Programación de aplicaciones (15/04/2013)

Detalle de la aplicación de patrón **Singleton** sobre un contenedor de entidades (ContenedorClientes) en el proyecto.

```
public class ContenedorClientesSingleton {

    private LinkedList<Cliente> coleccionElementos;


    private static ContenedorClientesSingleton _instancia;

private ContenedorClientesSingleton(){

        coleccionElementos = new LinkedList();

}

public static ContenedorClientesSingleton getInstancia(){

if (_instancia == null) {
                //Si la instancia no se ha creado crearla
                _instancia = new ContenedorClientesSingleton();
  }
 return _instancia;
}

public Cliente obetenerCliente( String id){…}

public bool borrarElementos (){…}

public bool grabarElementos (){…}
```

Detalle de utilización de una clase Singleton anterior en el proyecto

```
ContenedorClientesSingleton contenedorCli= ContenedorClientesSingleton.getInstancia();
```

Caso de Uso : Grabar Elementos

Este caso conllevará en nuestro caso la aplicación del patrón **ADO (Active Domain Object) visto en clase. Para ver el detalle de implementación del patrón se muestra una simplificación de clase Customer (del libro Data Access Patterns: Database Interactions in Object-Oriented Applications) que el alumno deberá adaptar al proyecto.**

Nota :En negrita se muestra las características relevantes respecto al patrón

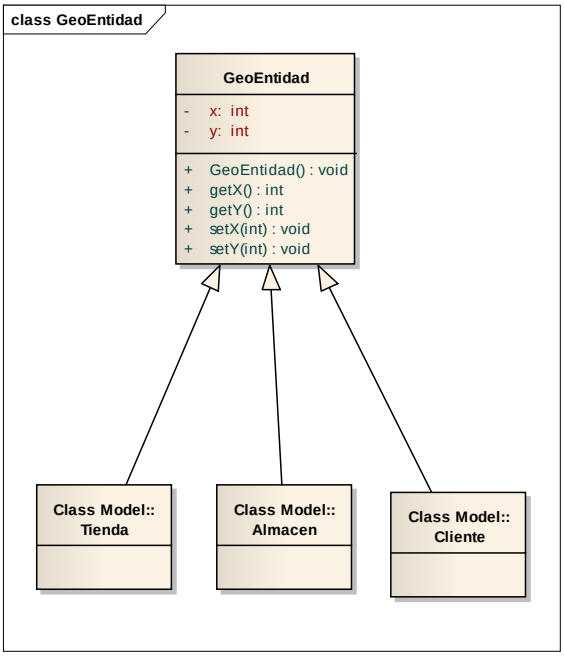
```
public class Customer {

    // SQL statements for data access.

    private static final String INSERT
        = "INSERT INTO CUSTOMERS "
        + "(LAST, FIRST, ADDRESS, CITY, STATE, COUNTRY, "
        + "ZIP, CUSTID) VALUES(?, ?, ?, ?, ?, ?, ?, ?)";

    private static final String UPDATE
        = "UPDATE CUSTOMERS SET LAST = ?, FIRST = ?, "
        + "ADDRESS = ?, CITY = ?, STATE = ?, COUNTRY = ?, "
        + "ZIP = ? WHERE CUSTID = ?";

    // The customer's data attributes.
    private int id;
    private String name;
    private Address address;

    // Indicates whether the data in the active domain
    // object matches that in the database.
    private boolean saved = false;

    /**
    Constructs a Customer object.  This constructor is
    intended for application code that adds new customers
    to the database.  It leaves the saved flag false to
    indicate that the data is not stored in the database
    yet.
    */
    public Customer(int id, String name, Address address) {
        this.id = id;
        this.name = name;
        this.address = address;
    }

    Constructs a Customer object.  This constructor is
    called only by CustomerList as it populates its
    contents.
    */
    public Customer(int id,
                    String lastName,
                    String firstName,
                    String address,
                    String city,
                    String state,
                    String country,
                    String zip) {

        initialize(id, lastName, firstName, address,
            city, state, country, zip);
    }

    /**
    Initializes the contents of this object based on
    physical data.  This is the mapping of data from
    its relational form to its domain object form.
    */
    private void initialize(int id,
```

```
                          String lastName,
                          String firstName,
                          String address,
                          String city,
                          String state,
                          String country,
                          String zip) {

        // Combine the first and last names, since
        // that is how the application
        // needs them.
        StringBuffer buffer = new StringBuffer();
        buffer.append(lastName);
        buffer.append(", ");
        buffer.append(firstName);
        name = buffer.toString();

        // Combine the city and state, since
        // that is how the application
        // needs them.
        buffer = new StringBuffer();
        buffer.append(city);
        buffer.append(", ");
        buffer.append(state);
        String cityState = buffer.toString();

        // Initialize the address.
        this.address = new Address();
        this.address.setAddress1(address);
        this.address.setCityState(cityState);
        this.address.setCountry(country);
        this.address.setPostalCode(zip);

        // Set this to true, since this information
        // was read from the database.
        saved = true;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Address getAddress() {
        return address;
    }

    public void setAddress(Address address) {
        this.address = address;
    }

    /**
    Saves the customer information to the database.
    This is the mapping of data from its domain object
    form to its relational form.  It uses the saved flag
    to determine whether to insert new data or update
    existing data.
    */
    public void save() throws DataException
    {
        // Split the name into first and last, since
        // the CUSTOMERS table stores these items in separate
        // columns.
        String first;
        String last;
        int comma = name.indexOf(',');
        if (comma >= 0) {
            first = name.substring(comma + 1).trim();
            last = name.substring(0, comma).trim();
        }
        else {
            first = "";
            last = name;
        }
```

```
        // Split the city and state, since the CUSTOMERS table
        // stores these items in separate columns.
        String cityState = address.getCityState();
        String city = "";
        String state = "";
        if (cityState != null) {
            comma = cityState.indexOf(',');
            if (comma >= 0) {
                city = cityState.substring(0, comma).trim();
                state = cityState.substring(comma).trim();
            }
            else {
                city = cityState;
            }
        }

        try {
            Connection connection
                = ConnectionFactory.getConnection();

            // If the customer information came from
            // the database, then issue an update.
            if (saved) {
                PreparedStatement statement
                    = connection.prepareStatement(UPDATE);
                statement.setString(1, last);
                statement.setString(2, first);
                statement.setString(3, address.getAddress1());
                statement.setString(4, city);
                statement.setString(5, state);
                statement.setString(6, address.getCountry());
                statement.setString(7, address.getPostalCode());
                statement.setInt(8, id);
                statement.executeUpdate();
                statement.close();
            }

            // If the customer information did not
            // come from the database, then issue an insert.
            else {
                PreparedStatement statement
                    = connection.prepareStatement(INSERT);
                statement.setString(1, last);
                statement.setString(2, first);
                statement.setString(3, address.getAddress1());
                statement.setString(4, city);
                statement.setString(5, state);
                statement.setString(6, address.getCountry());
                statement.setString(7, address.getPostalCode());
                statement.setInt(8, id);
                statement.executeUpdate();
                statement.close();

                // Indicate that the information now exists
                // in the database.
                saved = true;
            }

            connection.close();
        }
        catch(SQLException e) {
            throw new DataException("Unable to save customer "
                + id, e);
        }
    }
}
```

