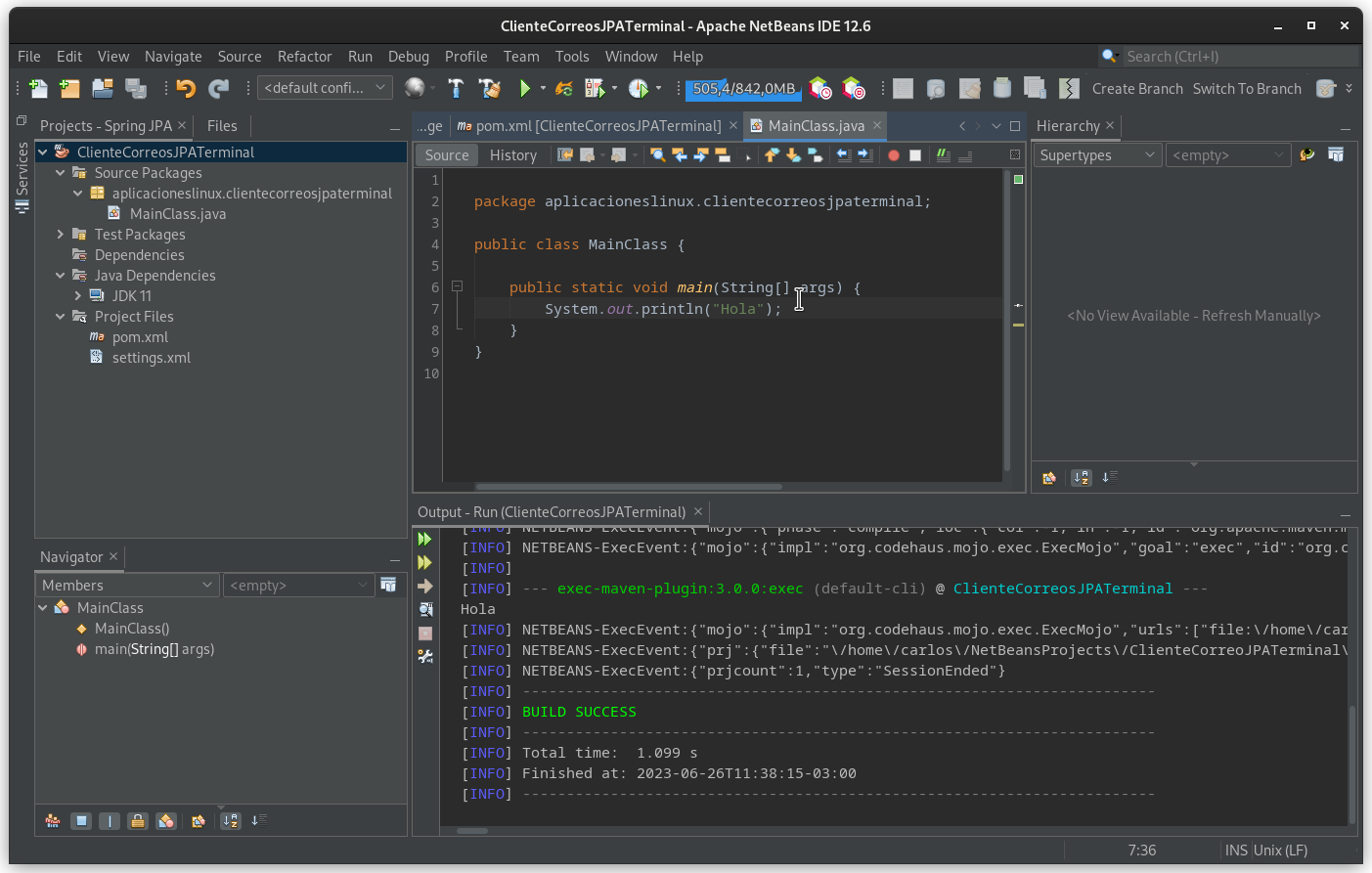
</plugins>

</build>

</project>

Agregar una clase con el método main() según se ha especificado en el archivo POM.xml y correr el programa para verificar que compila.

<mainClass>aplicacioneslinux.clientecorreosjpaterminal.MainClass</mainClass>



Realizar un clean and build para generar el archivo jar. Ejecutar el archivo jar y verificar que está todo bien.

Configuración adicional pero no obligatoria en el archivo POM.xml

El archivo POM.xml debe ubicarse en el directorio raíz del proyecto, el directorio raiz es conocido normalmente como el “resource directory”. Dicho directorio debe configurarse en las propiedades del proyecto en el IDE y en el archivo POM.xml.

