Licenciatura en Sistemas - Orientación a Objetos I – 2016

Prof. Titular: Lic. María Alejandra Vranić alejandravranic@gmail.com
facebook: alejandravranic

Prof. Ayudantes: APU Leandro Ríos leandro.rios . unla @ gmai l .com
facebook: Leandro Ríos

APU Gustavo Siciliano gussiciliano@gmail.com
facebook: Gus Shaggy Siciliano

Grupo en facebook: UNLa2016OO1

Trabajo Práctico n° 3 - Listas

Todos intuitivamente manejamos el concepto de lista en la vida cotidiana. Cuando nos vamos de campamento hacemos una lista con las cosas necesarias y a medida que nos vamos acordando agregamos al final de nuestra lista el objeto.

También ordenar nuestros objetos según un criterio como por ejemplo cocina, camping, ropa, perfumería etc y a medida que nos vamos acordando intercalar objetos.

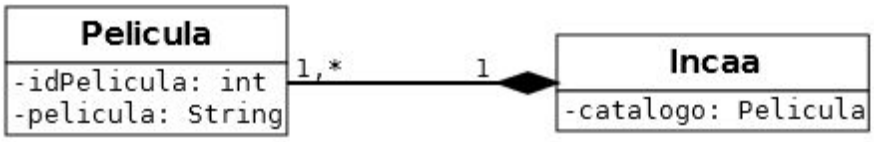
Las listas como los arreglos pueden contener en sus elementos variables primitivas o variables de objetos

Lista de String y formas de recorrerlas.

```
test;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Scanner;
public class TestListas {
        public static void main(String argv[]) {
       Scanner in =new Scanner( System.in);
       System.out.println("Vamos a llenar un ArrayList del tipo String, el contenido de
c/elemento es un String ");
       List<String> lista = new ArrayList<String>(); //inicializamos un objejto lista de
tipo String
       //entrada de las cadenas
        String elem="";
        while(!( (elem.equalsIgnoreCase("stop")))){
        System.out.println("Para finalizar tipear stop");
               elem= in.next();
        if (! (elem.equals("stop"))) lista.add(elem);
        }
       System.out.println("1) Impresión implementando foreach loop");
        for (String s : lista) {
        System.out.println(s);
        }
       System.out.println("2) impresión implementando for loop with index");
        for (int p = 0; p < lista.size(); p++) {
        System.out.println(lista.get(p));
        }
        System.out.println("3) impresión implementando Iterator<tipo>");
               for (Iterator<String> iter = lista.iterator(); iter.hasNext();) {
        System.out.println(iter.next());
        }
in.close();
}}
```

Proyecto Incaa

modelo:



Casos de uso:

+ agregarPelicula (String pelicula) : boolean

Si la película existe en la lista lanzar la excepción

+ traerPelicula (int idPelicula) : Pelicula

Si la película no existe devolver null

+ traerPelicula (String pelicula) : Pelicula

Si la película no existe devolver null

+ modificarPelicula (Pelicula pelicula) :

Modificar la película "seteando" los atributos del objeto (si no existe la película lanzar la excepción).

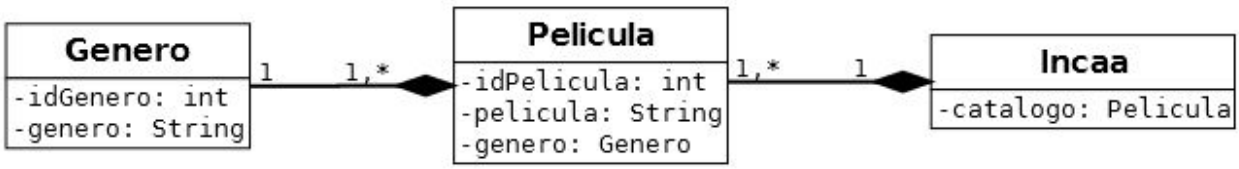
+ eliminarPelicula (Pelicula pelicula): boolean

Eliminar la película (si no existe la película lanzar la excepción)

Test: crear los test de para cada caso de uso con sus escenarios.

El paradigma de Objetos se basa en la reutilización de los Casos de Usos, cuando implementen cada comportamiento es bueno detenerse a pensar si es posible reutilizar otro Caso de Uso ya implementado, a fin de que la funcionalidad del sistema esté implementada en un solo método, conduciendo a un mejor diseño lo que facilita el mantenimiento.

modelo:



Crear la clase Genero como indica el diagrama de clases y en Pelicula agregar el atributo genero.

