Ejemplo de ejercitación: paso a paso

Sitio: Agencia de Habilidades para el Futuro
Curso: Desarrollo de Sistemas Orientado a Objetos 1º D
Libro: Ejemplo de ejercitación: paso a paso

Imprimido por: Eduardo Moreno
Día: lunes, 5 de mayo de 2025, 11:14

Tabla de contenidos

1. Preguntas orientadoras

2. Introducción

3. Enunciado problema

4. Posible resolución
4.1. Programando las clases

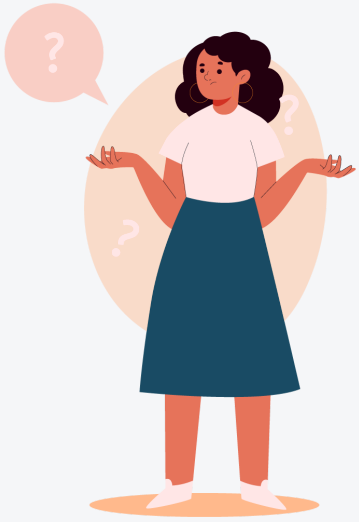
5. Agregando funcionalidad
5.1. Programando los métodos...

6. En resumen

7. Práctica para repasar
7.1. Claves de autocorrección

Preguntas orientadoras

- ¿Cómo vamos a detectar las clases de nuestro programa?
- ¿Cómo detectar los atributos?
- ¿Cómo detectar los métodos?
- ¿Cómo comienzo a analizar un problema en base a un requerimiento?
- ¿Cómo pasamos del diagrama UML a código C#?



Introducción

En las semanas previas hemos abordado una variedad de temas fundamentales en la Programación Orientada a Objetos (POO).

¿Qué aprendimos? En principio, la importancia de la encapsulación, que nos permite ocultar los detalles internos de una clase y proporcionar una interfaz controlada para interactuar con los objetos. También hemos explorado la abstracción, que nos permite modelar entidades del mundo real y representarlas en forma de clases y objetos. Mediante la abstracción, podemos identificar las características y comportamientos esenciales de un objeto y representarlos en nuestro código de manera eficiente y comprensible.

Por otra parte, resaltamos la importancia de la asociación simple, que nos permite establecer relaciones entre objetos y crear conexiones entre objetos que pueden interactuar entre sí sin una relación de dependencia permanente.

En este libro, rescatamos la necesidad de ejercitar. Por eso, podrás recorrer un ejercicio, paso a paso. A medida que avanzamos, es crucial que dediquemos tiempo a revisar y consolidar los conocimientos adquiridos hasta el momento.

¡Manos a la obra!

Carrito de compras

La empresa **Vendo todo S.A.** nos convoca a sumarnos a su equipo de desarrollo ya que necesita modelar el correcto funcionamiento de su futuro sitio de comercio online.

La empresa, o el sistema, cuenta con un listado de productos disponibles para adquirir. Cada producto tiene un número identificador correlativo, un nombre único, precio unitario y su stock actual.

Además, el sistema debe contar con un único carrito de compras. El mismo se crea al iniciar una compra, recibiendo por parámetro el dni de la persona que va a comprar y el número identificador correlativo. Al finalizar o descartar una compra, el carrito se elimina. Además, cuenta con un listado de los ítems que se van a adquirir. Cada uno de esos elementos contiene id, nombre, precio unitario y la cantidad solicitada del mismo. Esta cantidad solicitada nunca puede ser inferior a 1.

Se deben tener en cuenta las siguientes...

Se pide desarrollar los siguientes métodos en las clases que correspondan.

01 **iniciarCompra**. Recibe un dni y crea el carrito. El método debe retornar un booleano indicando si se pudo crear el carrito o no (recordá que no se puede crear un nuevo carrito si ya hay uno definido).

02 **registrarProducto**. Recibe un nombre, un precio unitario y un stock inicial. Con dichos datos, agrega un nuevo producto a la lista de productos disponibles de la empresa. El método retorna verdadero si se pudo registrar el producto, o falso si el stock proporcionado es negativo.