Estas configuraciones en el archivo pom.xml que no son necesarias, pero en algunas ocasiones los IDEs requieren se sean configuradas.

El siguiente fragmento del archivo POM.xml muestra la configuración de los directorios más habituales para configurar:

* Directorio raíz del proyecto <directory>${basedir}</directory>
* Directorio de los fuentes el proyecto <sourceDirectory>src/main/java</sourceDirectory>
* Directorio de salida del proyecto <outputDirectory>target/classes/</outputDirectory>

…

<build>

<sourceDirectory>src/main/java</sourceDirectory>

<outputDirectory>target/classes/</outputDirectory>

<resources>

<resource>

<directory>${basedir}</directory>

<includes>

<include>\*</include>

</includes>

<filtering>true</filtering>

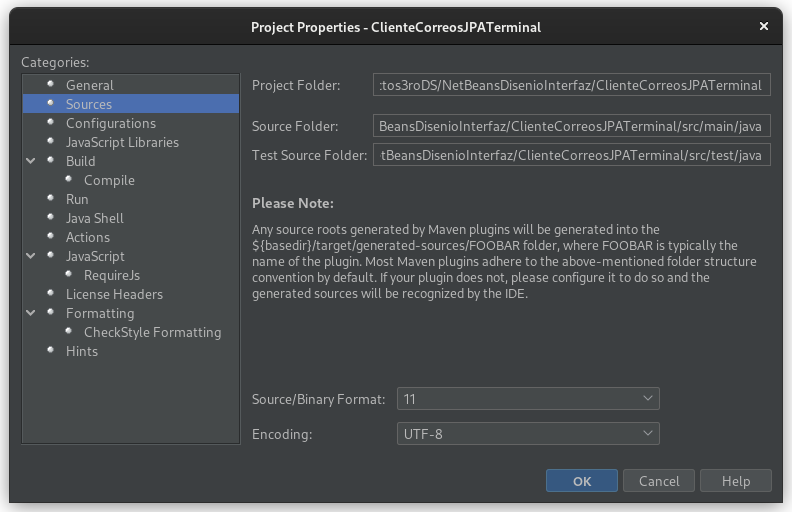
</resource>

</resources>

</build>

…

En la imagen podemos ver la configuración de las propiedades del proyecto en NetBeans.



Source Folder: .../ClienteCorreosJPATerminal/src/main/java

../ClienteCorreosJPATerminal/... es el directorio raiz del proyecto, también es conocido como directorio base.

.../src/main/java/… es el directorio para los archivos fuentes y para otros paquetes con archivos fuentes o recursos para compilar o incluir en el archivo JAR.

Fin del Paso N.º 01

Paso N.º 02 – Crear el archivo persistence.xml

<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence

http://xmlns.jcp.org/xml/ns/persistence/persistence\_2\_1.xsd">

<persistence-unit name="DBCliente" transaction-type="RESOURCE\_LOCAL">

<provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

<properties>

<property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/dbcliente?serverTimezone=UTC"/>

<property name="javax.persistence.jdbc.user" value="carlos"/>

<property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>

<property name="javax.persistence.jdbc.password" value="12345678"/>

<property name="javax.persistence.schema-generation.database.action" value="update"/>

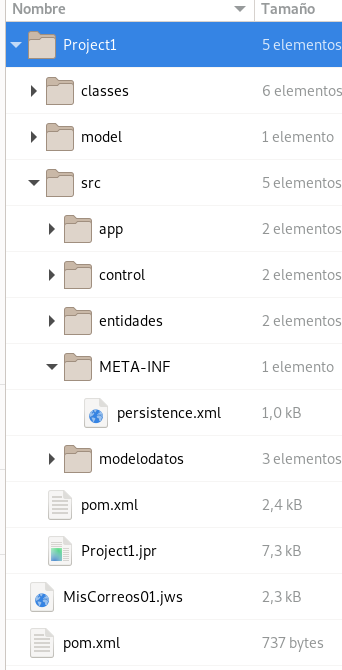
<property name="hibernate.dialect" value="org.hibernate.dialect.MySQL8Dialect" />

</properties>

</persistence-unit>

</persistence>

La siguiente imagen muestra la ubicación del archivo persistence.xml. El mencionado archivo se localiza en el directorio META-INF.



El directorio META-INF debe localizarse en algún directorio que esté indicado como “source directory” en las propiedades del proyecto en el IDE y en el archivo POM.xml.

