Universidad de Huelva

MFIS

Práctica 1

ETSI

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA

uhu.es

Autores:

- Ismael Da Palma Fernández
- Cristian Delgado Cruz

- Juan Jimenez Serrano
- Fernando García-Palomo Albarrán

Índice

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 CompraNoVacia: 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)

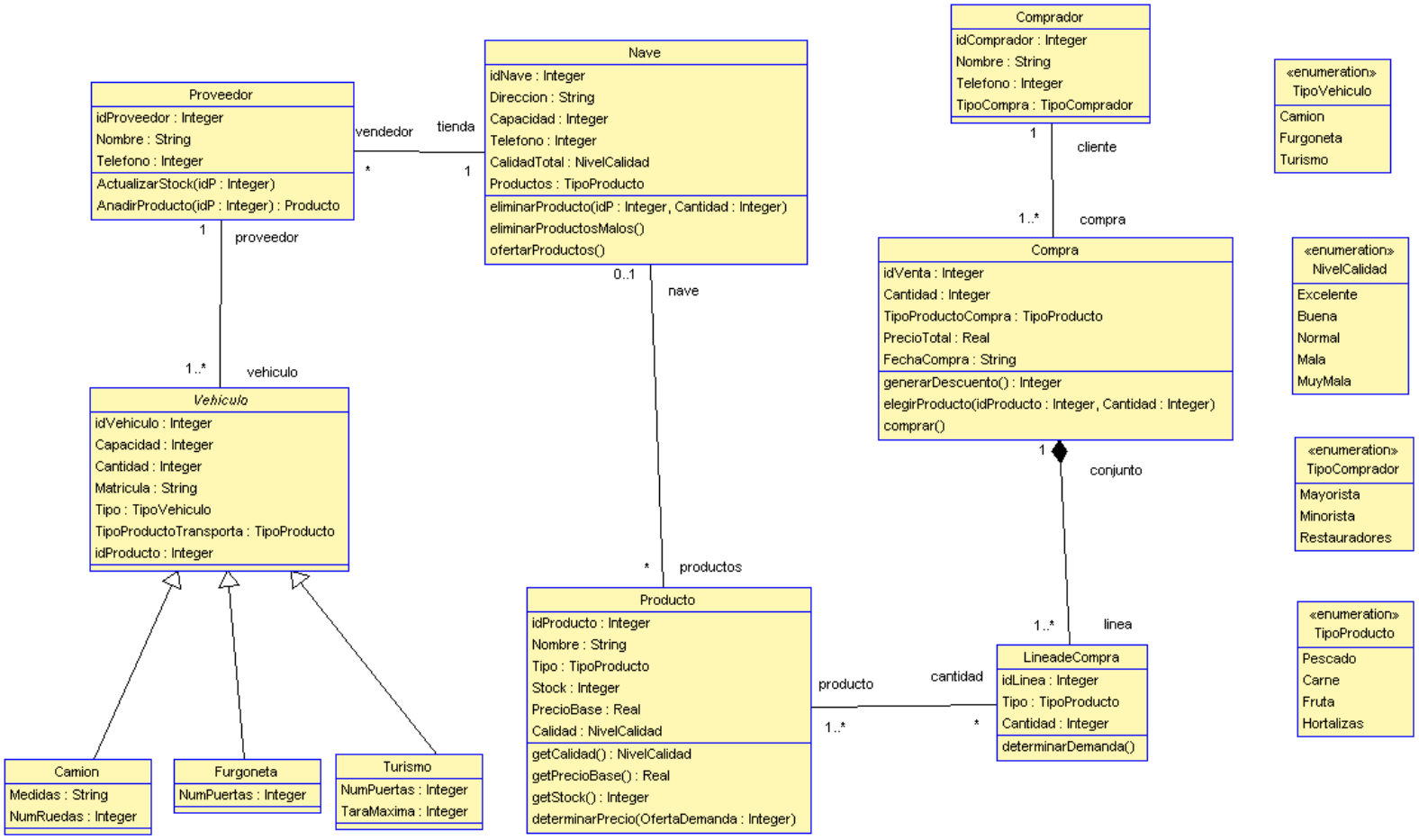
4) Método de *Compra*, **GenerarDescuento():** si el precio de compra Total es mayor que 50€ o por otra parte la cantidad comprada supera los 25 productos, se aplicará un descuento variable, siendo un mínimo de 25%.

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)

