



Universidad
de Huelva

MFIS

Práctica 1



Autores:

- Ismael Da Palma Fernández
- Cristian Delgado Cruz

- Juan Jimenez Serrano
- Fernando García-Palomo Albarrán

Índice

Índice	1
1. Explicación del funcionamiento	2
1.1. Invariantes	2
1.2. Contratos	4
2. Diagrama de Clases	6
3. Comprobación de restricciones	7
3.1. Invariantes	7
3.2. Contratos	11
4. Script de Creación	15
5. Conclusión	16

1. Explicación del funcionamiento

1.1. Invariantes

Nuestra práctica consta de **44** invariantes, pero la mayoría son repetidas y muy triviales, como por ejemplo que las instancias no tengan ni los id ni los nombres en nulo (que sean obligatorios), o simplemente invariantes que se repiten porque es necesario, como que no pueda haber dos Vehículos, Personas, Productos, etc con la misma id, por lo que hemos seleccionado las siguientes **9** para realizar su explicación:

- 1) Si un comprador compra 50 o más productos, sólo podrá ser Mayorista.

context Comprador inv TipoMayo: self.compra->forAll(C | C.Cantidad >= 50 implies self.TipoCompra = TipoComprador::Mayorista)

- 2) Si dos compradores tienen distinto id implica que estos tengan distinto teléfono móvil.

context Comprador inv idDistintoC: Comprador.allInstances->forAll(c1, c2 | c1.idComprador <> c2.idComprador implies c1.Telefono <> c2.Telefono)

- 3) La línea de compra no puede superar en cantidad el stock del producto que se quiere comprar.

inv CompraReal: self.producto->forAll(P | self.Cantidad <= P.Stock)

- 4) Los productos de una nave deben ser del mismo tipo, es decir, no puede haber una nave con más de un tipo de producto.

context Nave inv ProductosN: self.productos->forAll(P | self.Productos = P.Tipo)

5) Un comprador mayorista sólo puede comprar un solo tipo de producto.

context Comprador inv CompradorMayorista: self.compra->forAll(C1, C2| self.TipoCompra = TipoComprador::Mayorista implies C1.TipoProductoCompra = C2.TipoProductoCompra)

6) Un comprador mayorista sólo puede realizar una compra al día.

context Comprado inv UnicaCompraDiaMayorista: self.compra->forAll(C1,C2 | self.TipoCompra = TipoComprador::Mayorista and C1 <> C2 implies C1.FechaCompra <> C2.FechaCompra)

7) Cada proveedor dispone de al menos un camión.

context Proveedor inv MinimoCamion: self.vehiculo->exists(v:Vehiculo | v.Tipo = TipoVehiculo::Camion)

8) Todas las líneas de compra deben de tener una cantidad de productos mayor que 0.

context LineadeCompra inv CompraNoVacía: LineadeCompra.allInstances->forAll(L | self.Cantidad > 0)

9) Todos los vehículos del proveedor sólo pueden transportar un mismo tipo de producto.

context Proveedor inv MismoTipoProducto: self.vehiculo->forAll(v1,v2 | v1.TipoProductoTransporta = v2.TipoProductoTransporta)

1.2. Contratos

Con respecto a los contratos ocurre lo mismo que con las invariantes, tenemos **13** operaciones, de las cuales algunas, son operaciones Getters, por lo cual no tenemos contratos definidos para todas ellas, Los métodos cuyos contratos caben destacar son los siguientes:

1) Método de Nave, EliminarProducto(int Cantidad): Elimina una cantidad del producto elegido.

context Nave :: eliminarProducto(idP : Integer, Cantidad : Integer)
pre HayProducto: self.productos->exists(P | P.idProducto = idP and P.Stock >= Cantidad)

2) Método de Nave, EliminarProductosMalos(): Elimina los productos todos aquellos que sean de calidad mala o inferior.

context Nave :: eliminarProductosMalos()
pre HayProductoMalo: self.productos->exists(P | P.Calidad = NivelCalidad::Mala or P.Calidad = NivelCalidad::MuyMala)

3) Método de Nave, OfertarProductos(): Reduce el precio de los productos, pero solo de aquellos que sean de calidad Normal.

context Nave :: ofertarProductos() **pre**
HayProductoNormal: self.productos->exists(P | P.Calidad = NivelCalidad::Normal)
post valeMenosahora: self.productos->forAll(P | P.PrecioBase < P.PrecioBase@pre)