…

<sourceDirectory>src/main/java</sourceDirectory>

…

Paso N.º 03 – Crear la clase DaoCRUD03

package modelodatos;

import java.util.List;

public interface DAOGRUD03<T> {

//Métodos abstractos GRUD create, read, update, delete

void guardar(T t); //Create //Crear

List<T> listarTodo();//Read //Leer

void editar(T t); //Update //Actualizar

void borrar(T t); //Delete //Borrar

///////////////////////////////////////

//Otros métodos relacionados con los requerimientos del sistema o reglas de negocios

///////////////////////////////////////

}

Paso N.º 04 – Crear la clase EntityManagerUtil04

package modelodatos;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

public class EntityManagerUtil04 {

public EntityManagerUtil04() {

super();

}

private static EntityManager manager = null;

public static EntityManager getEntityManager() {

if(manager==null) {

EntityManagerFactory factory = Persistence.createEntityManagerFactory("DBCliente");

manager = factory.createEntityManager();

}

return manager;

}

//public static void main(String[] args) {

// EntityManager manager = EntityManagerUtil04.getEntityManager();

// System.out.println("Se ha creado un objeto EntityManager ==> " + manager.getClass().getCanonicalName());

//}

}

Paso N.º 05 – Crear la clase AbstractDAO05

package modelodatos;

import java.util.List;

import java.util.function.Consumer;

import javax.persistence.EntityManager;

import javax.persistence.EntityTransaction;

import javax.persistence.Query;

public class AbstractDAO05<T> implements DAOGRUD03<T> {

private EntityManager entityManager = EntityManagerUtil04.getEntityManager();

private Class<T> entidadTipoT; //Cliente{} o Correo{} o OtraClase{} o ...

public Class<T> getEntidadTipoT() { //Pregunta con la entidad que se está trabajando

return entidadTipoT;

}

public void setEntidadTipoT(Class<T> entidadTipoT) { //Asignar una entidad de algún tipo T

this.entidadTipoT = entidadTipoT;

}

public AbstractDAO05() {

super();

}

//Implementar los métodos CRUD heredades desde la interface

//Estos métodos pueden ser sobre-escritos desde las entidades tipo T

@Override

public List<T> listarTodo() {

// TODO Implement this method

String sqlString = "FROM " + entidadTipoT.getName();

Query query = entityManager.createQuery(sqlString);

return query.getResultList();

}

@Override

public void guardar(Object t) {

// TODO Implement this method

executeInsideTransaccion(entityManager -> entityManager.persist(t));

}

@Override

public void editar(Object t) {

// TODO Implement this method

executeInsideTransaccion(entityManager -> entityManager.merge(t));

}

@Override

public void borrar(Object t) {

// TODO Implement this method

executeInsideTransaccion(entityManager -> entityManager.remove(t));

}

//Transacciones que se pueden deshacer

private void executeInsideTransaccion(Consumer<EntityManager> action) {

EntityTransaction tx = entityManager.getTransaction();

try {

tx.begin();

action.accept(entityManager);

tx.commit();

} catch (RuntimeException er) {

tx.rollback();

throw er;

} catch (Exception er) {

tx.rollback();

throw er;

}

}

public EntityManager getEntityManager() {

return entityManager;

}

///////////////////////////////////////

//Otros métodos relacionados con los requerimientos del sistema o reglas de negocios

// globales a todas las entidades

///////////////////////////////////////

}

Paso N.º 06 – Crear las clases que son entidades del dominio del problema a resolver

Primero crear las entidades que no se relacionan con otras entidades

package entidades;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name = "cliente" )

public class Clientes06 {

public Clientes06() {

super();

}

public Clientes06(String dni, String nombre, String apellido) {

this.dni = dni;

this.nombre = nombre;

this.apellido = apellido;

}

@Column(name="DNI", nullable=false)

@Id

private String dni;

@Column(name="Nombre")

private String nombre;

@Column(name="Apellido")

private String apellido;

public void setDni(String dni) {

this.dni = dni;

}

public String getDni() {

return dni;

}

public void setNombre(String nombre) {

this.nombre = nombre;

}

public String getNombre() {

return nombre;

}

public void setApellido(String apellido) {

this.apellido = apellido;

}

public String getApellido() {

return apellido;

}

@Override

public String toString() {

// TODO Implement this method

return "id=" + dni + ", Nombre=" + nombre + ", " + apellido + "\n";

}

}

package entidades;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.FetchType;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.ManyToOne;

import javax.persistence.Table;

@Entity

@Table(name = "correo")

public class Correos06 {

public Correos06() {

super();