Caso de Uso : Georeferenciar

Detalle **de la extensión de las entidades Tienda, Almacén y Cliente** para añadirle características de Georeferenciacion.

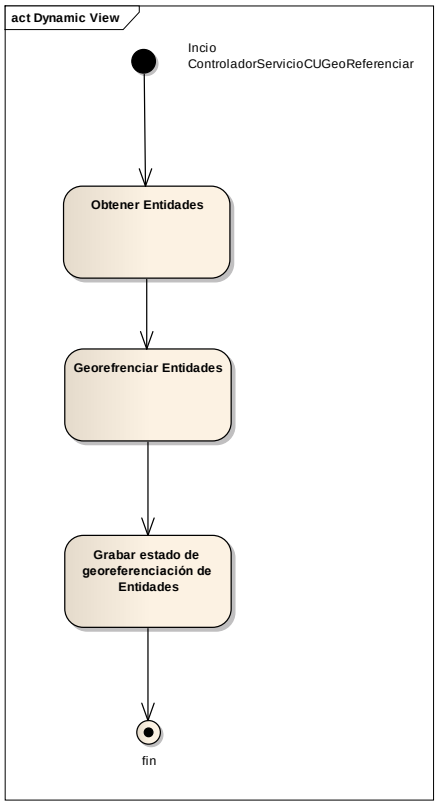


En el diagrama anterior **GeoEntidad es una clase base con un contructor por defecto**. Las propiedades x,y serán asignadas posteriormente en el caso de uso Georeferenciar

Nota : El alumno **deberá cambiar el constructor de cada subclase**

Caso de Uso : Georeferenciar

Detalle del diagrama de actividades para la clase **ControladorServicioCUGeoreferenciar**