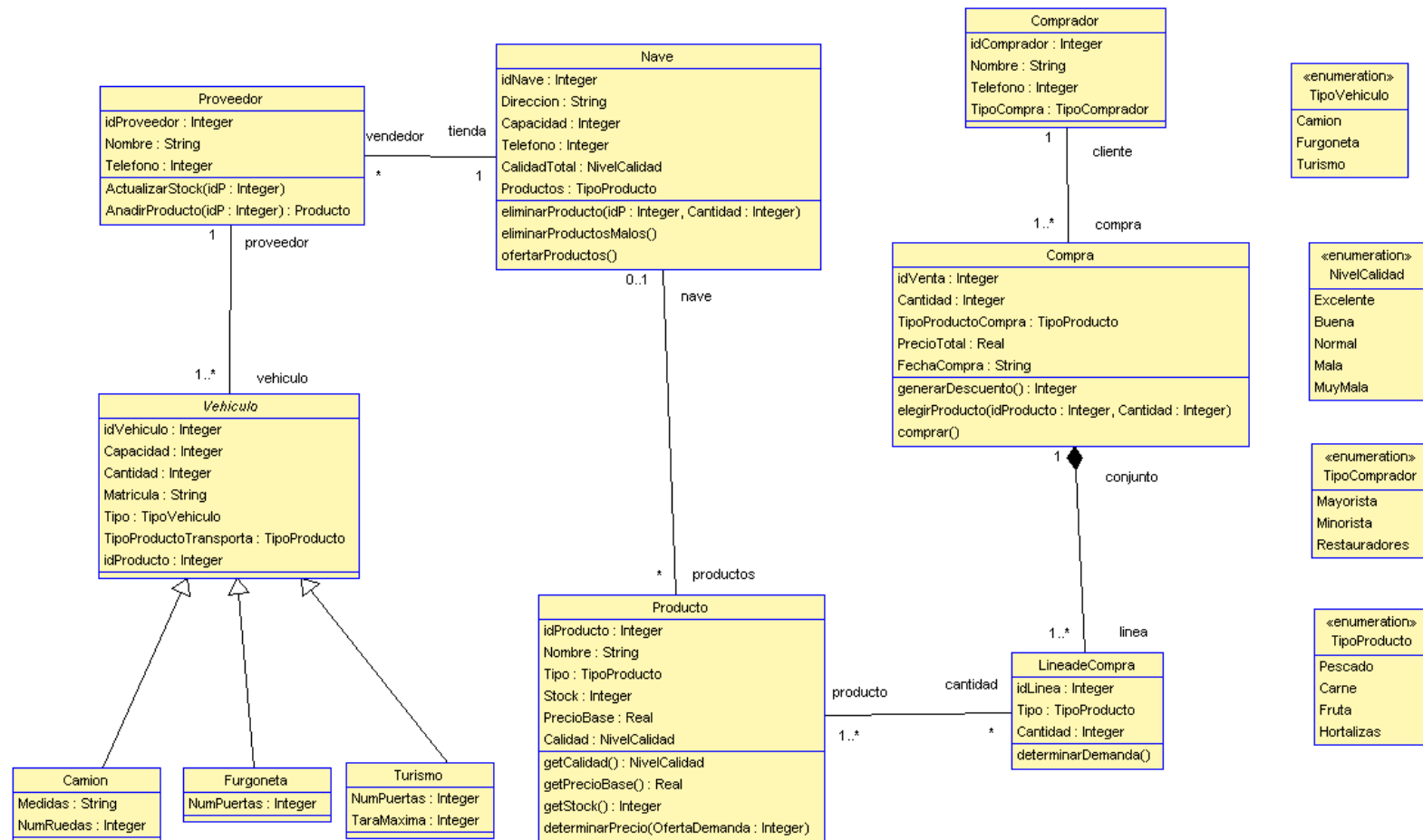
4) Método de *Compra*, **GenerarDescuento()**: si el precio de compra Total es mayor que 50€ o por otra parte la cantidad comprada supera los 25 productos, se aplicará un descuento variable, siendo un mínimo de 25%.

context *Compra* :: **generarDescuento()** : *Integer* **pre**
CompraSuficiente: *self.PrecioTotal* >= 50 or *self.Cantidad* >= 25 **post**
DescuentoMinimo: *result* <= *self.PrecioTotal* * 0.25

5) Método de *Proveedor*, **ActualizarStock()**: Al llegar un vehículo, este se vaciará y la cantidad del mismo se sumará a la cantidad de stock del producto, para ello debe existir un vehículo que tenga algún tipo de producto cuyo id exista en la nave, donde pretende descargarlo

context *Proveedor* :: **ActualizarStock(idP : Integer)** **pre**
VehiculoTransporta: *self.vehiculo->exists(V | V.Cantidad > 0 and V.idProducto = idP)*
pre ExisteEseProducto: *self.tienda.productos->exists(P | P.idProducto = idP)*
post MasProducto: *self.tienda.productos->forall(P| P.idProducto = idP implies P.Stock > P.Stock@pre)*

