Aquí va la explicación de como funciona el proyecto paquete por paquete

El programa:

Al ejecutar la aplicación te aparecerá arriba dos pestañas: Clientes y Vehículos. Clientes te permite consultar y modificar la información de los clientes, incluyendo las rentas de cada uno, asi como agregar o borrar clientes,para agregar un cliente presiona nuevo, introduce la información necesaria y presiona guardar,al presionar guardar se bloqueara el campo cedula para evitar su modificación.

Para editar los datos de un cliente búscalo en la lista a la izquierda y haz doble clic sobre el Realiza las modificaciones y presiona guardar.

Para eliminar un cliente búscalo en la lista a la izquierda , haz doble clic sobre el y luego presiona eliminar;

Para agregar un registro de renta al cliente actual presiona el botón “+” sobre la lista de rentas, aparecerá un cuadro de dialogo para introducir los datos, luego presiona aceptar.

Para eliminar un registro de renta selecciónalo de la lista y presiona el botón “-“.

La segunda pestaña permite editar la información de los vehículos disponibles

Los vehículos se agregan y eliminan de la misma forma que los clientes.

Para editar la información de un vehiculo búscalo en la lista a la izquierda, haz clic en el, realiza los cambios necesarios y presiona el botón “>>” al terminar de editar presiona guardar, esto guardara todos los cambios realizados.

Funcionamiento:

El paquete ObjectSerializator:

Este paquete contiene un conjunto de clases que permiten guardar y restaurar objetos a un archivo; un proceso llamado serializacion. Es un paquete de propósito general y puede ser reutilizado en otros proyectos.

Para que una clase pueda ser guardada y restaurada por este sistema debe cumplir que:

-sea public, las clases protected o de acceso por defecto no se pueden serializar

-implementar la interfaz Saveable. Esta define dos métodos: load y print, que permiten leer y escribir los datos de un objeto respectivamente

-poseer al menos un constructor sin parámetros.

Aquí un ejemplo:

Import ObjectSerializator.\*;

/\*\*

\*una clase que representa un punto en coordenadas rectangulares (x;y)

\*/

public class Punto implements Saveable{

private int x,y;

public Punto(){

}

public Punto(int x,int y){

this.x=x;

this.y=y;

}

public int getX(){

Return x;

}

public int getY(){

Return y;

}

public int setX(int x){

This.x=x;

}

public int setY(int y){

This.y=y;

}

@Override

public void print(DataWriter writer) throws WriteException {

writer.writeInt(x);

writer.writeInt(y);

}

@Override

public void load(DataReader reader) throws ReadException {

x=reader.readInt();

y=reader.readInt();

}

}

Del guardado y restauración se encargan las Interfaces DataReader y DataWriter respectivamente, mediante sus implementaciones TextDataReader y TextDataWriter que trabajan con archivos de texto formateados a XML

Cada clase posee métodos para leer y los tipos int, double, boolean, String y objetos Saveable

Para el caso del punto el guardado seria:

Punto p = new Punto (22 , 15);

DataWriter write=new TextDataWriter (new File (“el punto.xml”));

write.writeObject(p);

write.close();

y para leerlo:

DataReader read=new TextDataReader(new File(“el punto.xml”));

Punto p = (Punto)read.readObject();

read.close();

este sistema se usa para guardar la información de los vehículos

El paquete StringDB :

Este paquete permite trabajar con Archivos de registros de texto, sirviendo como bases de datos para guardar información organizada en registros.

Este sistema usa una técnica bastante simple llamada archivos de acceso aleatorio con registros de longitud fija, auxiliándose de la clase java.io.RandomAccessFile que permite moverse por un archivo de manera similar a un arreglo de bytes y además leer los tipos primitivos de java. Simplemente al abrir el archivo se suministra la longitud de los registros y de esa forma la posición de cada registro se puede calcular. Por ejemplo si cada registro ocupa 500 bytes el registro 1 está en la posición 0, el dos en la quinientos, el tres en la 1000, etc.

La clase DBFormat:

Esta clase permite definir el formato de un archivo de registros. Puede ser leída de un archivo de definición de formato XML (creado por ObjectSerializator) o definirse mediante su constructor, este requiere 4 parámetros

1. Un String : el nombre del formato es para documentación y no afecta el funcionamiento de la clase
2. Un arreglo de String : los títulos de los campos de los registros
3. Un arreglo de int : las longitudes(en caracteres) de los campos, debe ser de la misma longitud que el anterior
4. Un int : el índice de los arreglos anteriores que representa al campo clave

Una vez creado puede consultarse su información o usarse para crear un DBFile.

La clase DBFile:

Esta clase representa un archivo de registros de acceso aleatorio; para crear uno debe pasarse al constructor un DBFormat con el formato del archivo y un String o File con la ruta de este.