}

@Id

@Column(name = "idCorreo", length = 6)

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int idCorreo;

@Column(name = "correo", length = 100, unique=true, nullable=false)

private String correo;

@ManyToOne(fetch = FetchType.LAZY) //Un cliente muchos correos

private Clientes06 clientes06=null; //Se crea un objeto tipo Cliente06{}

public Correos06(String correo, Clientes06 clientes06) {

super();

this.correo = correo;

this.clientes06 = clientes06; //El cliente relacionado con este correo

}

public void setIdCorreo(int idCorreo) {

this.idCorreo = idCorreo;

}

public int getIdCorreo() {

return idCorreo;

}

public void setCorreo(String correo) {

this.correo = correo;

}

public String getCorreo() {

return correo;

}

public void setClientes06(Clientes06 clientes06) {

this.clientes06 = clientes06;

}

public Clientes06 getClientes06() {

return clientes06;

}

@Override

public String toString() {

// TODO Implement this method

return "id=" + idCorreo + ", Correo=" + correo + "\n";

}

}

Paso N.º 07 – Crear todas las clases controlador de cada entidad

package control;

import modelodatos.AbstractDAO05;

import entidades.Clientes06;

import java.util.List;

import javax.persistence.Query;

public class ClientesDAO07 extends AbstractDAO05<Clientes06> {

public ClientesDAO07() {

super();

this.setEntidadTipoT(Clientes06.class);

}

////Escribir los métodos que son reglas de negocios personalizados para los clientes

public Clientes06 unCliente(String dni) {

String sql = "FROM " + Clientes06.class.getName() + " WHERE dni='" + dni + "'";

Query query = getEntityManager().createQuery(sql).setMaxResults(1);

return (Clientes06) query.getSingleResult();

}

public List<Clientes06> listarClientes() {

String sql = "FROM " + Clientes06.class.getName();

Query query = getEntityManager().createQuery(sql);

return query.getResultList();

}

///////////////////////////////////////

//Opcional: Sobre-escribir los métodos CRUD heredados de la clase abstracta

////////////////////////////////////////////////////

}

package control;

import modelodatos.AbstractDAO05;

import entidades.Correos06;

import java.util.List;

import javax.persistence.Query;

public class CorreosDAO07 extends AbstractDAO05<Correos06> {

public CorreosDAO07() {

super();

this.setEntidadTipoT(Correos06.class);

}

////Escribir los métodos que son reglas de negocios personalizados para los correos

public Correos06 unCorreo(int idCorreo) {

String sql = "FROM " + Correos06.class.getName() + " WHERE idCorreo=" + idCorreo;

Query query = getEntityManager().createQuery(sql).setMaxResults(1);

return (Correos06) query.getSingleResult();

}

public List<Correos06> listarCorreos(String dni) {

String sql = "FROM " + Correos06.class.getName() + " WHERE clientes06\_DNI='" + dni + "'";

Query query = getEntityManager().createQuery(sql);

return query.getResultList();

}

//////////////////////////////////////

//Opcional: Sobre-escribir los métodos CRUD heredados de la clase abstracta

////////////////////////////////////////////////////

}

Paso N.º 08 – Crear las distintas vistas: Terminal

package app;

import control.ClientesDAO07;

import control.CorreosDAO07;

import entidades.Clientes06;

import entidades.Correos06;

import java.util.List;

import javax.persistence.RollbackException;

public class App08 {

public App08() {

super();

}

private static CorreosDAO07 correosDAO=null;

private static ClientesDAO07 clienteDAO=null;

private static Correos06 correo=null;

private static Clientes06 cliente=null;

public static void main(String[] args) {

App08 app08 = new App08();

System.out.println("Inicio");

List l = null;

clienteDAO = new ClientesDAO07();

correosDAO = new CorreosDAO07();

//Dar de alta a un cliente y su correo

cliente = new Clientes06("32", "Juan", "Pérez");

clienteDAO.guardar(cliente);

correo = new Correos06("ss4ss@dddd.com", cliente);

correosDAO.guardar(correo);

//dar de alta a un cliente sin correo

cliente = new Clientes06("20", "Pedro", "Martínez");

clienteDAO.guardar(cliente);

//Dar de alta un nuevo correo al cliente con DNI 32

cliente = new Clientes06("32", "", "");

correo = new Correos06("ss44ss@dddd44.com", cliente);

correosDAO.guardar(correo);

System.out.println("Listar todos los clientes");

l = clienteDAO.listarClientes();

System.out.println(l);

System.out.println("Listar un clientes");