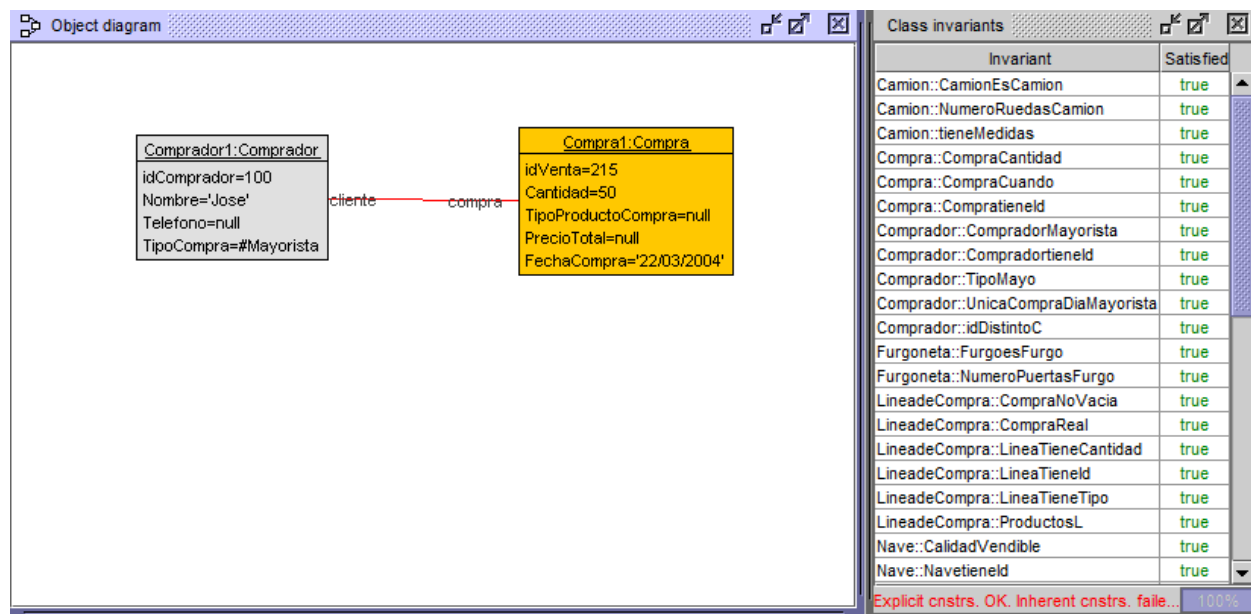
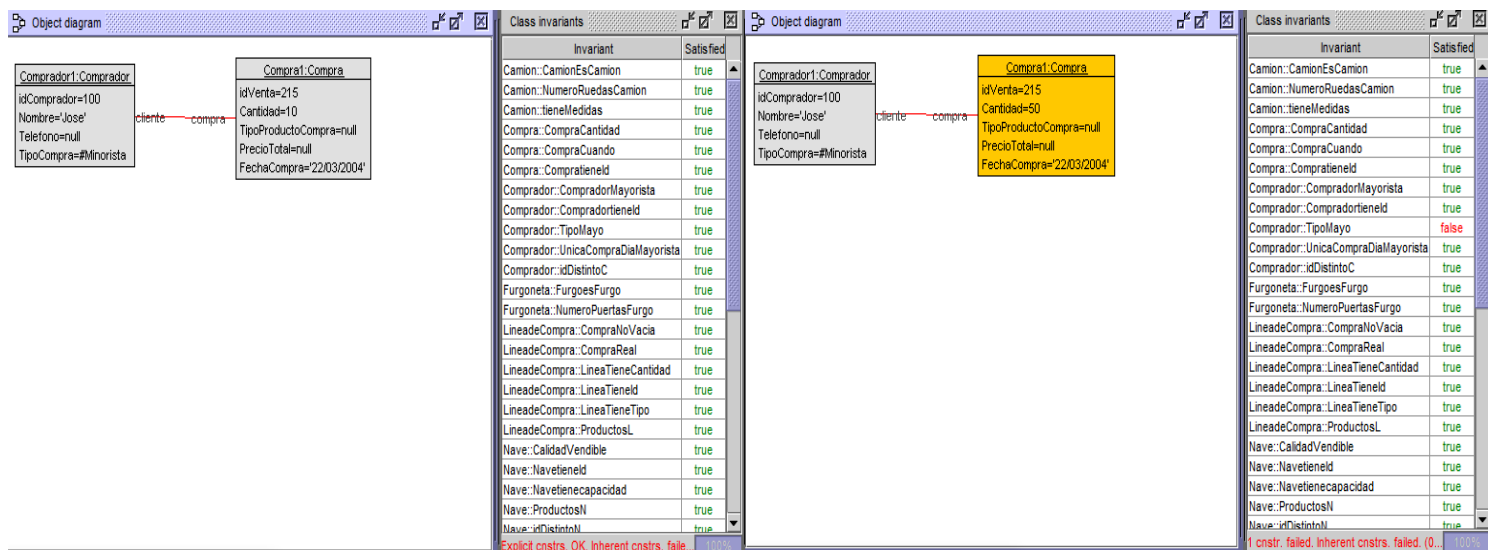
2. Diagrama de Clases