03 **agregarProductoCarrito**. Recibe un nombre de producto y una cantidad. El método retorna uno de los siguientes posibles resultados:

- "COMPRA_NO_INICIADA", si aún no se inició la compra
- "PRODUCTO_INVALIDO", si el producto especificado no existe
- "NO_HAY_STOCK", si el producto no cuenta con la cantidad solicitada en stock
- "AGREGAR_OK", si el producto se pudo agregar correctamente al carrito.

En este caso, se debe actualizar el stock restante de dicho producto para próximas compras. Tener en cuenta, que si el producto ya estaba previamente en el carrito, solo se debe sumar su cantidad. Si no existía, se debe crear el elemento en el carrito.

04 **finalizarCompra**. Si existe el carrito, muestra la lista de los productos adquiridos y el total a abonar. Luego, descarta el carrito (descartar implica eliminar la referencia a dicho carrito). Devuelve un booleano indicando la confirmación de la finalización de compra.

05 **descartarCompra**. Si existe el carrito lo descarta. Devuelve un booleano indicando la confirmación del descarte del carrito.

Podés definir todos los métodos que consideres necesarios para mantener el correcto encapsulamiento y distribución de responsabilidades entre las clases del sistema.
Te dejamos la Clase Test a utilizar.

Recomendación: andá agregando los métodos a medida que vayas teniendo las clases y métodos codificadas, así el programa siempre compila.

Posible resolución

Para detectar las clases en un programa, puedes seguir estos pasos:

01 Identificar los sustantivos y entidades relevantes en el problema o dominio del programa. Las clases suelen representar objetos o conceptos del mundo real.

02 Analizar las propiedades o atributos que podrían tener los objetos pertenecientes a cada clase. Estos atributos representarán las características o datos que cada objeto debe poseer.

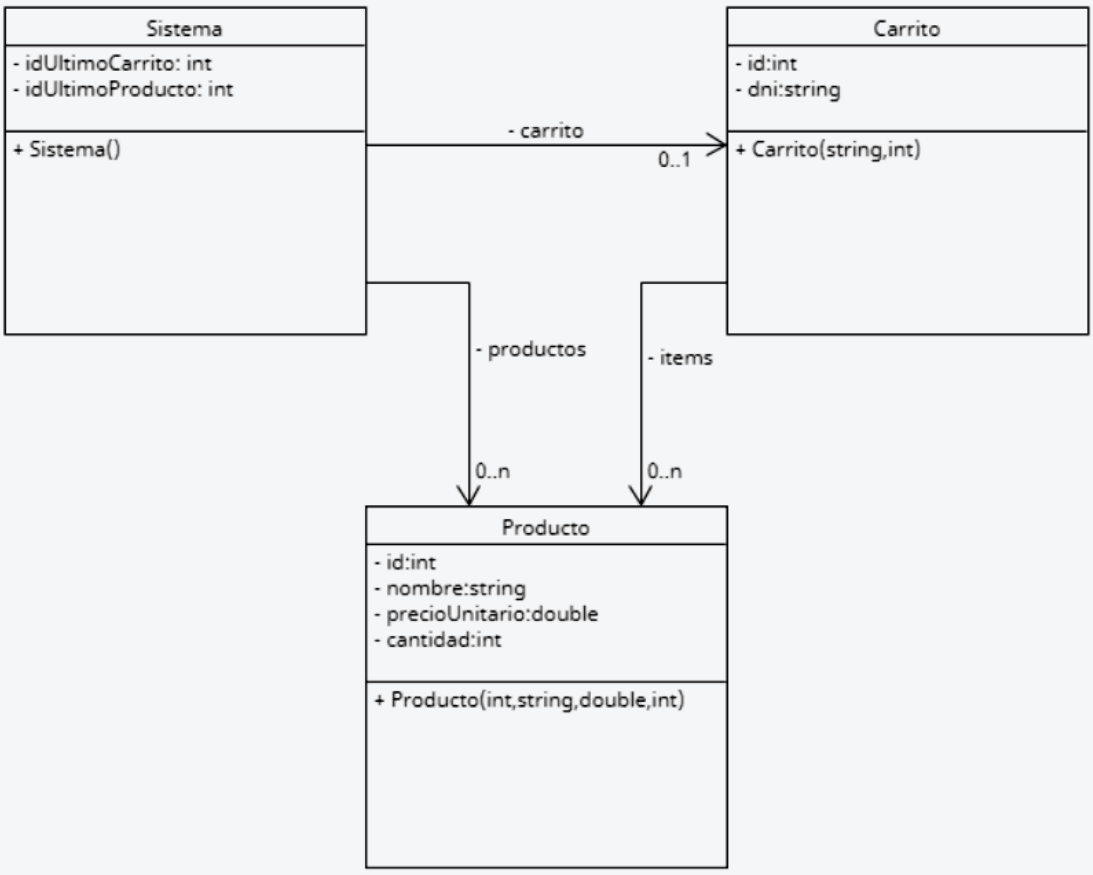
03 Identificar los comportamientos o métodos que los objetos de cada grupo deben poder realizar. Estos métodos representarán las acciones que los objetos pueden llevar a cabo.