Casos de uso:

Sobrecargar el método traer película:

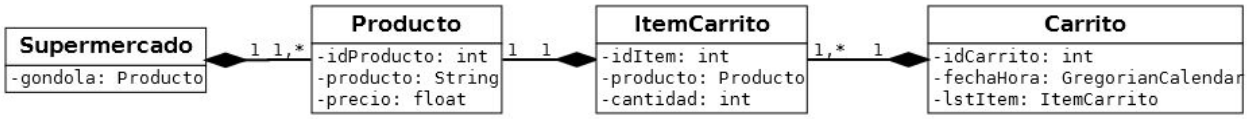
+ traerPelicula(Genero genero) : Pelicula

Test:

Traer las películas del catálogo de un género e imprimirlas.

Proyecto Supermercado

modelo:



+ agregarProducto(String producto, float precio) : boolean

Si el producto existe en la lista lanzar la excepción

+ traerProducto(int idProducto) : Producto

Si el producto no existe devolver null

+ modificarProducto(Producto producto) : boolean

Modificar el producto "seteando" los atributos del objeto (si no existe el objeto lanzar la excepción).

+ eliminarProducto(Produto producto) : boolean

+ agregarItem(Producto producto, int cantidad) : boolean

Cuando se agrega un producto al carrito si producto existe en algún item solo se incrementará la cantidad de lo contrario se agregará el item.

+ eliminarItem (Producto producto, int cantidad) : boolean

En el caso de eliminar un producto del carrito si la cantidad es la misma se eliminará el item, si es menor se decrecerá la cantidad y de lo contrario, si no existe el ítem que contenga el producto lanzará una excepción.