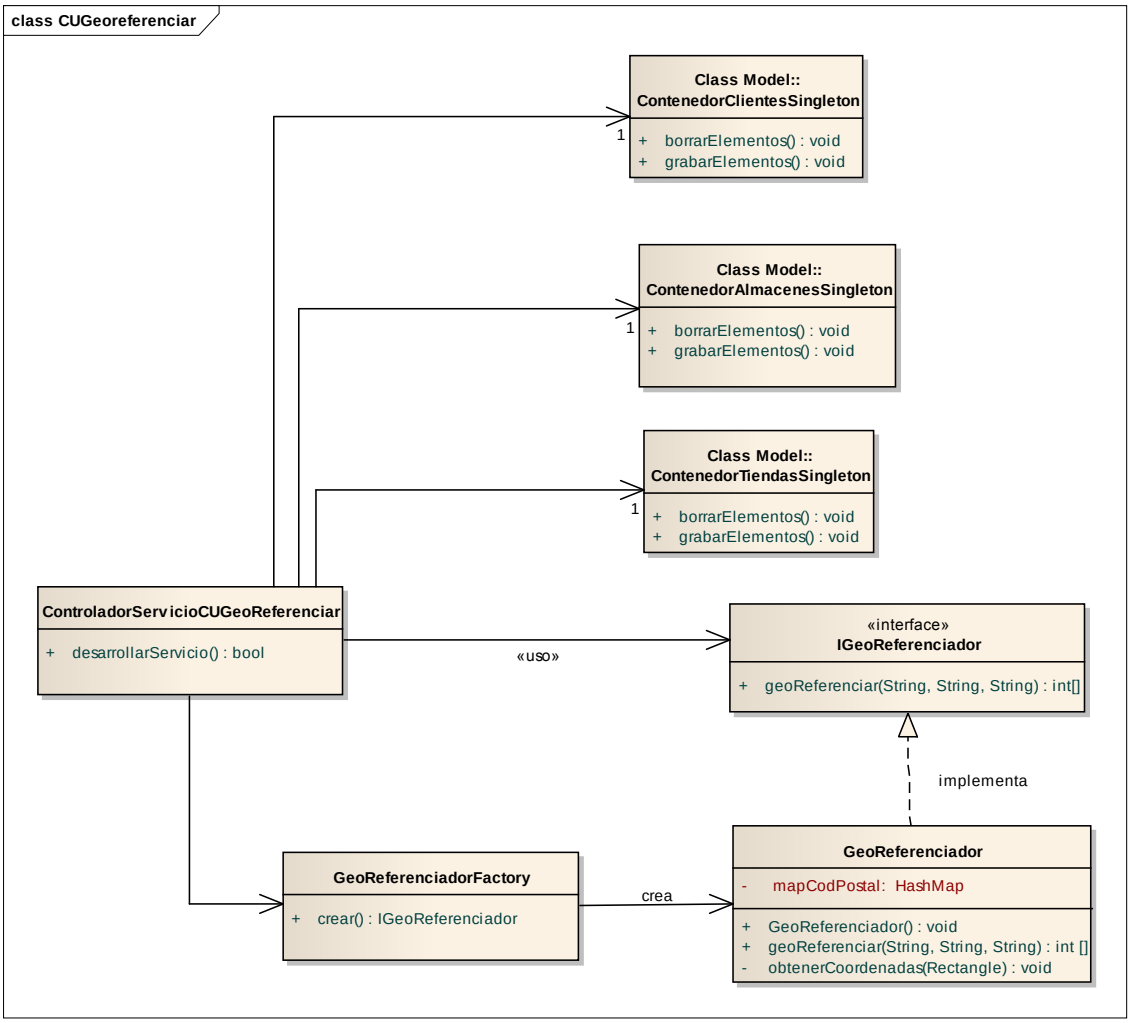


Para la actividad de Georeferenciación el alumno dispondrá de la librería **GeoRefrenciación Lib**

Caso de Uso : Georeferenciar

Este caso de uso contempla la asignación de un par de coordenadas a cada entidad con propiedades de localización espacial (ver jerarquía anterior)

Detalle de diagrama de clases



Donde

```
geoReferenciar(String cod_postal,String dirección,String poblacion):int[]
```

es el método de la interfaz **IGeoRefrenciador** que permite obtener la coordenadas geográficas de una Entidad dadas sus características de ubicación (cod_postal,dirección y población)

Este caso de uso considera el **patrón Simple Factory** visto en clase, para desacoplar el controlador de Servicio respecto del componente de Georeferenciación