

Equipo docente: Mg. María Alejandra Vranic

Esp. Lic. Gustavo Siciliano

Esp. Lic. Ezequiel Scordamaglia

Lic. Oscar Ruina

Listas

Manejo de Listas	2
Lista de String y formas de recorrerlas.	2
Práctica	4
5.A (Proyecto Incaa - NIVEL 1)	4
5.B (Proyecto Incaa - NIVEL 2)	5
5.C (Proyecto Supermercado - NIVEL 1)	5
5.D (Proyecto Supermercado - NIVEL 2)	6
5.E (Proyecto Supermercado - NIVEL 3)	7

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

Tanto las listas como los arreglos (denominados colecciones de manera general) pueden contener en sus elementos tipos de dato primitivos o de objetos.

Lista de String y formas de recorrerlas.

```
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 ");

        //inicializamos un objeto lista de tipo String
        List<String> lista = new ArrayList<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();
    }
}
```

Extra: Análisis de casteo

La cuestión ocurre al castear por ejemplo entre long y double, porque un long utiliza 63 bits para el valor y 1 para el signo, mientras que el double usa 52 para la mantisa y 1 para el signo. el resto es el exponente. Si el long que queremos castear ocupa más de 52 bits va a ocurrir pérdida de precisión, porque el double sólo puede almacenar 52 bits significativos. Puede (no vamos a tener error) almacenarse porque el double usa exponente, pero se perderán los dígitos menos significativos.

El int ocupa 32 bits en java, así que no debería haber problema al castearlo a double, ya que entra, el problema aparecería si lo queremos castear a float (usa 23 bits para la mantisa), por ejemplo.

Algunos sugieren verificar que la longitud en bits del número no supere la de la mantisa y levantar una excepción, si así ocurre. Si hacemos el casteo de un int a un float directamente, java pierde los dígitos menos significativos y hace la conversión sin decir nada.

De todos modos, la conversión como la estamos haciendo es ineficiente, ya que pasa el int a un string y después ese string a un double y nos evita el problema, ya que tampoco funciona:

```
int i = 2147483638; //(2^31-10)
float f = (float)i;

System.out.println(i);
System.out.println(f);
System.out.println((int)f);
System.out.println((int)Float.parseFloat(String.valueOf(i)));
```

```
Esto devuelve:
2147483638
2.14748365E9
2147483647
2147483647
```

En ambos casos se pierde precisión, y la conversión a String y float es más cara.

REALIZAR PRÁCTICA: 5.A (Proyecto Incaa - NIVEL 1)

REALIZAR PRÁCTICA: 5.B (Proyecto Incaa - NIVEL 2)

REALIZAR PRÁCTICA: 5.C (Proyecto Supermercado - NIVEL 1)

REALIZAR PRÁCTICA: 5.D (Proyecto Supermercado - NIVEL 2)

REALIZAR PRÁCTICA: 5.E (Proyecto Supermercado - NIVEL 3)

Práctica

5.A (Proyecto Incaa - NIVEL 1)

Modelo:

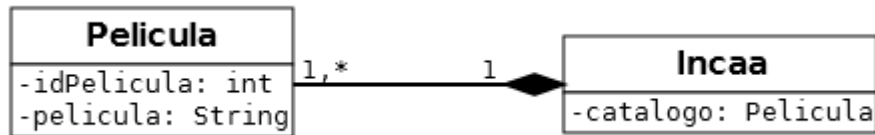


Diagrama de Clases

La clase Pelicula, con los atributos idPelicula tipo int, pelicula tipos String.

Por último la clase Incaa con el atributo catalogo del tipo List<Pelicula> donde se van a realizar Altas, Modificaciones y Bajas de objetos. Por lo tanto existe una relación de agregación entre Incaa y Pelicula de uno a muchos, la objeto Incaa va tener muchos objetos Pelicula en la lista.

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 partePelicula) : List<Pelicula> //todas las películas que contengan en su título el string partePelicula

+ modificarPelicula (int idPelicula, String pelicula) :

Modificar la película traerPelicula por id, si no existe la película lanzar la excepción, de lo contrario setPelicula con el parámetro pelicula .

+ eliminarPelicula (int idPelicula): boolean

Eliminar la película: traerPelicula por su id, si no existe la película lanzar la excepción, de lo contrario eliminar el elemento de la lista (remove)

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 Uso. En el momento de implementar cada comportamiento, es bueno detenerse a pensar si es posible reutilizar otro Caso de Uso ya implementado; con el fin de que la funcionalidad del sistema exista en un solo método, lo que conduce a un mejor diseño y facilita el mantenimiento.