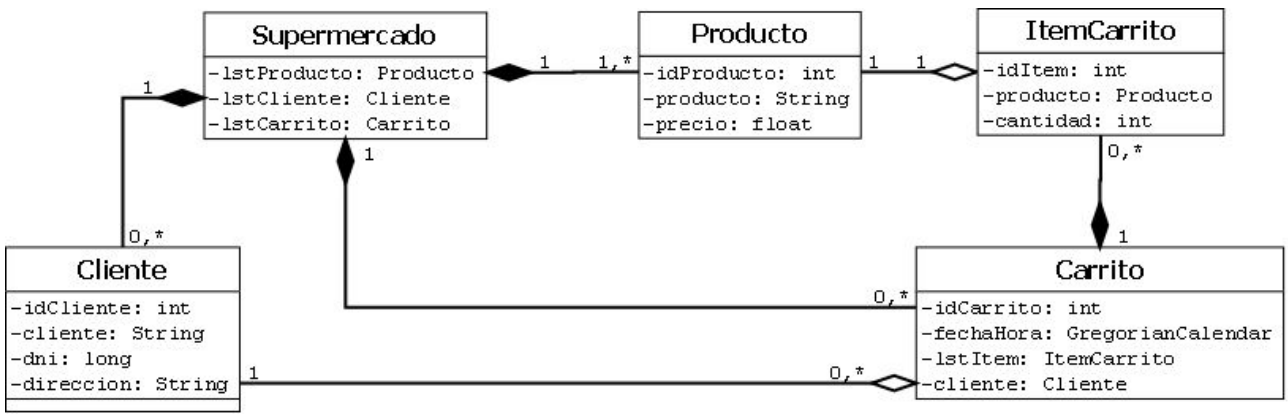
+ calcularSubTotal() : float

+ calcularTotal() : float

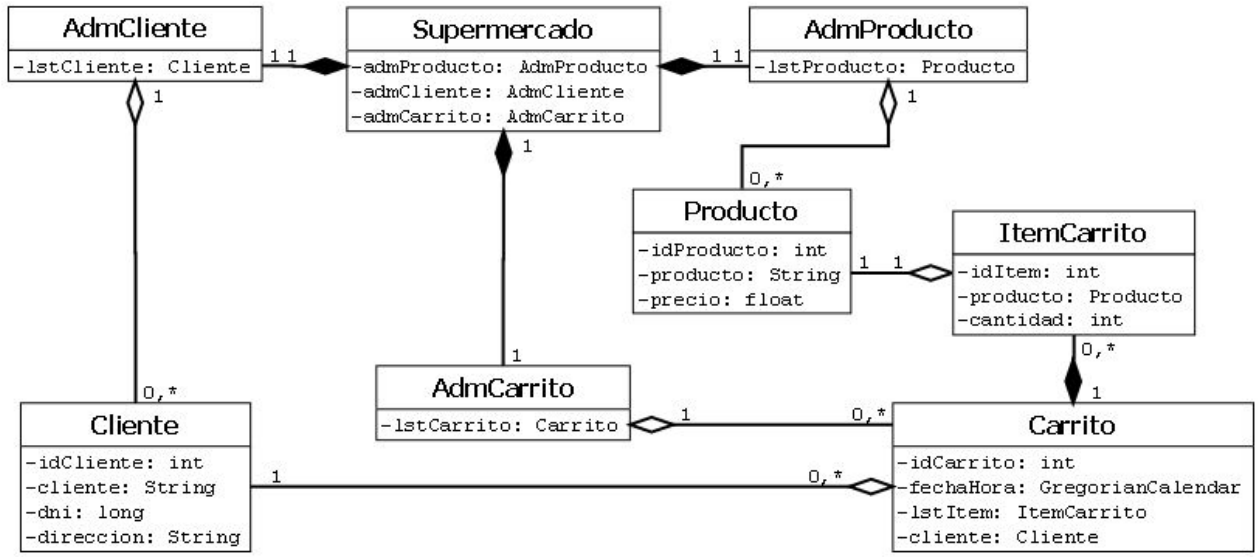
Test: crear los test de para cada caso de uso con sus escenarios.

Agregamos funcionalidad, les ofrecemos dos diseños del modelo; la idea es que logren implementar el segundo nivel.

Nivel 1:



Nivel 2:



Casos de uso a implementar en las clases correspondientes:

+ agregarProducto(String producto, float precio) : boolean

Si el producto existe en la lista lanzar la excepción

+ traerProducto(int idProducto) : Producto

Si el producto no existe devolver null

+ eliminarProducto(Produto producto) : boolean

+ agregarItem(Producto producto, int cantidad) : boolean

Cuando se agrega un producto al carrito si producto existe en algún item solo se incrementará la cantidad de lo contrario se agregará el item.

+ eliminarItem (Producto producto, int cantidad) : boolean

En el caso de eliminar un producto del carrito si la cantidad es la misma se eliminará el item, si es menor se decrecerá la cantidad y de lo contrario, si no existe el ítem que contenga el producto lanzará una excepción.

+ calcularSubTotal() : float

+ calcularTotal() : float

+agregarCliente(String cliente, long dni, String direccion): boolean

Si el cliente existe en la lista levantar una excepción.

+traerCliente(int idCliente): Cliente

Si el cliente no existe devolver null.

+eliminarCliente(Cliente cliente): boolean

Si el cliente no existe en la lista levantar una excepción.

+agregarCarrito(Carrito carrito): boolean

Si el carrito existe en la lista levantar una excepción.

+agregarCarrito(GregorianCalendar fechaHora, cliente Cliente): boolean

Si el carrito existe en la lista levantar una excepción.

+traerCarrito(int idCarrito): Carrito

Si el carrito no existe devolver null.

+eliminarCarrito(Carrito carrito): boolean

Si el carrito no existe en la lista levantar una excepción.

+calcularTotal(Cliente cliente): float

Si el cliente no existe levantar una excepción.

+calcularTotal(int dniCliente): float

Si el cliente no existe levantar una excepción.

+calcularTotal(GregorianCalendar fechaInicio, GregorianCalendar fechaFin): float

+calcularTotal(GregorianCalendar fecha): float

+calcularTotal(int mes, int anio): float

Si el mes es incorrecto (por ejemplo 23), levantar una excepción.

+calcularTotal(GregorianCalendar fechaInicio, GregorianCalendar fechaFin,
Cliente cliente): float

Si el cliente no existe levantar una excepción.

+calcularTotal(GregorianCalendar fecha, Cliente cliente): float

Si el cliente no existe levantar una excepción.

+calcularTotal(int mes, int anio, Cliente cliente): float

Si el cliente no existe y/o el mes es incorrecto levantar una excepción.

+calcularTotal(int mes, int anio, int dniCliente): float

Si el cliente no existe y/o el mes es incorrecto levantar una excepción.

Test: crear los test de para cada caso de uso con sus escenarios.

Bibliografía:

Ver Programa de Orientación a Objetos I (publicado en el grupo la 1° semana de cursada), hay libros nuevos y muy buenos con distintas miradas de los autores, es muy importante como futuros profesionales en ciencias de la computación adquirir saberes por investigación, por la obsolescencia del conocimiento.