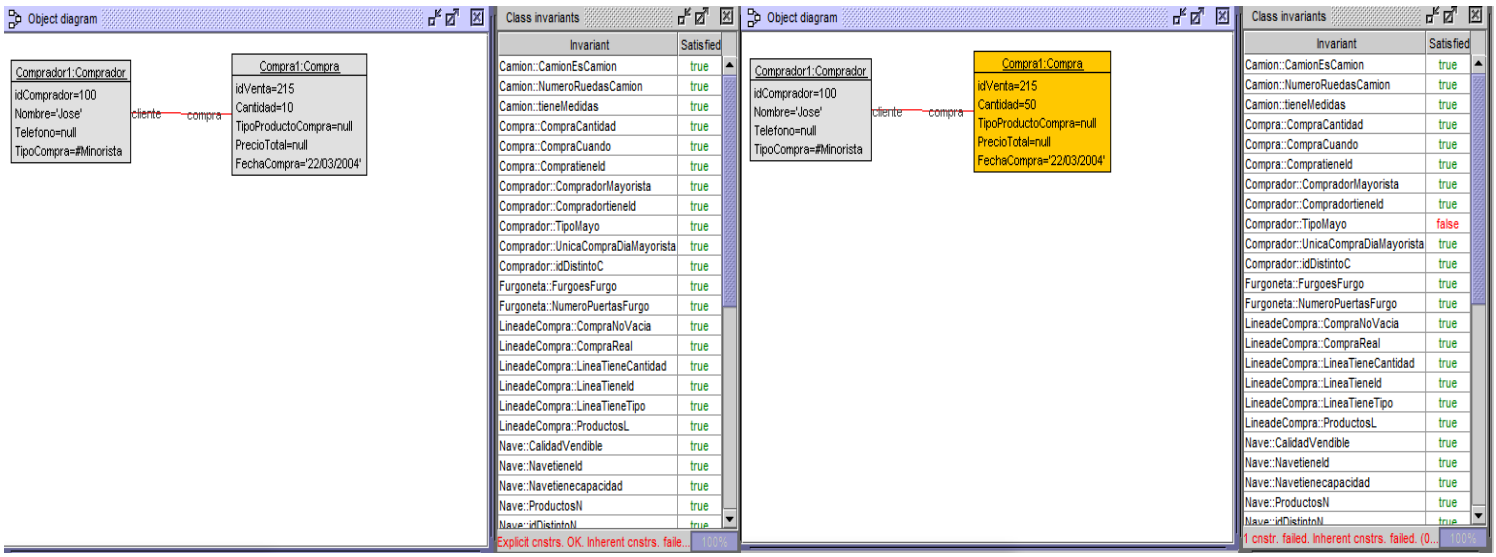
2. Diagrama de Clases

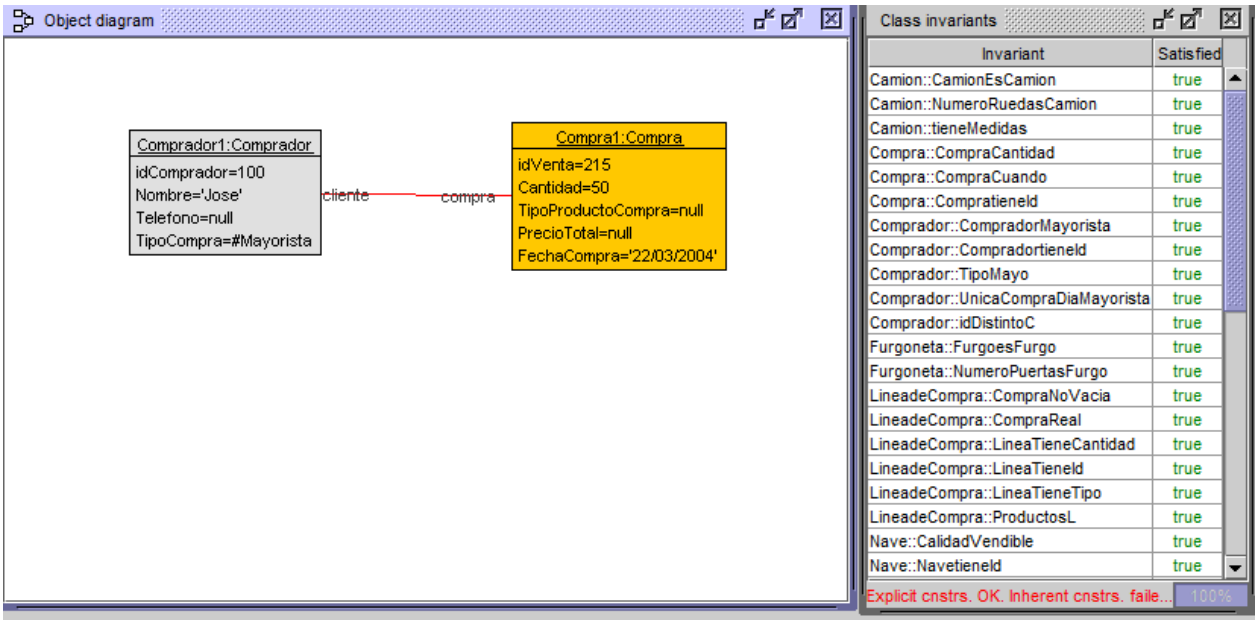


3. Comprobación de restricciones

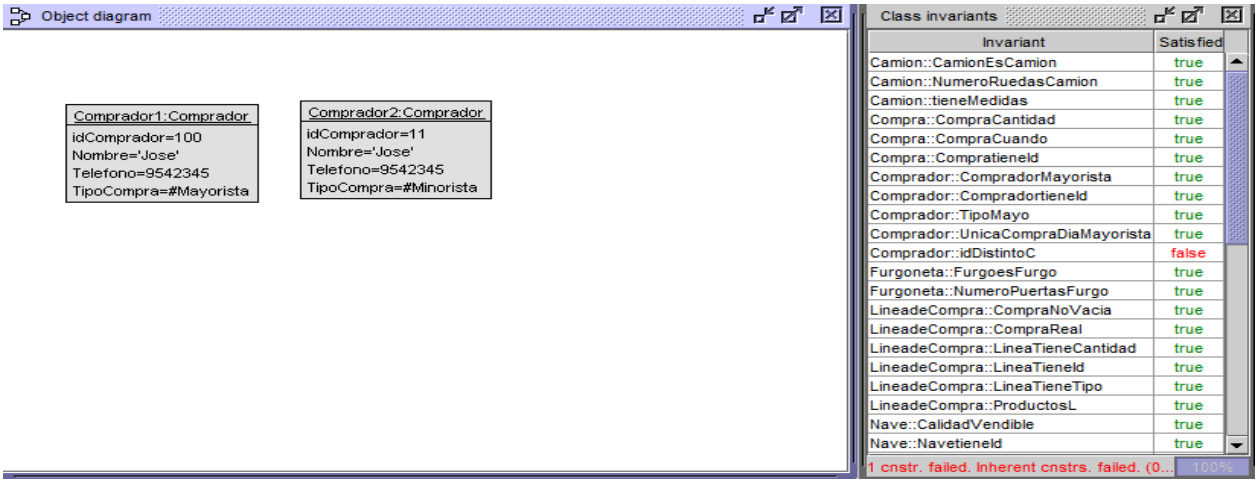
3.1. Invariantes

1. inv TipoMayo:

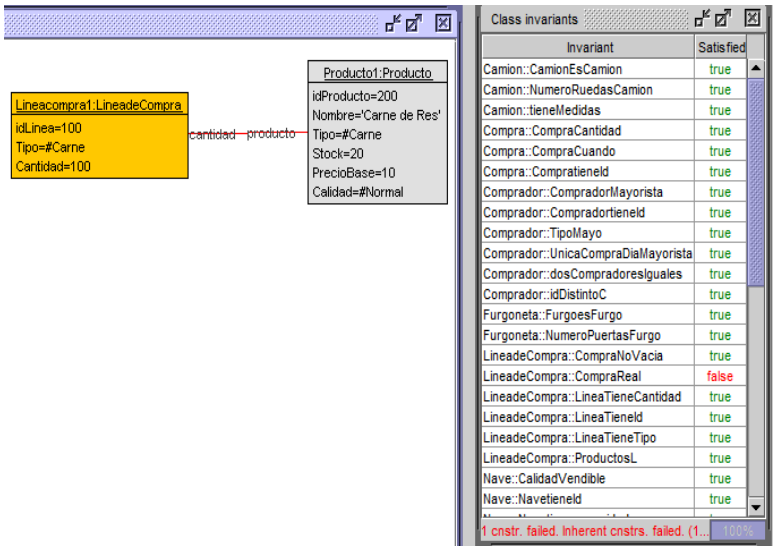


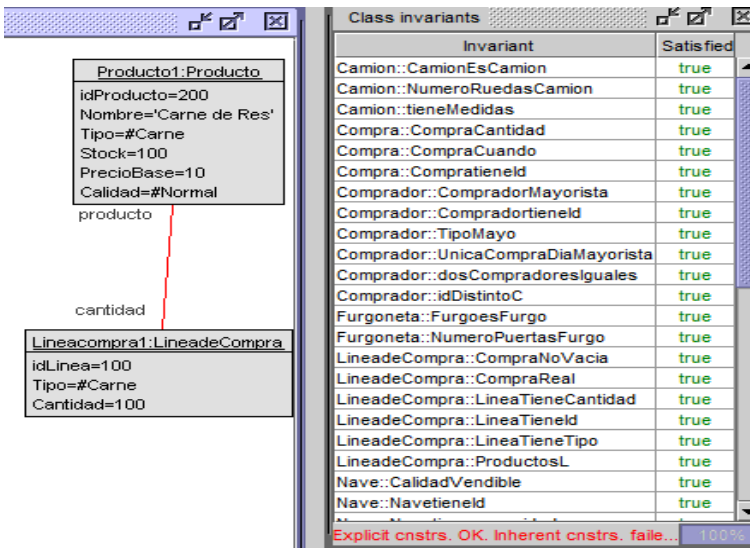


2. inv idDistintoC:

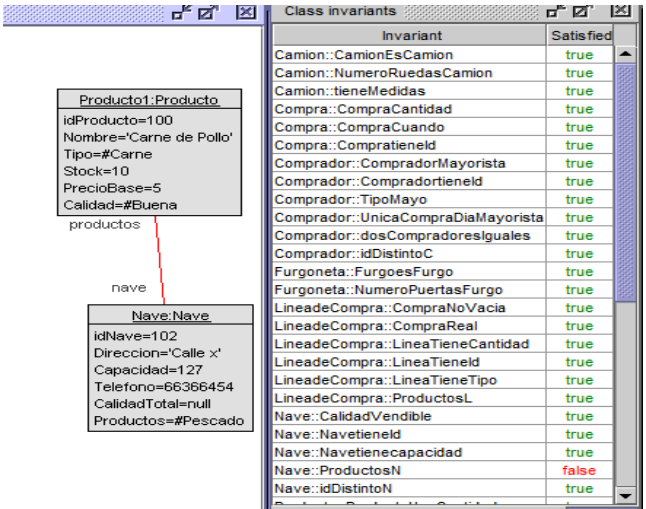


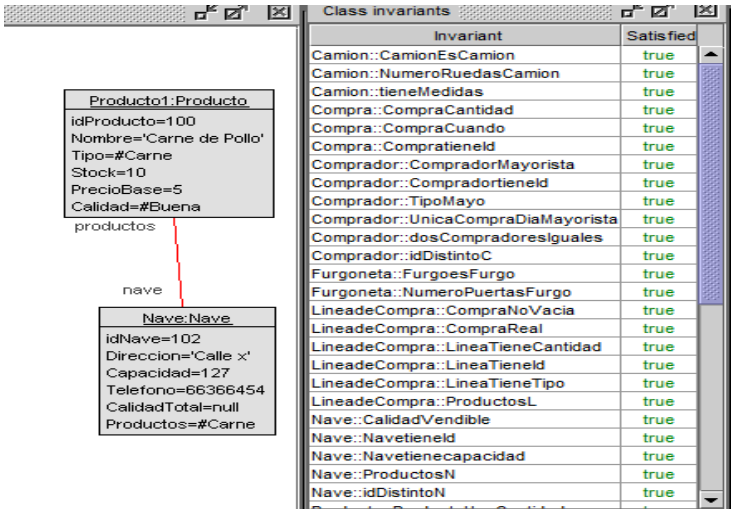
3. inv CompraReal:



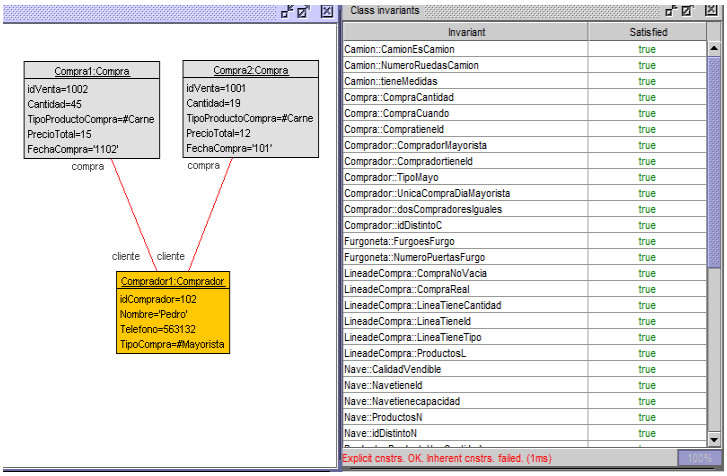


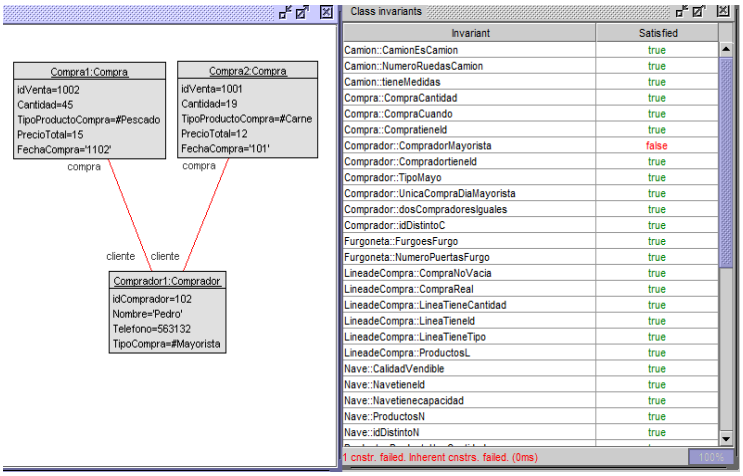
4. inv ProductosN:



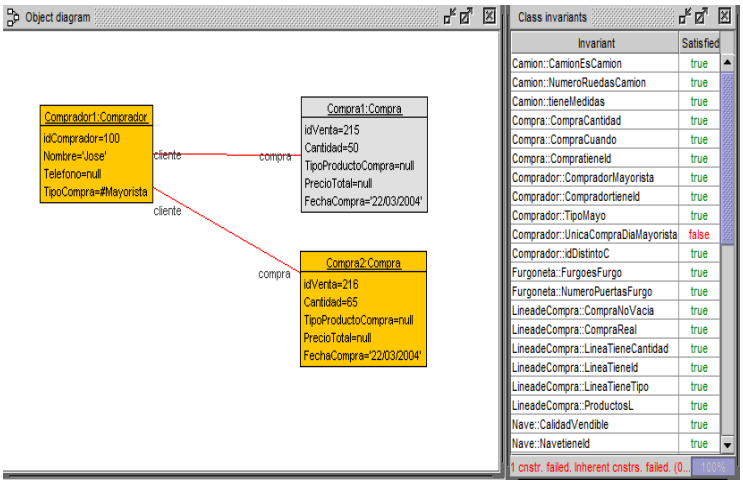


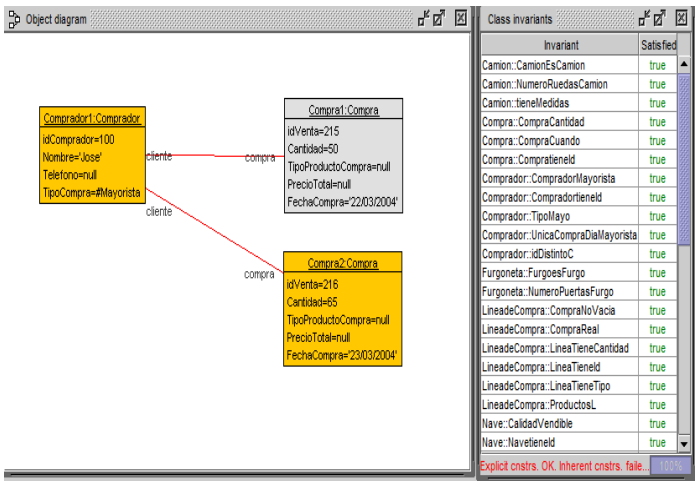
5. inv CompradorMayorista:



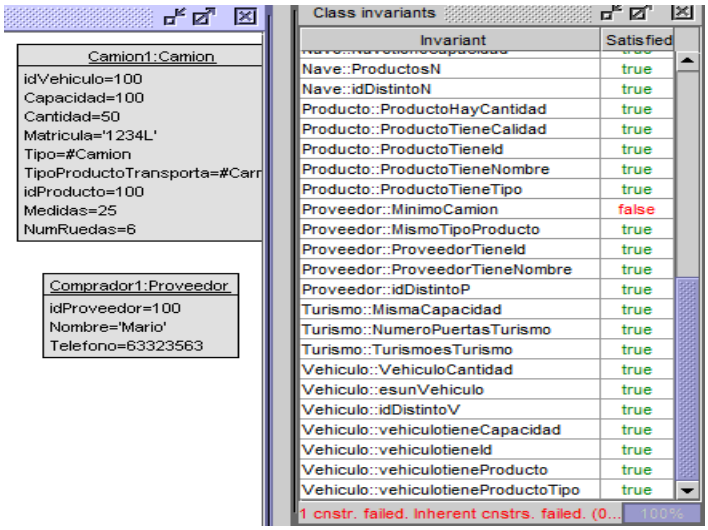


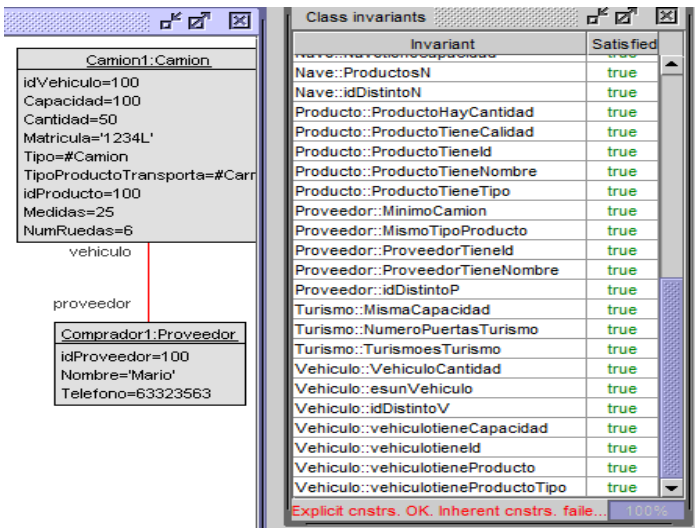
6. inv UnicaCompraDiaMayorista:



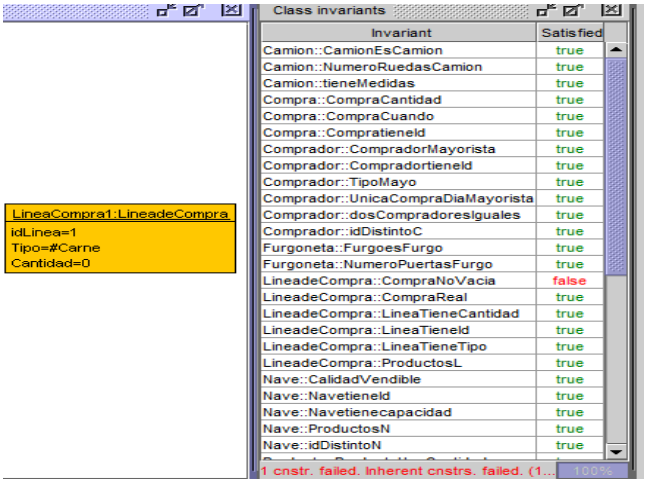


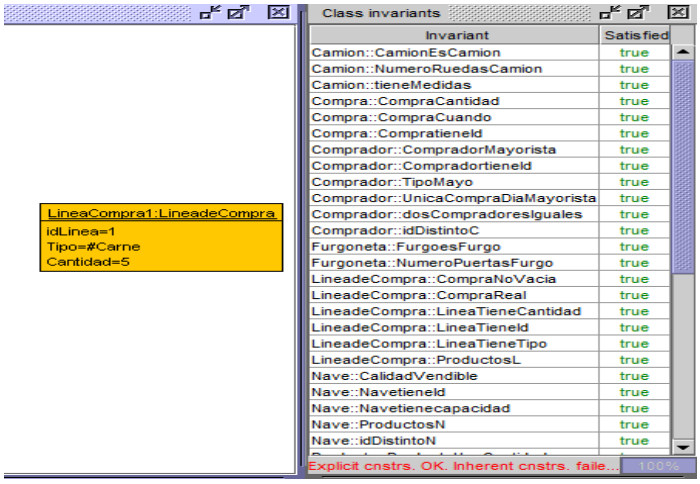
7. inv MinimoCamion



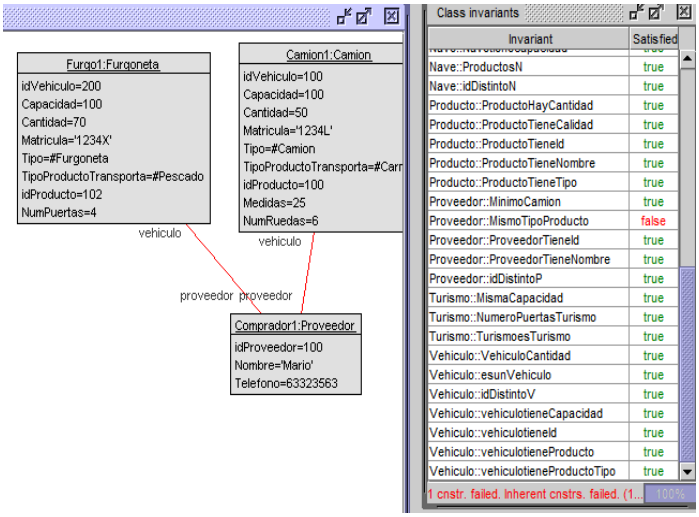


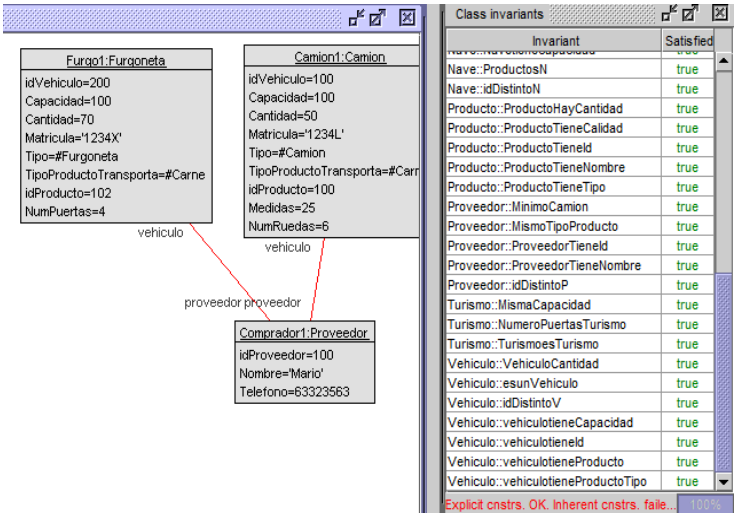
8. inv CompraNoVacia:





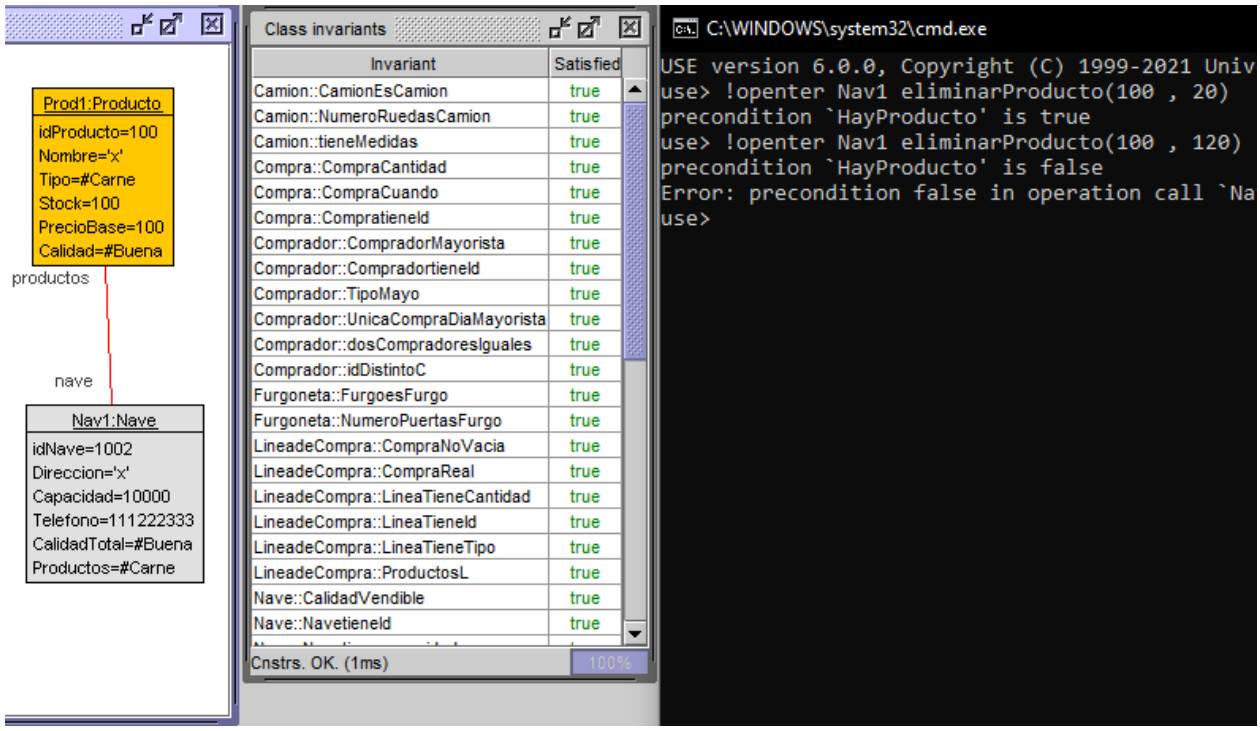
9. inv MismoTipoProducto:



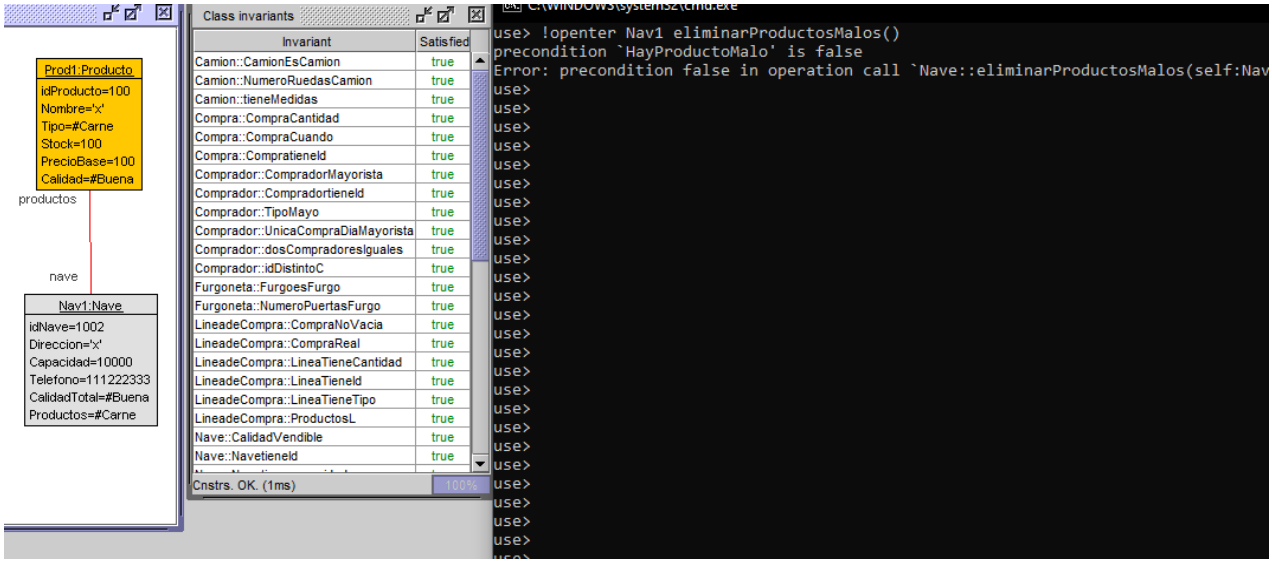


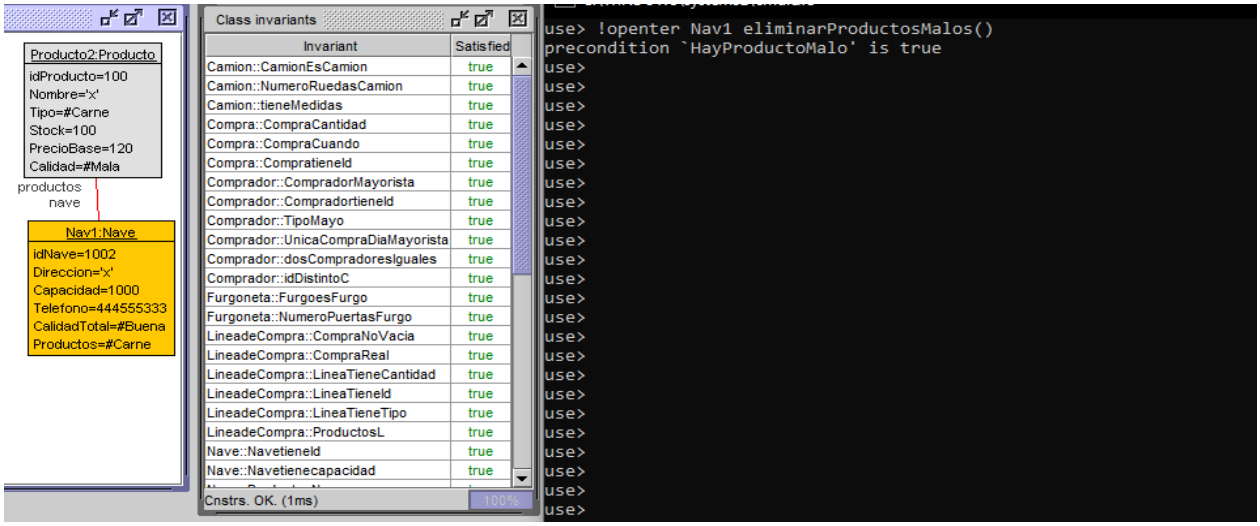
3.2. Contratos

1. **EliminarProducto(int idP , int Cantidad):**

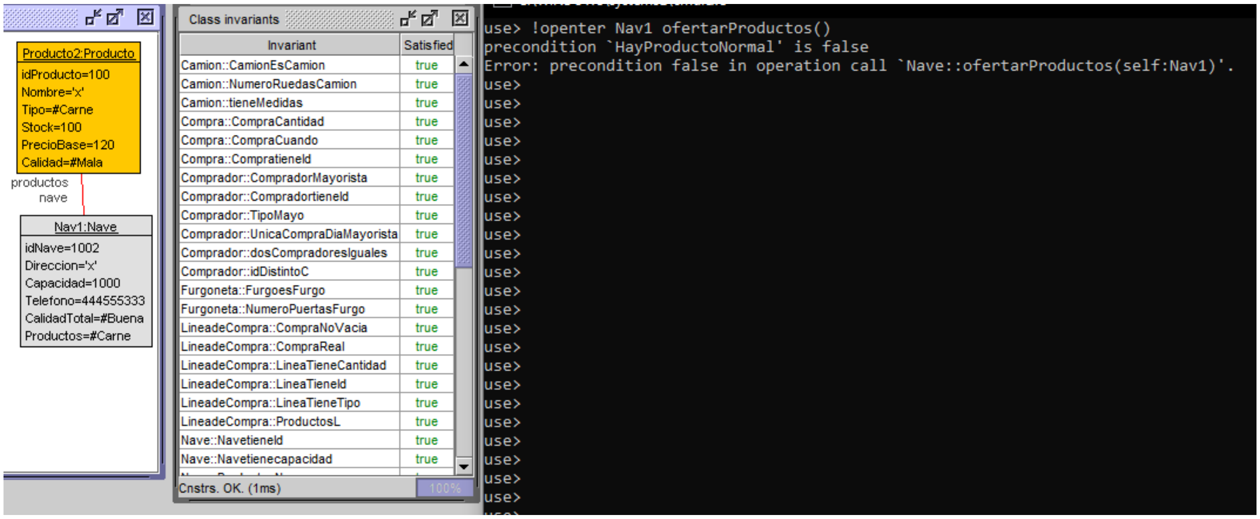