5.B (Proyecto Incaa - NIVEL 2)

Modelo:

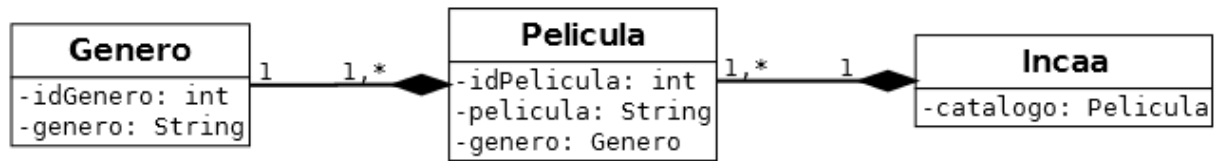


Diagrama de Clases

La clase Genero, con los atributos idGenero tipo int y genero tipo String.

La clase Pelicula, con los atributos idPelicula tipo int, pelicula tipos String y genero tipo Genero, por lo tanto existe una relación de composición entre las clases Pelicula y Genero, con cardinalidad de muchos a uno o sea que puede haber muchas películas de un género (no consideramos que una película pueda pertenecer a más un género para simplificar el problema).

Por último la clase Incaa con el atributo catálogo del tipo List<Pelicula> donde se van a realizar Altas, Modificaciones y Bajas de objetos.

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

Casos de uso:

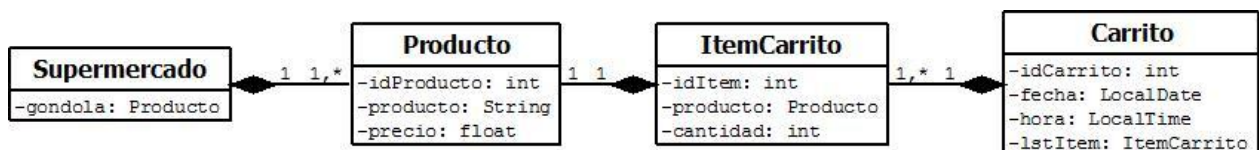
Sobrecargar el método traer película:

+ traerPelicula(Genero genero) : Pelicula

Test: Traer las películas del catálogo de un género e imprimirlas.

5.C (Proyecto Supermercado - NIVEL 1)

Modelo:



1) + agregarProducto(String producto, float precio) : boolean

Si el producto existe en la lista lanzar la excepción

2) + traerProducto(int idProducto) : Producto

Si el producto no existe devolver null

3) + modificarProducto(int idProducto, String producto, double precio) : boolean

Modificar el producto: traer producto por id, si no existe el objeto levantar la excepción, de lo contrario volver setear producto y precio.

4) + eliminarProducto(int idProducto) : boolean

Eliminar producto: traer producto por id, si no existe el objeto o existe en algún carrito levantar la excepción, de lo contrario eliminar el objeto (remove).

5) + 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.

6) + 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.

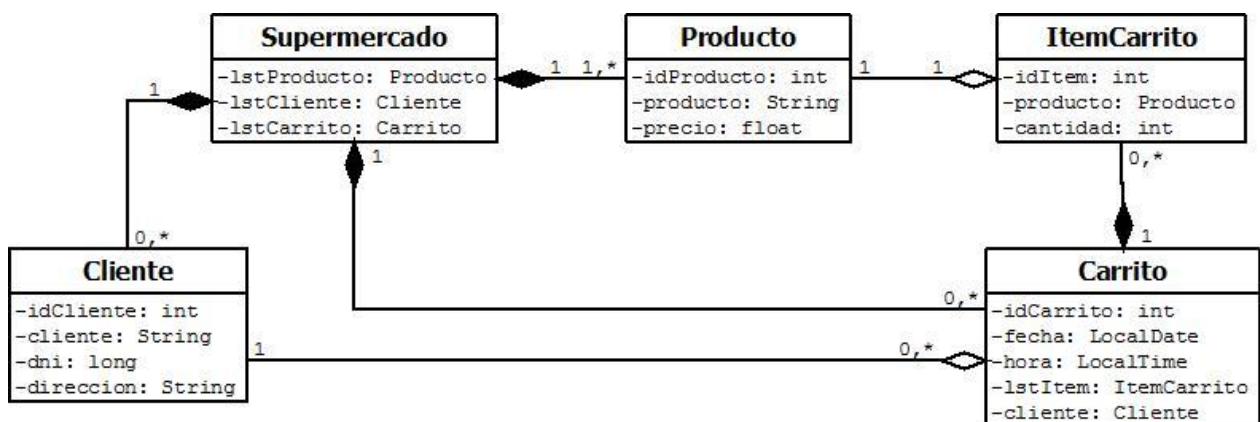
7) + calcularSubTotal() : float

8) + calcularTotal() : float

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

5.D (Proyecto Supermercado - NIVEL 2)