04 Definir las clases en base a lo identificado, asignando los atributos y métodos adecuados a cada una.

05 Refinar y ajustar las clases según sea necesario.

Es importante tener en cuenta que este proceso puede requerir iteraciones y ajustes a medida que se comprende mejor el problema y las interacciones entre los objetos. Además, es recomendable seguir buenas prácticas de diseño de software para crear clases más eficientes y fáciles de mantener.

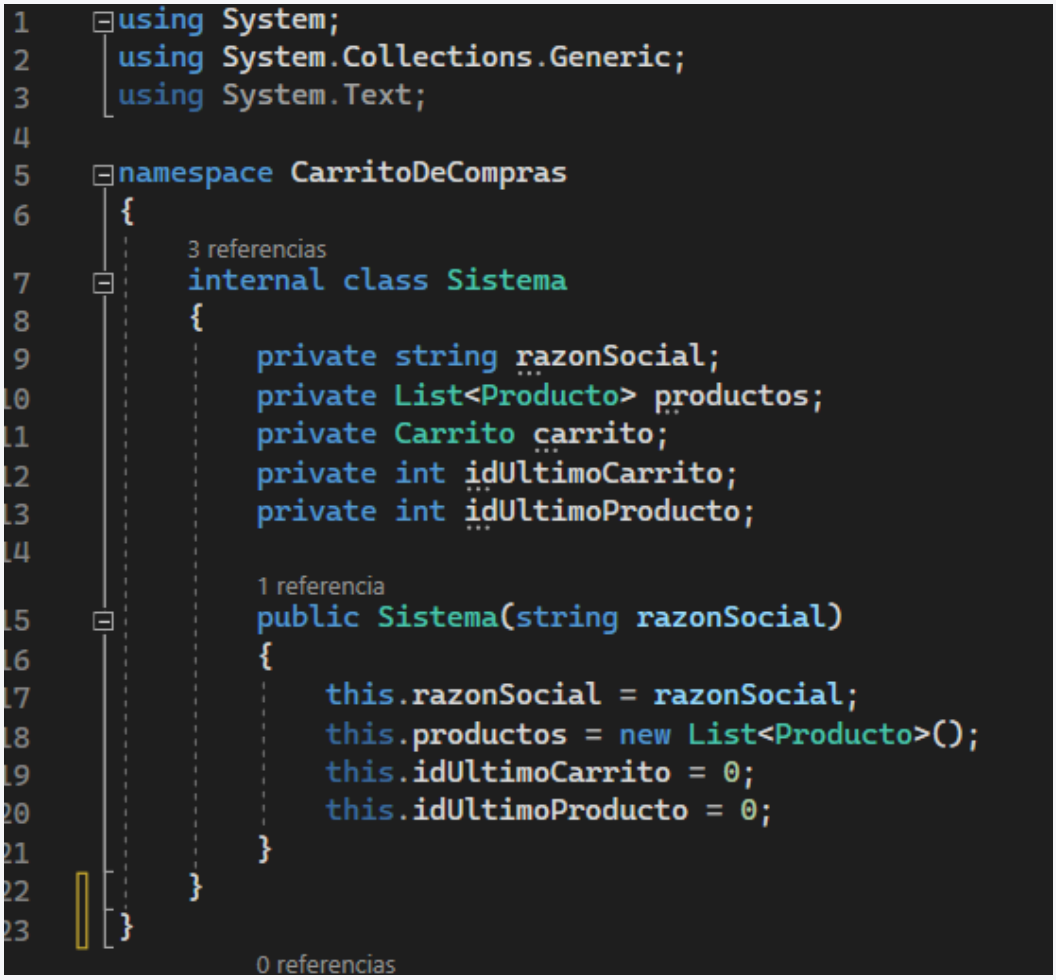
Detectamos las clases según el enunciado junto con su constructor, atributos y relaciones:



Programando las clases

Comenzamos a programar en código las clases con sus atributos y constructores

Clase Sistema:



```
using System;
using System.Collections.Generic;
using System.Text;

namespace CarritoDeCompras
{
    3 referencias
    internal class Sistema
    {
        private string razonSocial;
        private List<Producto> productos;
        private Carrito carrito;
        private int idUltimoCarrito;
        private int idUltimoProducto;

        1 referencia
        public Sistema(string razonSocial)
        {
            this.razonSocial = razonSocial;
            this.productos = new List<Producto>();
            this.idUltimoCarrito = 0;
            this.idUltimoProducto = 0;
        }
    }
}
```

0 referencias

Clase Carrito:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace CarritoDeCompras
{
    3 referencias
    internal class Carrito
    {
        private int id;
        private String dni;
        private List<Producto> items;

        1 referencia
        public Carrito(String dni, int it)
        {
            setId(id);
            setDni(dni);
            items = new List<Producto>();
        }
    }
}
```

Clase Producto:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace CarritoDeCompras
{
    17 referencias
    internal class Producto
    {
        private int id;
        private String nombre;
        private double precioUnitario;
        private int cantidad;

        2 referencias
        public Producto(int id, String nombre, double precioUnitario, int cantidad)
        {
            this.id = id;
            this.nombre = nombre;
            this.precioUnitario = precioUnitario;
            this.cantidad = cantidad;
        }
    }
}
```

Agregando Funcionalidad

Para detectar los métodos en un programa orientado a objetos, puedes seguir estos pasos:

01 Analizar las acciones o tareas que los objetos deben poder llevar a cabo dentro del contexto del problema o dominio.

02 Identificar los verbos o acciones relevantes en el problema. Estos verbos pueden sugerir las operaciones o comportamientos que los objetos necesitan realizar.

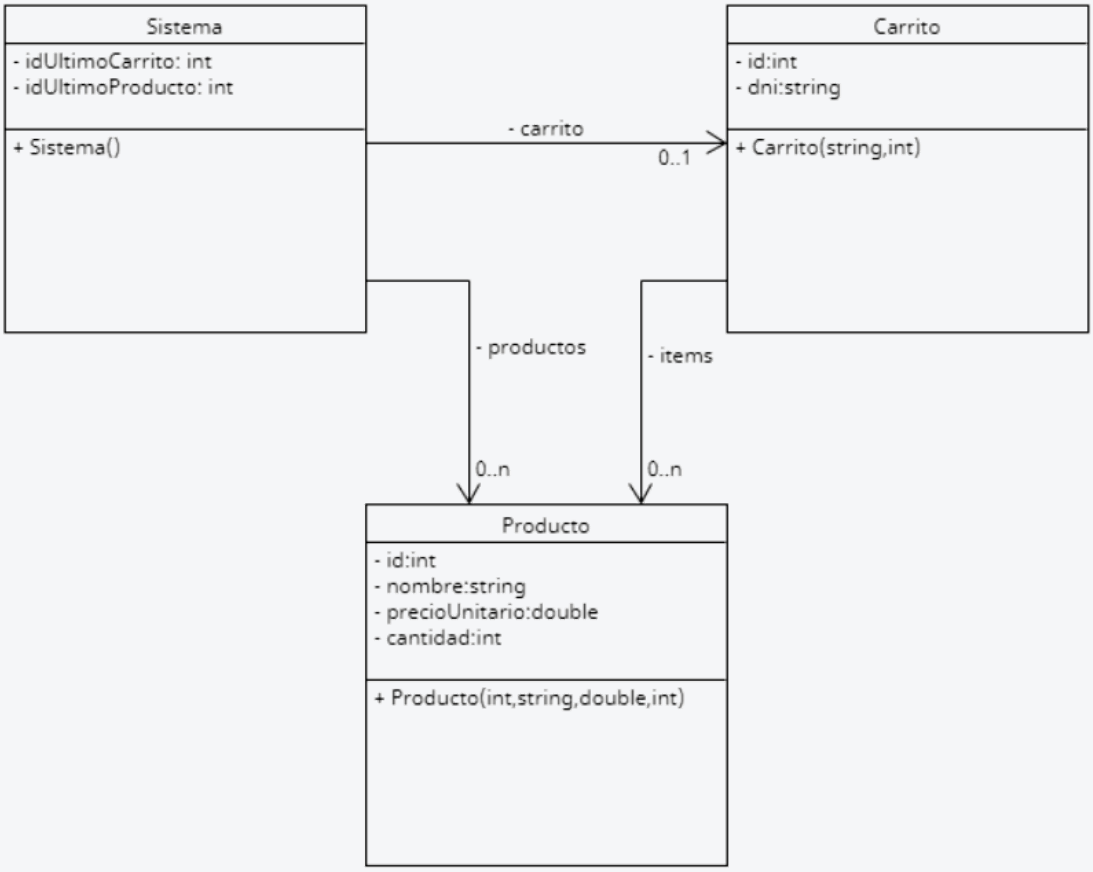
03 Agrupar las acciones relacionadas en conjuntos lógicos. Esto puede ayudarte a identificar posibles métodos y clases.