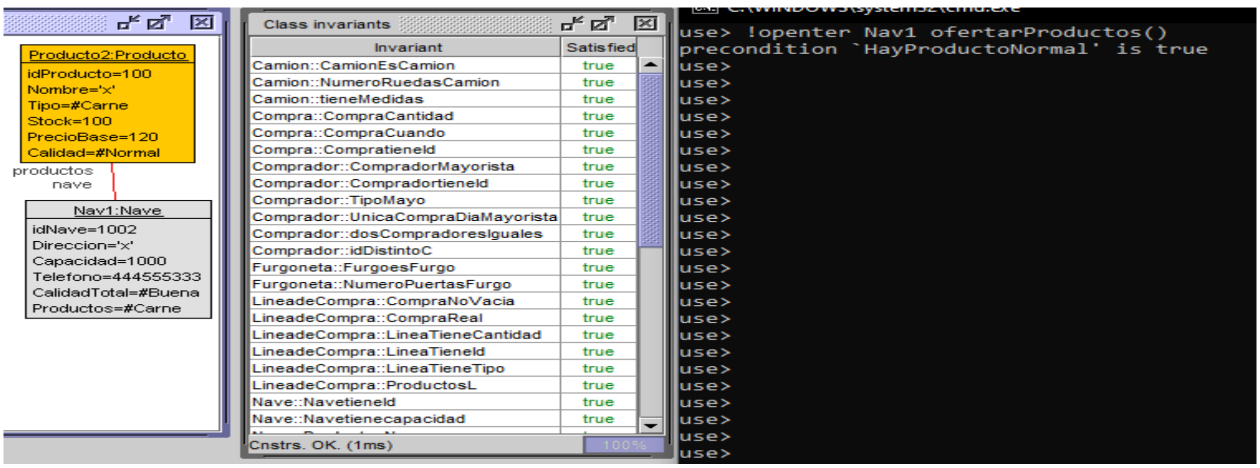
2. EliminarProductosMalos():

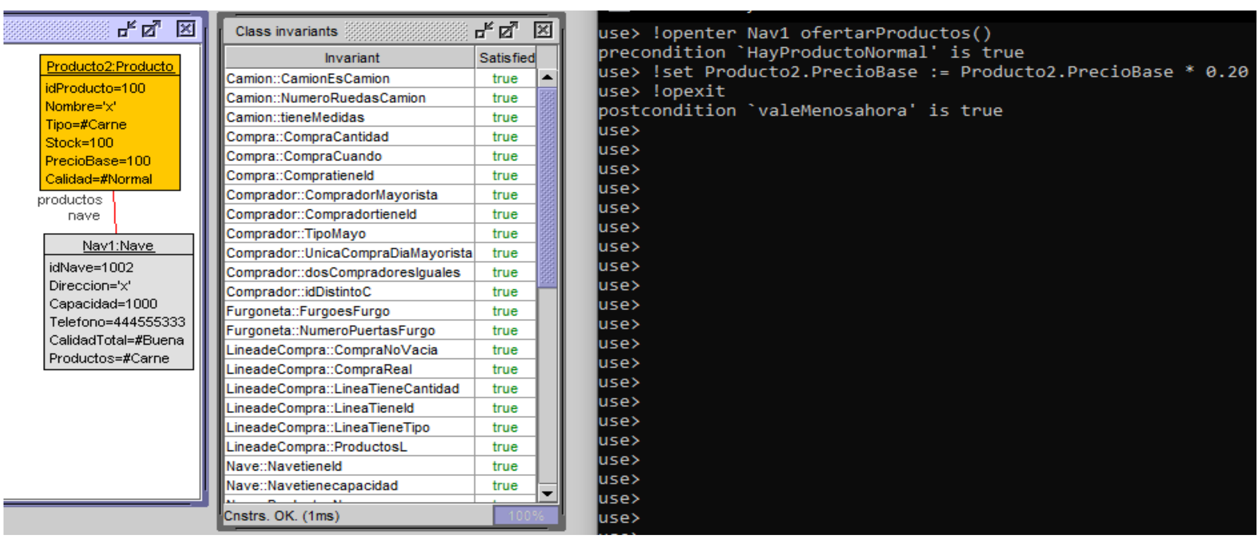




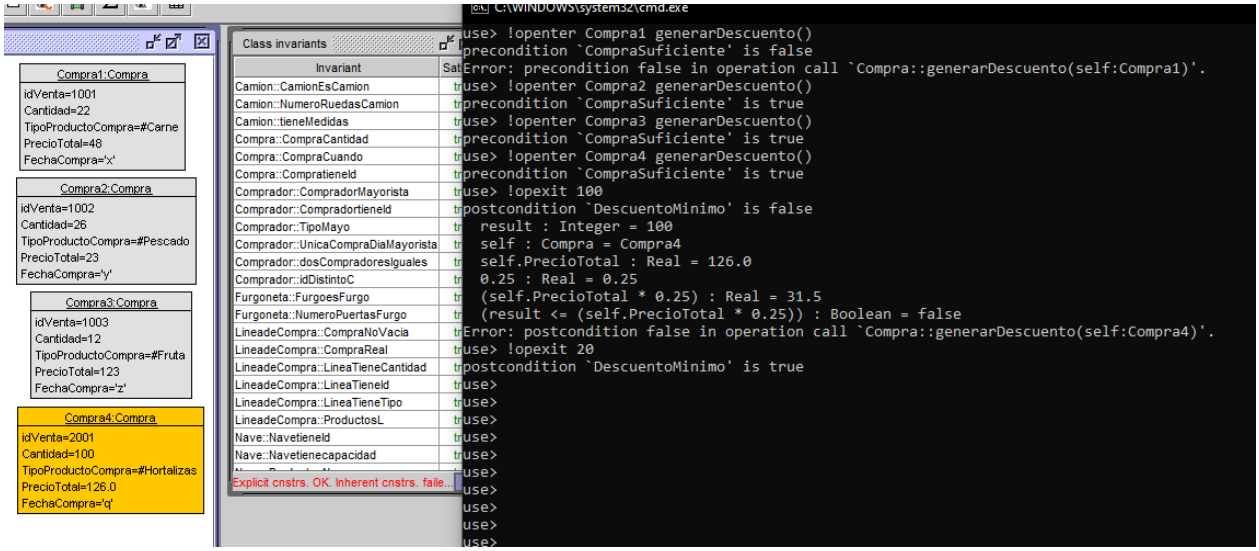
3. OfertarProductos():



