UTN
Facultad Regional Córdoba

TUP

Tecnicatura Universitaria en Programación

PROGRAMACIÓN I

Unidad Temática N°2:
Programación de Objetos con C#

Guía de Estudio
1° Año – 1° Cuatrimestre

Índice

PROGRAMACIÓN DE OBJETOS CON C#

Basado en el Paradigma Orientado a Objetos y utilizando modelado UML, diseñe una solución escrita en lenguaje de programación C# para cada uno de los problemas que se plantean a continuación.

En cada caso defina:

- Clase que incluya atributos, propiedades, constructor sin parámetros, constructor con parámetros, métodos de control y método ToString().
- Método Main() o punto de entrada que implemente la solución.

Problema 2.1:

Rectángulo. Calcular el perímetro y la superficie de un rectángulo de base 3 y alto 5.

Problema 2.2:

Silo. El dueño de un campo almacena sus cosechas de granos en silos cilíndricos. Le pide que desarrolle un programa que le permita conocer el volumen máximo de un silo.

Problema 2.3:

Punto. nos han solicitado desarrollar una clase que permita modelar un punto en el plano. Cada punto se representa mediante un par ordenado de coordenadas cartesianas y es posible conocer:

- cuál es su distancia al eje de coordenadas
- Adicionalmente se necesita mostrar un punto como una cadena con formato “(x;y)”.

Problema 2.4:

Persona. Se nos pide modelar una clase Persona con las siguientes condiciones:

- Sus atributos son: nombre, edad, sexo (H hombre, M mujer), peso y altura.
- Los métodos que se implementarán son:
  - calcularIMC(): calculará si la persona está en su peso ideal (peso en kg/(altura^2 en m)), si esta fórmula devuelve un valor menor que 20, la función devuelve un -1, si devuelve un número entre 20 y 25 (incluidos), significa que está por debajo de su peso ideal la función devuelve un 0

y si devuelve un valor mayor que 25 significa que tiene sobrepeso, la función devuelve un 1. Se recomienda usar constantes para devolver estos valores.

- esMayorDeEdad(): indica si es mayor de edad(21 años), devuelve un booleano.

Problema 2.5:

Password. Modelar una clase llamada Password que tenga los atributos longitud y valor. Por defecto, la longitud será de 8.

Se pide:

- esFuerte(): devuelve un booleano si es fuerte o no, para que sea fuerte debe tener mas de 2 mayúsculas, mas de 1 minúscula y mas de 5 números.
- generarPassword(): genera la contraseña del objeto con la longitud que tenga.

Problema 2.6

Auto. Diseñar una clase que permita modelar el funcionamiento básico de un automóvil, tal como se indica a continuación:

- **conducir**: esta opción debe recibir la cantidad de kilómetros a recorrer e informar si pueden ser recorridos y en tal caso descontar el combustible utilizado del tanque (se conoce que, por especificaciones técnicas del vehículo, con un litro recorre 11 km). Caso contrario informar que no hay combustible suficiente para el recorrido indicado.
- **cargar combustible**: está opción debe reponer combustible en el tanque tantos litros como se reciban como argumento pero teniendo en cuenta que el tanque tiene una capacidad de 49 lts más una reserva de 5 lts., en el caso de que los litros informados superen la capacidad del tanque, devolver la cantidad de lts de combustible derramados.

ARREGLOS

Utilizando arreglos tipo listas o tablas como soporte de una colección de datos, implemente una solución en C# para cada uno de los problemas que se plantean a continuación.

Lista o vector.

Problema 2.7:

Capturar en una lista los sueldos de 5 empleados y desplegarlos en una segunda lista aumentados en un 30%.

Problema 2.8:

Capturar los datos de 5 productos comprados en una tienda, incluyendo nombre, precio y cantidad en sus 3 listas respectivas, después calcular una cuarta lista con el gasto total por cada producto desplegarlo todo en un segundo panel e incluir también el gran total.

Tabla o Matriz.

Problema 2.9:

Construir una tabla o cuadro que contenga los ingresos mensuales por ventas de productos durante los tres primeros meses del año de cuatro sucursales de una cadena de supermercados, agregar al final una lista que muestre los ingresos mensuales totales por cada mes y una segunda lista que muestre los ingresos mensuales totales por cada sucursal.

RELACIONES DE ASOCIACIÓN Y DEPENDENCIA

Problema 2.10:

Realizar un programa que permita modelar el funcionamiento básico de un automóvil, tal como se indica a continuación:

- **conducir**: esta opción debe recibir la cantidad de kilómetros a recorrer e informar si pueden ser recorridos y en tal caso descontar el combustible utilizado del tanque (se conoce que, por especificaciones técnicas del vehículo, con un litro recorre 11 km). Caso contrario informar que no hay combustible suficiente para el recorrido indicado.
- **cargar combustible**: está opción debe reponer combustible en el tanque tantos litros como se reciban como argumento pero teniendo en cuenta que el tanque tiene una capacidad de 49 lts más una reserva de 5 lts., en el caso de que los litros informados superen la capacidad del tanque, devolver la cantidad de lts de combustible derramados.
- **chequear nivel de combustible**: informar si el tanque está al 25%, 50%, 75% o lleno y dar una alerta en el caso que el tanque se encuentre en reserva, indicando que hay que reponer combustible de inmediato. El mensaje debe devolver una cadena con 25% ... o el alerta.

Ayuda: para una resolución más sencilla, pensar en la existencia de dos clases: Automóvil y Tanque.

Problema 2.11:

Vamos a desarrollar una baraja de cartas españolas orientado a objetos.

Una carta tiene un número entre 1 y 12 (el 8 y el 9 no los incluimos) y un palo (espadas, bastos, oros y copas)

La baraja estará compuesta por un conjunto de cartas, 40 exactamente.

Las operaciones que podrá realizar la baraja son:

- **barajar**: cambia de posición todas las cartas aleatoriamente
- **siguienteCarta**: devuelve la siguiente carta que está en la baraja, cuando no haya más o se haya llegado al final, se indica al usuario que no hay más cartas.
- **cartasDisponibles**: indica el número de cartas que aún puede repartir
- **darCartas**: dado un número de cartas que nos pidan, le devolveremos ese número de cartas (piensa que puedes devolver). En caso de que haya menos cartas que las pedidas, no devolveremos nada pero debemos indicárselo al usuario.
- **cartasMonton**: mostramos aquellas cartas que ya han salido, si no ha salido ninguna indicárselo al usuario
- **mostrarBaraja**: muestra todas las cartas hasta el final. Es decir, si se saca una carta y luego se llama al método, este no mostrara esa primera carta.