Clientes06 unCli = clienteDAO.unCliente("32");

System.out.println(unCli.toString());

System.out.println("Listar los correos de un cliente");

String algunDNICliente = "32";

l = correosDAO.listarCorreos(algunDNICliente);

System.out.println(l);

System.out.println("Listar un Correo");

int idCorreo = 1;

Correos06 unCorreo = correosDAO.unCorreo(idCorreo);

System.out.println(unCorreo.toString());

//Borrar el primer correo ss4ss@dddd.com para el cliente DNI 32

idCorreo = 1; //Hay que saber el idCorreo para borrar

correo = correosDAO.unCorreo(idCorreo);

correosDAO.borrar(correo);

//Borrar un cliente

try {

String idCliente = "32"; //Hay que saber el idCleinte para borrar

System.out.println("Borrando el cliente: " + idCliente);

cliente = clienteDAO.unCliente(idCliente);

clienteDAO.borrar(cliente);

}catch(RollbackException err) {

System.out.println("Error: mo es posible borrar un cliente con correos asignados. " +

err.getMessage());

}catch(Exception err) {

System.out.println("Error: " + err.getMessage());

}

}

}

package app;

import control.ClientesDAO07;

import control.CorreosDAO07;

import entidades.Clientes06;

import entidades.Correos06;

import java.util.List;

import java.util.Scanner;

public class AppVista08 {

private static List l = null;

private static CorreosDAO07 correosDAO=null;

private static ClientesDAO07 clienteDAO=null;

private static Correos06 correo=null;

private static Clientes06 cliente=null;

public AppVista08() {

super();

}

public static void main(String[] args) {

AppVista08 appVista08 = new AppVista08();

clienteDAO = new ClientesDAO07();

correosDAO = new CorreosDAO07();

int teclaNum = 0;

Scanner miTeclado = new Scanner(System.in);

do {

System.out.print("[1]Alta cliente [2]Listar Clientes [3]Salir ==> ");

teclaNum = miTeclado.nextInt();

switch(teclaNum) {

case 1: altaCliente(); break;

case 2: listarClientes(); break;

case 3: break;

default: break;

}

}while(teclaNum != 3);

System.out.println("El programa ha terminado normalmente");

}

private static void altaCliente() {

String teclasStr = "";

Scanner miTeclado = new Scanner(System.in);

try {

System.out.print("Escribir el DNI del Cliente ==> ");

teclasStr = miTeclado.nextLine();

String dni = teclasStr;

System.out.print("Escribir el nombre del Cliente ==> ");

teclasStr = miTeclado.nextLine();

String nombre = teclasStr;

System.out.print("Escribir el apellido del Cliente ==> ");

teclasStr = miTeclado.nextLine();

String apellido = teclasStr;

cliente = new Clientes06(dni, nombre, apellido);

clienteDAO.guardar(cliente);

}catch(Exception err){

System.out.println("Error de entrada de teclado o alta cliente " + err.getMessage());

}

}

private static void listarClientes() {

//l = clienteDAO.listarClientes();

l = clienteDAO.listarTodo();

System.out.println(l);

int teclaNum = 0;

Scanner miTeclado = new Scanner(System.in);

do {

System.out.print("[1]Alta Correo [2]Listar Correos [3]Salir ==> ");

teclaNum = miTeclado.nextInt();

switch(teclaNum) {

case 1: altaCorreo(); break;

case 2: listarCorreos(); break;

case 3: break;

default: break;

}

}while(teclaNum != 3);

}

private static void altaCorreo() {

String teclasStr = "";

Scanner miTeclado = new Scanner(System.in);

try {

System.out.print("Escribir el DNI del Cliente ==> ");

teclasStr = miTeclado.nextLine();

String dni = teclasStr;

System.out.print("Escribir el queCorreo del Cliente ==> ");

teclasStr = miTeclado.nextLine();

String queCorreo = teclasStr;

cliente = new Clientes06(dni, "", "");

correo = new Correos06(queCorreo, cliente);

correosDAO.guardar(correo);

}catch(Exception err){

System.out.println("Error de entrada de teclado o alta cliente " + err.getMessage());

}

}

private static void listarCorreos() {

String teclasStr = "";

Scanner miTeclado = new Scanner(System.in);

System.out.print("Escribir el DNI del Cliente ==> ");

teclasStr = miTeclado.nextLine();

String algunDNICliente = teclasStr;

l = correosDAO.listarCorreos(algunDNICliente);

System.out.println(l);

}

}

Paso 9 – Crear el diagrama de clases UML