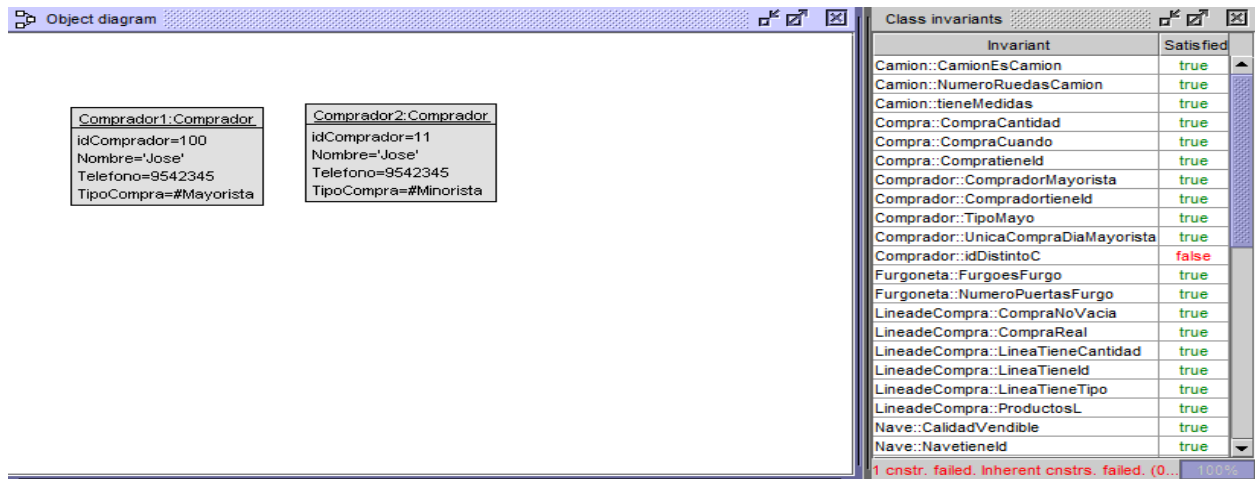
3. Comprobación de restricciones

3.1. Invariantes

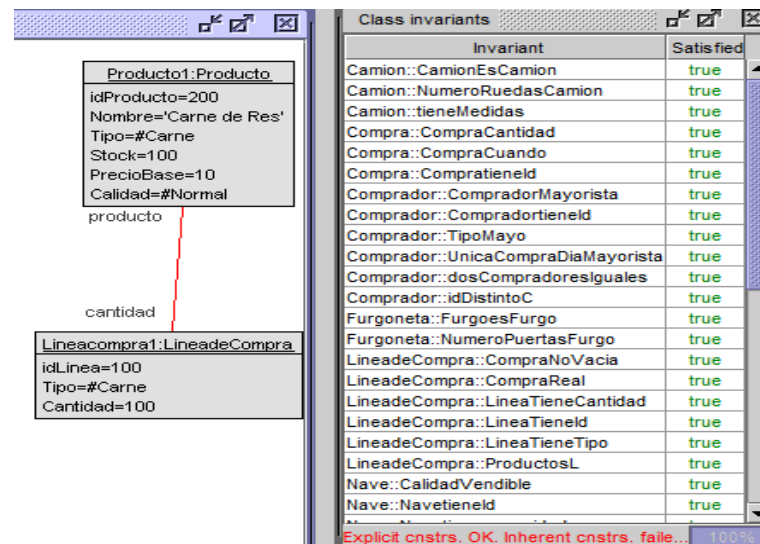
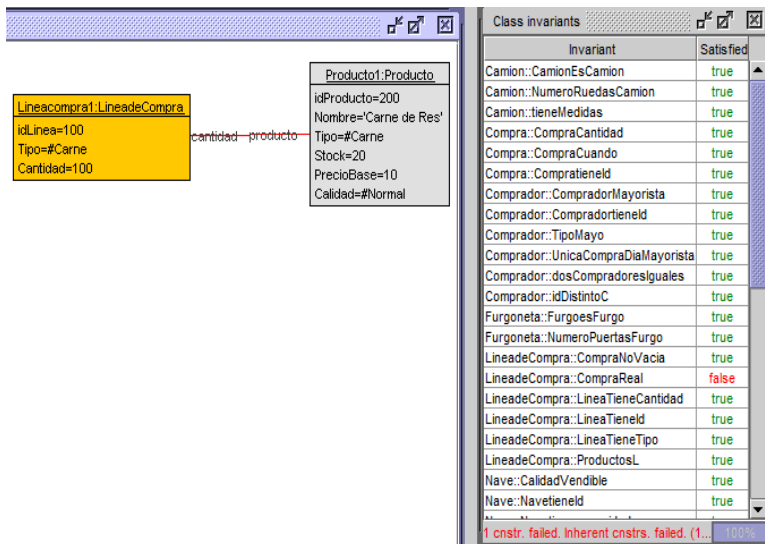
1. inv TipoMayo:



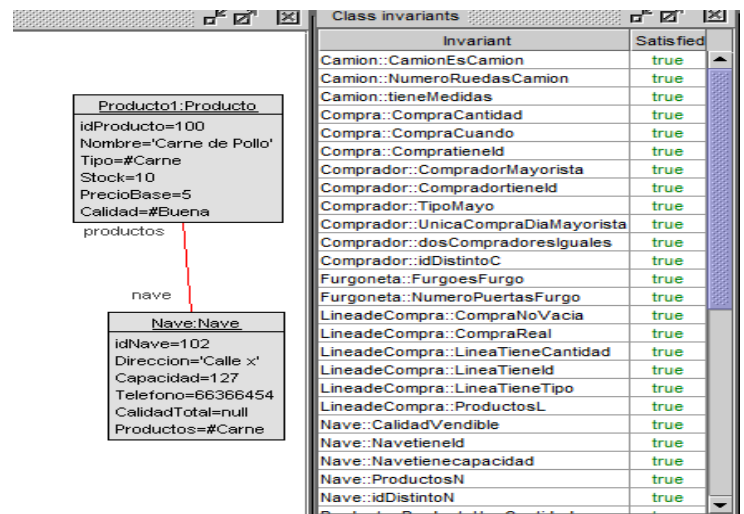
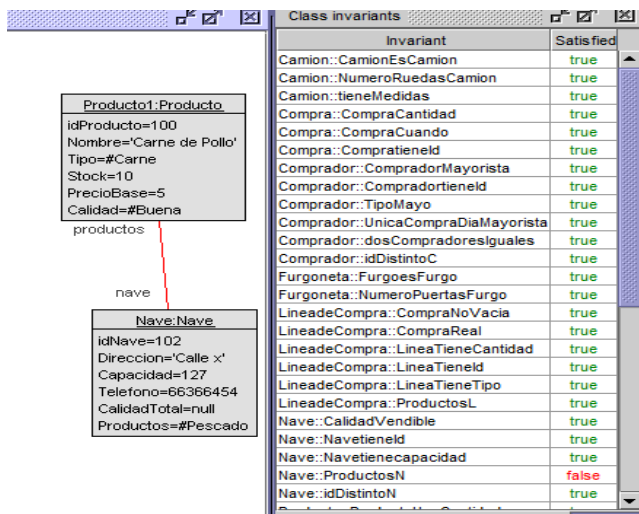
2. inv idDistintoC:



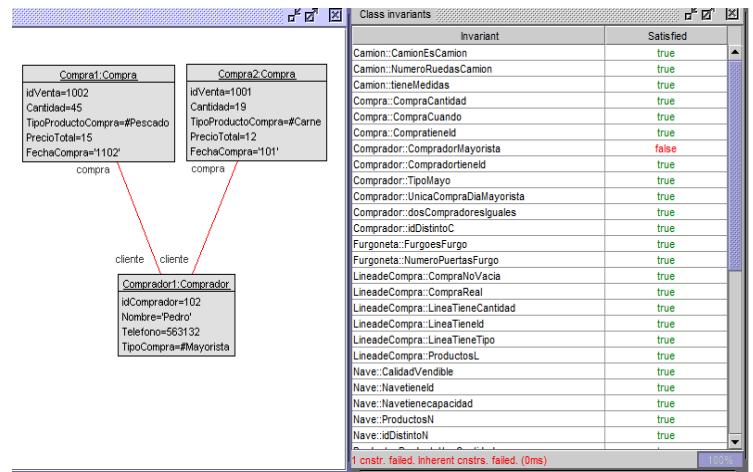
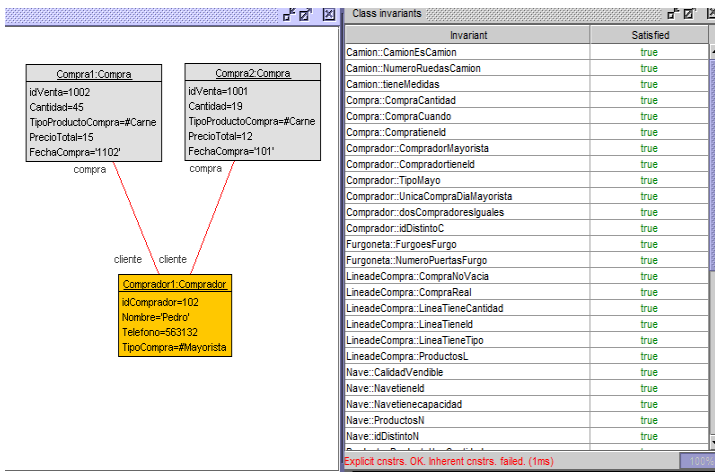
3. inv CompraReal:



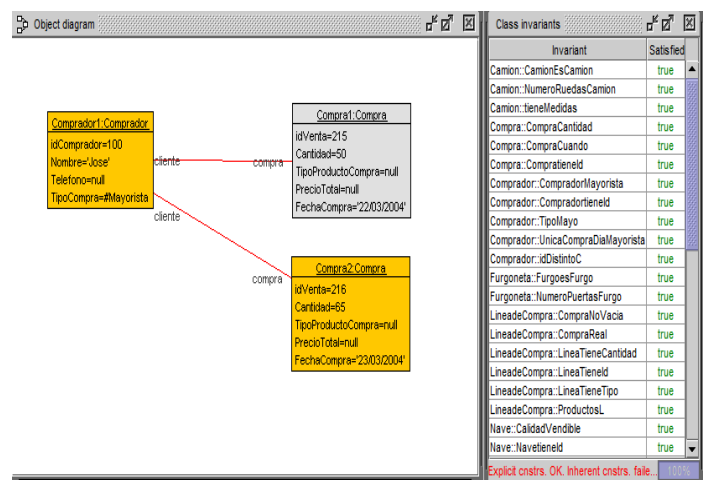
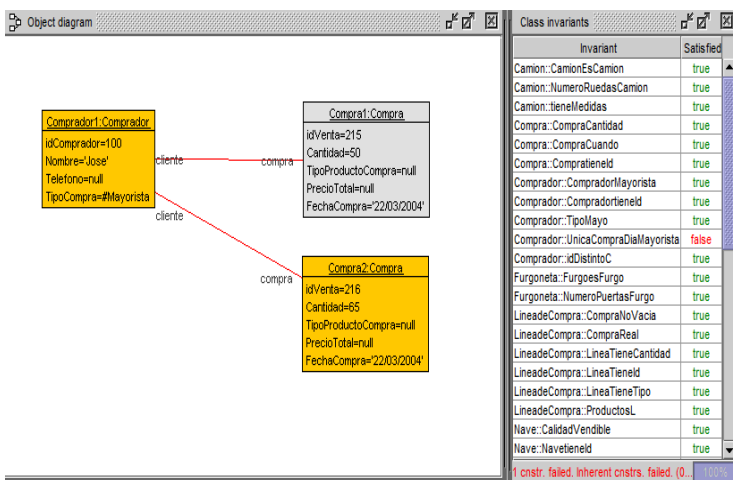
4. inv ProductosN:



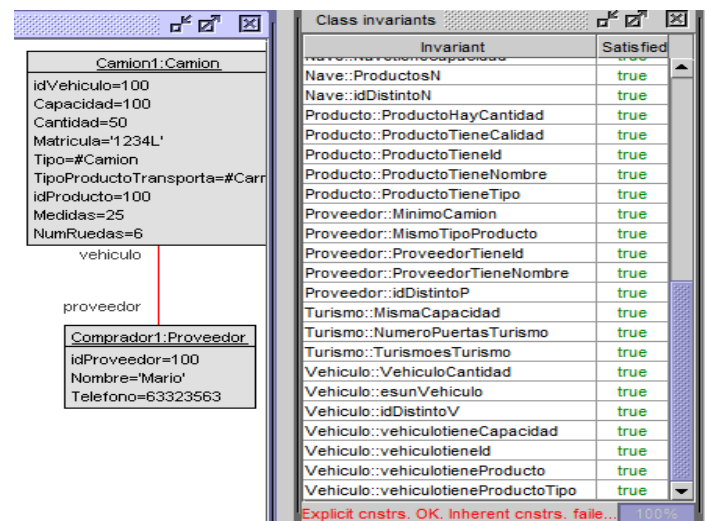
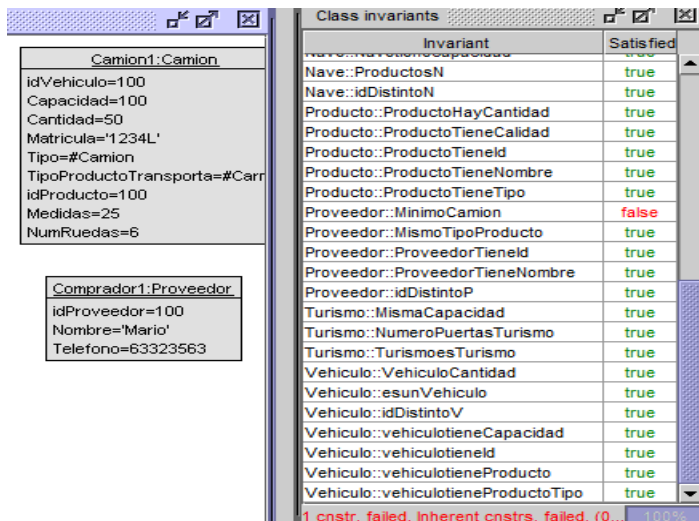
5. inv CompradorMayorista:



6. inv UnicaCompraDiaMayorista:



7. inv MinimoCamion



8. inv CompraNoVacía:

Class invariants	
Invariant	Satisfied
Camion::CamionEsCamion	true
Camion::NumeroRuedasCamion	true
Camion::tieneMedidas	true
Compra::CompraCantidad	true
Compra::CompraCuando	true
Compra::CompratieneId	true
Comprador::CompradorMayorista	true
Comprador::CompradortieneId	true
Comprador::TipoMayo	true
Comprador::UnicaCompraDiaMayorista	true
Comprador::dosCompradoresIguales	true
Comprador::idDistintoC	true
Furgoneta::FurgoesFurgo	true
Furgoneta::NumeroPuertasFurgo	true
LineaCompra::CompraNoVacía	false
LineaCompra::CompraReal	true
LineaCompra::LineaTieneCantidad	true
LineaCompra::LineaTieneId	true
LineaCompra::LineaTieneTipo	true
LineaCompra::ProductosL	true
Nave::CalidadVendible	true
Nave::NavetieneId	true
Nave::Navetienecapacidad	true
Nave::ProductosN	true
Nave::idDistintoN	true

1 cnstr. failed. Inherent cnstrs. failed. (1... 100%

LineaCompra1: LineaCompra
idLinea=1
Tipo=#Carne
Cantidad=0

Class invariants	
Invariant	Satisfied
Camion::CamionEsCamion	true
Camion::NumeroRuedasCamion	true
Camion::tieneMedidas	true
Compra::CompraCantidad	true
Compra::CompraCuando	true
Compra::CompratieneId	true
Comprador::CompradorMayorista	true
Comprador::CompradortieneId	true
Comprador::TipoMayo	true
Comprador::UnicaCompraDiaMayorista	true
Comprador::dosCompradoresIguales	true
Comprador::idDistintoC	true
Furgoneta::FurgoesFurgo	true
Furgoneta::NumeroPuertasFurgo	true
LineaCompra::CompraNoVacía	true
LineaCompra::CompraReal	true
LineaCompra::LineaTieneCantidad	true
LineaCompra::LineaTieneId	true
LineaCompra::LineaTieneTipo	true
LineaCompra::ProductosL	true
Nave::CalidadVendible	true
Nave::NavetieneId	true
Nave::Navetienecapacidad	true
Nave::ProductosN	true
Nave::idDistintoN	true

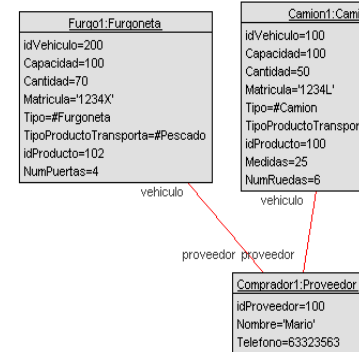
Explicit cnstrs. OK. Inherent cnstrs. failed. (1... 100%

LineaCompra1: LineaCompra
idLinea=1
Tipo=#Carne
Cantidad=5

9. inv MismoTipoProducto:

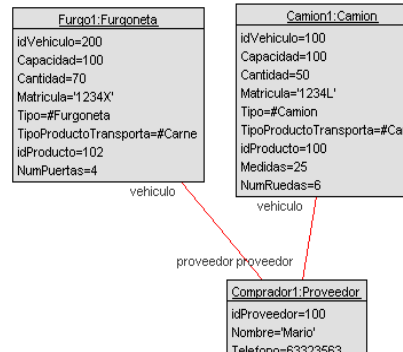
Class invariants	
Invariant	Satisfied
Nave::ProductosN	true
Nave::idDistintoN	true
Producto::ProductoHayCantidad	true
Producto::ProductoTieneCantidad	true
Producto::ProductoTieneId	true
Producto::ProductoTieneNombre	true
Producto::ProductoTieneTipo	true
Proveedor::MinimoCamion	true
Proveedor::MismoTipoProducto	false
Proveedor::ProveedorTieneId	true
Proveedor::ProveedorTieneNombre	true
Proveedor::idDistintoP	true
Turismo::MismaCapacidad	true
Turismo::NumeroPuertasTurismo	true
Turismo::TurismoesTurismo	true
Vehiculo::VehiculoCantidad	true
Vehiculo::esunVehiculo	true
Vehiculo::idDistintoV	true
Vehiculo::vehiculotieneCapacidad	true
Vehiculo::vehiculotieneId	true
Vehiculo::vehiculotieneProducto	true
Vehiculo::vehiculotieneProductoTipo	true

1 cnstr. failed. Inherent cnstrs. failed. (1... 100%



Class invariants	
Invariant	Satisfied
Nave::ProductosN	true
Nave::idDistintoN	true
Producto::ProductoHayCantidad	true
Producto::ProductoTieneCantidad	true
Producto::ProductoTieneId	true
Producto::ProductoTieneNombre	true
Producto::ProductoTieneTipo	true
Proveedor::MinimoCamion	true
Proveedor::MismoTipoProducto	true
Proveedor::ProveedorTieneId	true
Proveedor::ProveedorTieneNombre	true
Proveedor::idDistintoP	true
Turismo::MismaCapacidad	true
Turismo::NumeroPuertasTurismo	true
Turismo::TurismoesTurismo	true
Vehiculo::VehiculoCantidad	true
Vehiculo::esunVehiculo	true
Vehiculo::idDistintoV	true
Vehiculo::vehiculotieneCapacidad	true
Vehiculo::vehiculotieneId	true
Vehiculo::vehiculotieneProducto	true
Vehiculo::vehiculotieneProductoTipo	true