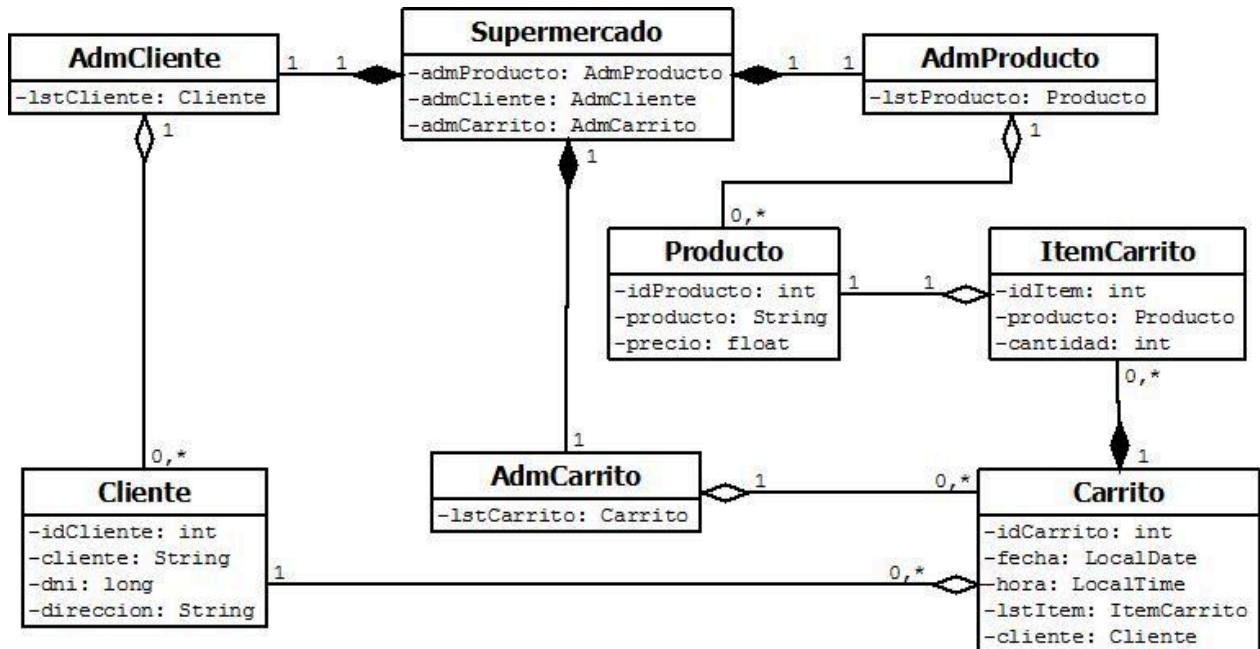
Modelo:



Terminar de implementar las clases y los casos de uso del diagrama.

5.E (Proyecto Supermercado - NIVEL 3)

Modelo:



Casos de uso a implementar en las clases correspondientes:

- 1) + agregarProducto(String producto, float precio) : boolean
- 2) + traerProducto(int idProducto) : Producto
- 3) + eliminarProducto(int idProducto) : boolean
- 4) + agregarItem(Producto producto, int cantidad) : boolean
- 5) + eliminarItem (Producto producto, int cantidad) : boolean
- 6) + calcularSubTotal() : float
- 7) + calcularTotal() : float
- 8) +agregarCliente(String cliente, long dni, String direccion): boolean

Si el cliente existe en la lista levantar una excepción.

- 9) +traerCliente(int idCliente): Cliente

Si el cliente no existe devolver null.

- 10) +eliminarCliente(int idCliente): boolean

Si el cliente no existe o tiene algún carrito se debe levantar una excepción.

- 11) +agregarCarrito(LocalDate fecha, LocalTime hora, cliente Cliente): boolean

Si el carrito existe en la lista levantar una excepción.

- 12) +traerCarrito(int idCarrito): Carrito

Si el carrito no existe devolver null.

- 13) +eliminarCarrito(int idCarrito): boolean

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

14) +calcularTotal(Cliente cliente): float

Si el cliente no existe levantar una excepción.

15) +calcularTotal(int dniCliente): float

Si el cliente no existe levantar una excepción.

16) +calcularTotal(LocalDate fechaInicio, LocalDate fechaFin): float

17) +calcularTotal(LocalDate fecha): float

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

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

19) +calcularTotal(LocalDate fechaInicio, LocalDate fechaFin,
Cliente cliente): float

Si el cliente no existe levantar una excepción.

20) +calcularTotal(LocalDate fecha, Cliente cliente): float

Si el cliente no existe levantar una excepción.

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

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

22) +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.