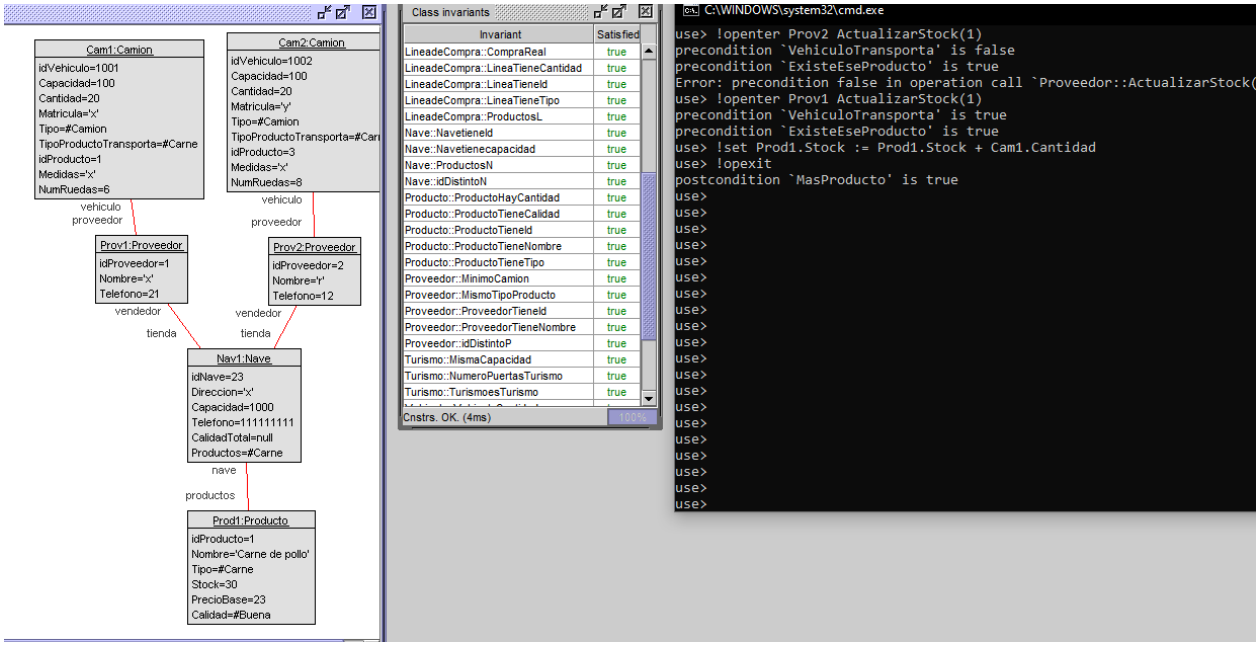




4. GenerarDescuento():

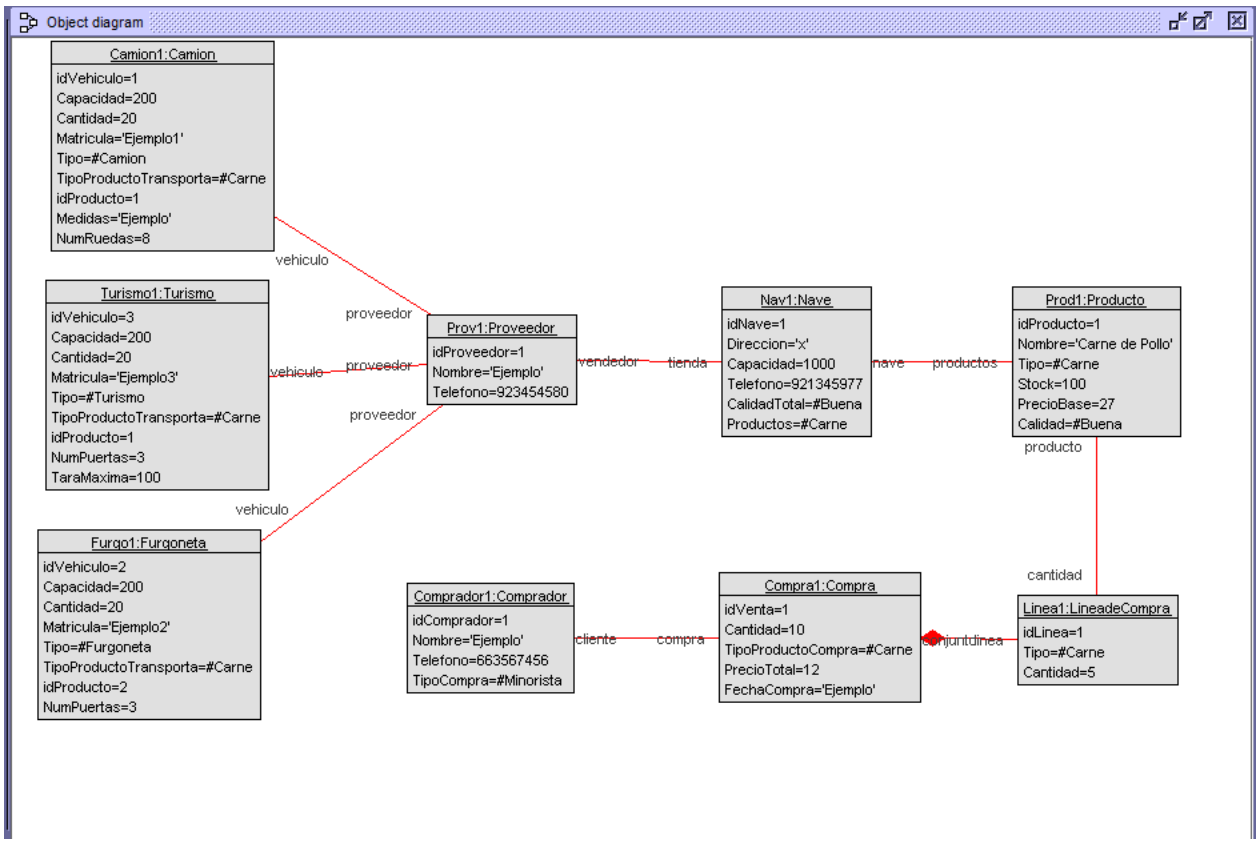


5. ActualizarStock():



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.