Una vez creado está listo para consultarse, si al crearlo el archivo no existe se creara uno vacío.

Para saber el numero de registros del archivo llame al método getRegisterCount();

Para moverse a un registro especifico llame al método seek () pasándole el índice (la posición, como en un arreglo) del registro a acceder recuerda comenzar desde cero.

Para agregar un registro llame al método addRegister(); esto creara un registro en blanco al final del archivo y lo retornara para su consulta o edición, este método desplaza el apuntador de seek a este registro.

Para eliminar un registro llame al método deleteRegister() pasándole el índice del registro a eliminar. La operación de eliminar causara un reordenamiento del archivo.

Para obtener una lista de los campos clave del archivo llame a getKeys() o getKeyFields(), el primero retorna un arreglo de Strings con las claves y el segundo retorna un arreglo de objetos KeyField, cada uno contiene la clave y el índice de su registro

Para acceder a un registro llama a seek para posicionarte en el y obtenlo con getRegister().

La clase Register:

Esta clase representa a un registro de un DBFile y permite editar su contenido.

Para obtener los campos del registro llama al método getFields().

Para obtener un campo especifico ,dado su posición o su titulo llama a getField().

Para cambiar los campos del registro llama a setFields() con un arreglo de String conteniendo los datos

Para cambiar un único campo del registro,dado su posición o su titulo llama a setField().

Los Eventos:

La Interfaz DBChangue Listener permite atrapar los eventos relacionados con la modificación de un registro. Contiene un único método : DBChangue( int changueType,Register target)

El primer parámetro representa el tipo de cambio puede ser:

ADD\_CHANGUE : adición de un registro

DELETE\_CHANGUE : borrado de un registro

UPDATE\_CHANGUE : modificación de un registro

KEY\_CHANGUE : modificación de la clave de un registro

El segundo es el registro afectado.

Para agregar un oyente a un DBFile solo implementa la interfaz ( puedes usar funciones lambda) y luego pásala al método addDBChangueListener() de la clase DBFile.

En el programa se usan para saber cuándo ha sido modificado el archivo de clientes para actualizar la lista de cedulas.

El registro de clientes:

La clase principal contiene el código encargado de gestionar los registros de clientes.

Para ello usa un DBFile con registros de 5 campos:

1. Id ,de longitud 20, para almacenar la cedula, es la clave
2. Nombre , de longitud 150 para el nombre y apellidos
3. Fecha de nacimiento ,de longitud 12, para la fecha de nacimiento
4. Email, de longitud 50 para el correo
5. Categoría, de longitud 1, guarda un carácter para identificar la categoría: z para zafiro, g para oro, s para plata y b para bronce.

El DBFormat requerido se carga del archivo “format.xml” ( ocurrirá un error si este no existe así que no lo dejes ni borres)

Para facilitar las pruebas la clase DBCreator contiene un método main, ejecútalo y creara un archivo con 100 registros generados aleatoriamente.

Ademas usa otro DBFile que almacena los registros de alquiler

Sus registros contienen los campos

1. Id, la clave, es generada por el programa combinando los otros campos, solo se usa para garantizar que cada clave sea única.
2. Cedula, longitud 20 , la cedula del cliente , husada para identificar el cliente que hizo la renta
3. Matricula, de longitud 10, la matricula del vehículo rentado.
4. Fecha, longitud 12 , la fecha de realización de la renta.
5. Días de renta, de longitud 3 , los días que se rentó el vehículo.

Al abrir un registro de cliente el programa busca en el archivo de rentas todos los registros que coincida la cedula con la de este y los muestra en la lista correspondiente, acción que se inicia al hacer doble clic en la lista. Para ello se auxilia de la clase RentalModel, una subclase de DefaultListModel, la clase husada para almacenar los valores de los JList, almacenando los datos de las rentas en un ArrayList para las consultas, y buscándolos en el archivo en caso de inserción o eliminación.

A diferencia de los anteriores el archivo de vehículos se guarda y carga todos los registros del archivo a la memoria y viceversa, almacenándolos en la clase SaveableListModel, también subclase de DefaultListModel, además la clase implementa Saveable, por lo que puede ser guardada y leída por TextDataReader. Que es lo que ocurre al presionar el botón guardar (escribir) o al iniciar la aplicación (leer).

La principal diferencia entre usar ObjectSerializator o StringDB es que la primera carga en memoria todo el archivo, lo que limita el tamaño que este puede alcanzar; por ello se puede emplear para el registro de vehículos; porque se sabe que no rebasara la capacidad de la memoria. Mientras que StringDB solo carga en memoria el registro actual, permitiendo un archivo arbitrariamente grande.