04 Asignar cada acción a la clase correspondiente. Un método es una función o procedimiento asociado a una clase específica.

05 Definir los métodos dentro de las clases, estableciendo los parámetros de entrada necesarios y el tipo de retorno, si corresponde.

06 Considerar la coherencia y consistencia en la nomenclatura de los métodos. Es recomendable seguir convenciones de nomenclatura adecuadas, como el uso de verbos en infinitivo o camel case, para que el código sea más legible y comprensible.

Es importante recordar que los métodos representan el comportamiento de los objetos y deben ser coherentes con la responsabilidad y funcionalidad de cada clase. Además, es posible que necesites iterar y ajustar los métodos a medida que comprendas mejor el problema y las interacciones entre los objetos en tu programa



Programando los métodos...

Hemos planteado y desarrollado la resolución del problema en UML, incluyendo los métodos necesarios. Sin embargo, antes de revisar nuestra solución, te animamos a intentar resolverlo basándote en el diagrama UML proporcionado.

El **diagrama UML** te brinda una representación visual de las clases, sus atributos y los métodos que se requieren. Podes utilizarlo como guía para diseñar la estructura del programa y definir los detalles de implementación.

Al intentar resolver el problema, tendrás la oportunidad de aplicar tus conocimientos de programación orientada a objetos y practicar la creación de clases, atributos y métodos. También te permitirá fortalecer tu comprensión de cómo se relacionan las clases y cómo implementar la lógica del programa.

Una vez que hayas intentado resolver el problema, puedes comparar tu solución con la que hemos desarrollado.

Recorda que la resolución de problemas y el diseño de programas son habilidades valiosas, y este ejercicio te brinda la oportunidad de practicarlas.

¡Te deseamos mucho éxito en tu intento de resolver el problema utilizando el diagrama UML como guía!

En resumen

Te dejamos una posible solución completa para que puedas descargar del ejercicio del carrito de compras en el sistema de comercio online de "Vendo todo S.A.".

La solución presentada demuestra un diseño sólido y coherente, donde se han aplicado adecuadamente los principios de abstracción, encapsulamiento y asociación simple. La implementación de las entidades principales, como la clase Sistema, Producto y CarritoDeCompras, muestra un manejo adecuado de las propiedades y métodos necesarios para el correcto funcionamiento del sistema.

La validación de las reglas de negocio, como la verificación de la existencia de un carrito de compras previo antes de iniciar una nueva compra, y la verificación de disponibilidad de stock al agregar un producto al carrito, reflejan una comprensión clara de los requisitos y restricciones del sistema.

Espero que esta experiencia les haya permitido consolidar su comprensión de la programación orientada a objetos y les haya brindado confianza en su capacidad para enfrentar desafíos similares en el futuro.

Recordá que el aprendizaje es un proceso continuo y siempre hay oportunidades de mejora.

Sigamos avanzando juntos en nuestro viaje hacia la excelencia en la programación orientada a objetos.

¡Hasta la próxima y continúen brillando en su camino hacia el dominio de la programación orientada a objetos!