Explicit cnstrs. OK. Inherent cnstrs. failed. (1... 100%



3.2. Contratos

1. EliminarProducto(int idP , int Cantidad):

The screenshot displays a software development environment with three main components:

- Class Diagram:** Shows two classes: `Prod1:Producto` and `Nav1:Nave`. `Prod1:Producto` has attributes `idProducto=100`, `Nombre='x'`, `Tipo=#Carne`, `Stock=100`, `PrecioBase=100`, and `Calidad=#Buena`. `Nav1:Nave` has attributes `idNave=1002`, `Direccion='x'`, `Capacidad=10000`, `Telefono=111222333`, `CalidadTotal=#Buena`, and `Productos=#Carne`. A red line labeled `productos` connects `Prod1:Producto` to `Nav1:Nave`, and another red line labeled `nave` connects `Nav1:Nave` to `Prod1:Producto`.
- Class Invariants Table:** A table with two columns: `Invariant` and `Satisfied`. It lists 20 invariants, all of which are satisfied (true).

Invariant	Satisfied
Camion::CamionEsCamion	true
Camion::NumeroRuedasCamion	true
Camion::tieneMedidas	true
Compra::CompraCantidad	true
Compra::CompraCuando	true
Compra::CompratieneId	true
Comprador::CompradorMayorista	true
Comprador::CompradortieneId	true
Comprador::TipoMayo	true
Comprador::UnicaCompraDiaMayorista	true
Comprador::dosCompradoresIguales	true
Comprador::idDistintoC	true
Furgoneta::FurgoesFurgo	true
Furgoneta::NumeroPuertasFurgo	true
LineadeCompra::CompraNoVacua	true
LineadeCompra::CompraReal	true
LineadeCompra::LineaTieneCantidad	true
LineadeCompra::LineaTieneId	true
LineadeCompra::LineaTieneTipo	true
LineadeCompra::ProductosL	true
Nave::CalidadVendible	true
Nave::NavetieneId	true
- Command Prompt:** A window titled `C:\WINDOWS\system32\cmd.exe` showing the following commands and output:

```
USE version 6.0.0, Copyright (C) 1999-2021 Univ
use> !openter Nav1 eliminarProducto(100 , 20)
precondition `HayProducto' is true
use> !openter Nav1 eliminarProducto(100 , 120)
precondition `HayProducto' is false
Error: precondition false in operation call `Na
use>
```

2. EliminarProductosMalos():

The screenshot shows the first test case for the `EliminarProductosMalos()` method. The test fails because the precondition `'HayProductoMalo' is false` is not satisfied.

Class invariants:

Invariant	Satisfied
Camion::CamionEsCamion	true
Camion::NumeroRuedasCamion	true
Camion::tieneMedidas	true
Compra::CompraCantidad	true
Compra::CompraCuando	true
Compra::CompratieneId	true
Comprador::CompradorMayorista	true
Comprador::CompradortieneId	true
Comprador::TipoMayo	true
Comprador::UnicaCompraDiaMayorista	true
Comprador::dosCompradoresIguales	true
Comprador::idDistintoC	true
Furgoneta::FurgoesFurgo	true
Furgoneta::NumeroPuertasFurgo	true
LineadeCompra::CompraNoVacía	true
LineadeCompra::CompraReal	true
LineadeCompra::LineaTieneCantidad	true
LineadeCompra::LineaTieneId	true
LineadeCompra::LineaTieneTipo	true
LineadeCompra::ProductosL	true
Nave::CalidadVendible	true
Nave::NavetieneId	true

Test Case:

```

use> !openter Nav1 eliminarProductosMalos()
precondition 'HayProductoMalo' is false
Error: precondition false in operation call `Nave::eliminarProductosMalos(self:Nave
  
```

The screenshot shows the second test case for the `EliminarProductosMalos()` method. The test passes because the precondition `'HayProductoMalo' is true` is satisfied.

Class invariants:

Invariant	Satisfied
Camion::CamionEsCamion	true
Camion::NumeroRuedasCamion	true
Camion::tieneMedidas	true
Compra::CompraCantidad	true
Compra::CompraCuando	true
Compra::CompratieneId	true
Comprador::CompradorMayorista	true
Comprador::CompradortieneId	true
Comprador::TipoMayo	true
Comprador::UnicaCompraDiaMayorista	true
Comprador::dosCompradoresIguales	true
Comprador::idDistintoC	true
Furgoneta::FurgoesFurgo	true
Furgoneta::NumeroPuertasFurgo	true
LineadeCompra::CompraNoVacía	true
LineadeCompra::CompraReal	true
LineadeCompra::LineaTieneCantidad	true
LineadeCompra::LineaTieneId	true
LineadeCompra::LineaTieneTipo	true
LineadeCompra::ProductosL	true
Nave::NavetieneId	true
Nave::Navetiencapacidad	true

Test Case:

```

use> !openter Nav1 eliminarProductosMalos()
precondition 'HayProductoMalo' is true
  
```

[illegible]

The screenshot shows the Visual Studio IDE with three windows open:

- Class Diagram:** Displays two classes, `Producto2::Producto` and `Nave1::Nave`. `Producto2::Producto` has attributes `idProducto=100`, `Nombre='x'`, `Tipo=#Carne`, `Stock=100`, `PrecioBase=120`, and `Calidad=#Normal`. `Nave1::Nave` has attributes `idNave=1002`, `Direccion='x'`, `Capacidad=1000`, `Telefono=44455333`, `CalidadTotal=#Buena`, and `Productos=#Carne`. A red line connects the `Productos` attribute of `Nave1::Nave` to the `productos` attribute of `Producto2::Producto`.
- Class Invariants:** A table listing invariants for various classes and their satisfaction status.
- Code View:** Shows C# code for the `Nav1` class, including a constructor and a method `ofertarProductos()`.