Repaso de contenidos para practicar

- **Tipo:** Práctica formativa de repaso con claves de autocorrección.
- **Objetivo:** Repasar los temas abordados en la unidad 1 y unidad 2.

Descripción:

Una empresa de trenes de carga quiere desarrollar un sistema para administrar el traslado de granos de trigo. Sólo se transporta un único producto (trigo) y todo lo transportado es homogéneo.

Los trenes se identifican por un número correlativo, a partir de 1 en adelante. Esta numeración es automática (no la ingresa el usuario sino que la indica el sistema cuando se crea un nuevo tren).

Cada formación (cada tren) tiene hasta 30 vagones. Existen tres tipos de vagones, denominados SMALL, MEDIUM y LARGE que son capaces de cargar 30, 40 y 50 toneladas de trigo respectivamente.

¿Qué te proponemos realizar?

Desarrollar en la/s clase/s que corresponda:

- Un método llamado crearFormacion, que crea un tren sin vagones. A cada tren se le asigna un número correlativo. Este método no recibe ningún parámetro y debe retornar el número de tren asignado a esa formación.
- Un método llamado agregarVagones que reciba un número de tren, una cantidad de vagones y un tipo de vagón para (de ser posible) crear vagones del tipo requerido agregándolos al final de la formación. Devuelve un enumerado con los siguientes valores posibles:.
  - NO_EXISTE_TREN: si el número no corresponde a un tren existente.
  - CANT_VAGONES_INVALIDA: si la cantidad de vagones es menor o igual que cero, o bien si la cantidad total de vagones (los existentes en el tren más los que se quieren agregar) excede el largo máximo posible.
  - AGREGADO_OK: si la operación se pudo realizar correctamente.
- Un método llamado cargarTren, que reciba un número de tren y una cantidad de toneladas de trigo a cargar. Para realizar la carga de los granos el tren debe existir y tener capacidad suficiente para acomodar la totalidad de la carga. Este método debe retornar verdadero si se pudo realizar la operación y falso en caso contrario.
- Un método llamado listarCapacidadDisponible que muestre todos los trenes con el porcentaje de espacio libre de cada formación (la carga acumulada de todos los vagones contra la capacidad total de cada tren).
- Un método llamado sacarVagonesVacíos que reciba un número de tren y extraiga del mismo los vagones que están totalmente vacíos. Este método debe retornar cuántos vagones fueron eliminados. Si el tren no existe debe devolver -1.

Realizá los métodos (tanto privados como públicos) que consideres necesarios para mantener el correcto encapsulamiento y responsabilidades de cada una de las clases del sistema.

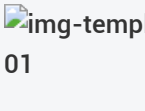
Se deben probar los métodos públicos desde una clase Test, sin pedir ingresos por teclado con el siguiente lote de prueba. Tené en cuenta que en esta clase no se crean trenes ni vagones:

- Procesá la creación de dos formaciones de trenes (tren1 y tren2) guardándote el número de tren de cada uno.
- Procesá el agregado de 5 vagones MEDIUM y 7 LARGE al primero de los trenes.
- Procesá el agregado de 20 vagones SMALL y otros 20 MEDIUM al segundo de los trenes.
- Procesá el agregado de 20 vagones LARGE al tren 999 (que no existe).
- Procesá la carga de 165 toneladas de grano en el primer tren.
- Procesá la carga de otras 200 toneladas de grano en el primer tren.
- Procesá la carga de 240 toneladas de grano en el segundo tren.
- Procesá la carga de otras 1000 toneladas de grano en el segundo tren.
- Procesá la carga de 10 toneladas de grano en el tren número 999.
- Listá la capacidad disponible (porcentaje) en todos los trenes.
- Procesá la extracción de los vagones vacíos del primer tren.
- Procesá otra vez la extracción de los vagones vacíos del primer tren.
- Procesá la extracción de los vagones vacíos del tren número 999.

Nota: Para desarrollar estos puntos con mayor facilidad y rapidez te recomendamos modularizar la clase Test. Hacé clic en el botón para ver una posible salida por pantalla, habiendo realizado el ejercicio completo y procesando los datos pedidos anteriormente:

Ver el código:

Claves de autocorrección



Descargá los archivos de la carpeta "Posibles soluciones / TREN" para acceder a las claves de autocorrección.