Invariant	Satisfied
Camion::CamionEsCamion	true
Camion::NumeroRuedasCamion	true
Camion::tieneMedidas	true
Compra::CompraCantidad	true
Compra::CompraCuando	true
Compra::CompratieneId	true
Comprador::CompradorMayorista	true
Comprador::CompradortieneId	true
Comprador::TipoMayo	true
Comprador::UnicaCompraDiaMayorista	true
Comprador::dosCompradoresIguales	true
Comprador::idDistintoC	true
Furgoneta::FurgoesFurgo	true
Furgoneta::NumeroPuertasFurgo	true
LineadeCompra::CompraNoVacía	true
LineadeCompra::CompraReal	true
LineadeCompra::LineaTieneCantidad	true
LineadeCompra::LineaTieneId	true
LineadeCompra::LineaTieneTipo	true
LineadeCompra::ProductosL	true
Nave::NavetieneId	true
Nave::Navetiencapacidad	true

```

using System;
using Nav1;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Clase_1
{
    class Nav1
    {
        public string idNave = "1002";
        public string Direccion = "x";
        public int Capacidad = 1000;
        public string Telefono = "44455333";
        public string CalidadTotal = "#Buena";
        public string Productos = "#Carne";

        public Nav1()
        {
            // Constructor logic
        }

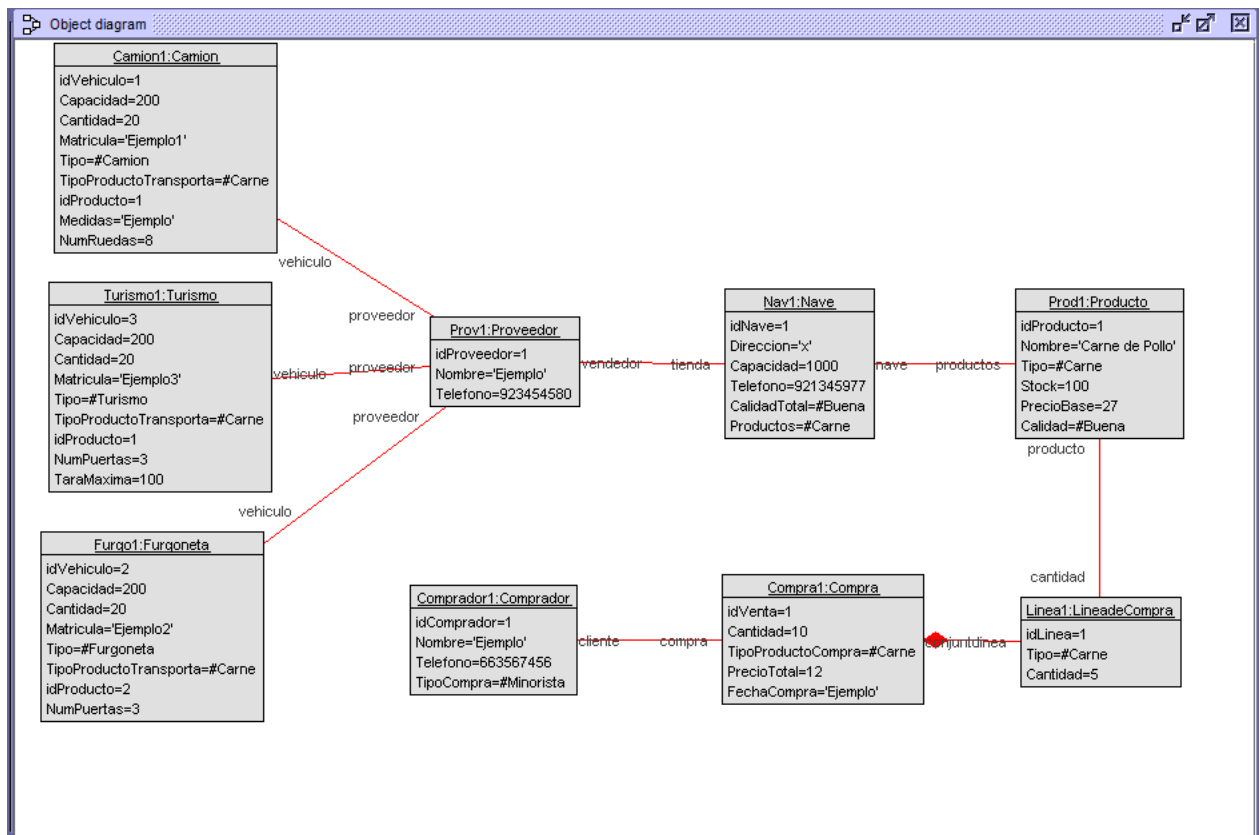
        public void ofertarProductos()
        {
            precondition `HayProductoNormal` is true
            // Method logic
        }
    }
}
  
```

[illegible]

Pág. 13

4. Script de Creación

Hemos creado un pequeño archivo con extensión **txt** para la creación del siguiente diseño de instancias, para que a partir del mismo se pueda modificar diferentes objetos según lo necesitemos, para realizar mejor las comprobaciones de las invariantes.



5. Conclusión

En conclusión, nuestro trabajo está basado en una interpretación del sistema creada desde nuestra propia perspectiva del enunciado de la práctica, esto implica que se podría llegar a otro posible resultado al que hemos obtenido aplicando otro punto de vista diferente, aún así, siempre y cuando se comparta la estructura base del sistema, resultará poco complicado cambiar algunas invariantes, clases (ya sean atributos o enlaces) o contratos, según las necesidades de nuestro cliente para adaptar el sistema y satisfacer los requisitos deseados.

Cabe destacar, que en el propio transcurso de la práctica nos hemos dado cuenta que podría haber invariables que se pisaran unas con otras, indicando lo mismo o lo contrario pero de otra forma. Por ello hemos tenido que ir con cuidado para no realizar incongruencias en el desarrollo del trabajo.