Unidad

1

DIPLOMATURA EN PROGRAMACION .NET

Ejercicios

Capítulo 1

Introducción

Capítulo 1

Ejercicio 1

En este ejercicio se investigan las variables de referencia, la creación de objetos y la asignación de variables de referencia

- Utilizando la clase que se le provee cuyo nombre es MiPunto, en la clase TesteoMiPunto en C# o en el Module1 de VB del proyecto ejercicio1 de la solución Capitulo1, programar la función Main la cual debe hacer lo siguiente:

  1. Declarar dos variables de tipo MiPunto llamadas comienzo y fin. Asignar a cada variable un objeto nuevo.
  2. Inicializar los valores x e y del objeto comienzo con el valor 10.
  3. Inicializar los valores x e y de fin con 20 y 30 respectivamente.
  4. Mostrar por consola los valores recientemente almacenados. Por ejemplo, para el valor x de comienzo la instrucción sería:

     ```
     Console.WriteLine("El valor x del comienzo es " + comienzo.x);
     ```

  5. Compilar y ejecutar el programa
  6. Declarar una nueva variable del tipo MiPunto y llamarla otroPunto
  7. Asignarle a otroPunto el contenido de la variable de referencia fin
  8. Mostrar por consola los valores x e y de fin y otroPunto
  9. Asignar nuevos valores a las variables x e y del objeto al que referencia otroPunto
  10. Mostrar por consola los valores x e y para comienzo, fin y otroPunto
  11. Modificar las salidas por pantalla para que se vean como se muestra a continuación:

      ```
      El punto de comienzo es [10,10]
      El punto de fin es [20,30]
      El otro punto es [20,30]
      El punto de fin es [20,30]
      El otro punto es [47,30]
      El punto de fin es [47,30]
      El punto de comienzo es [10,10]
      ```

Ejercicio 2

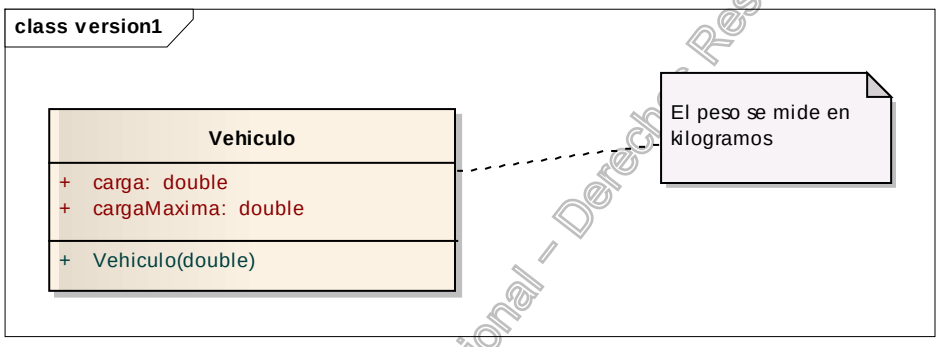
En este ejercicio se explora el propósito del encapsulado en los objetos. Se creará una clase en tres pasos para demostrar el uso del ocultamiento de la información.

Versión 1

Sin ocultamiento de la información

En esta versión de la clase Vehiculo se dejarán los atributos públicos para que el programa que prueba la clase los acceda directamente

1. Utilizar en el proyecto ejercicio2_version1 de la solución Capitulo1
2. Crear la clase Vehiculo que refleje el siguiente diagrama UML



3. Incluir dos atributos públicos: carga "el peso actual de carga del vehículo" y cargaMaxima "el peso máximo de carga del vehículo"
4. Incluir un constructor público para asignar el valor inicial a cargaMaxima

Nota: todos los pesos se asumen en kilogramos

5. Leer el código de la clase PruebaVehiculos. Notar que el programa se encuentra en problemas cuando se agrega la última caja con la que se carga el vehículo porque el código no verifica si al agregar una caja se supera la capacidad máxima.
6. Compilar y ejecutar las clases
7. Verificar que la salida sea igual que la que se muestra:

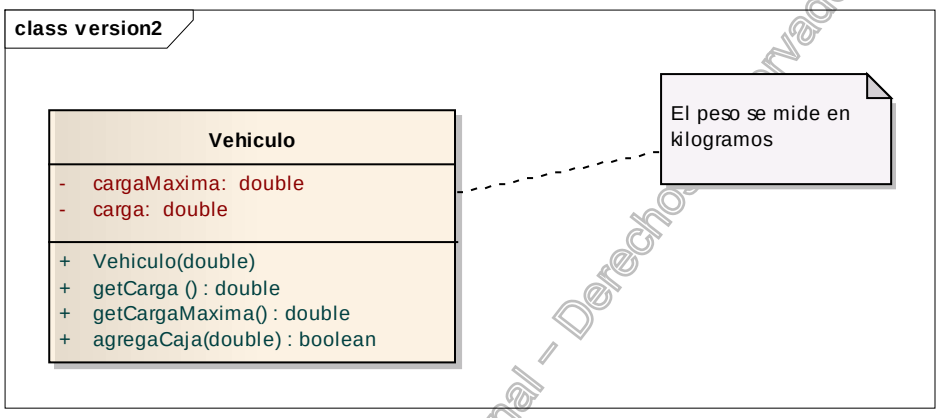
```
Creando un a vehículo con un máximo de 10,000kg de carga.
Agregar caja #1 (500kg)
Agregar caja #2 (250kg)
Agregar caja #3 (5000kg)
Agregar caja #4 (4000kg)
Agregar caja #5 (300kg)
La carga del vehículo es 10050.0 kg
```

Versión 2

Ocultamiento de la información básico

Para resolver el problema de la primera versión, se deberá ocultar los datos internos de la clase (carga y cargaMaxima) y proveer un método, agregaCaja, para realizar la verificación apropiada que no permite que se sobrecargue el vehículo.

1. Utilizar en el proyecto ejercicio2_version2 de la solución Capitulo1
2. Crear la clase Vehiculo que refleje el siguiente diagrama UML (puede copiar la clase del ejercicio anterior y modificarla)



3. Modificar los atributos carga y cargaMaxima para que sean privados

   Nota: los métodos getXXX representan los get de las propiedades

4. Agregar el método agregaCaja, el cual recibe un solo argumento, el cual es el peso de la caja en kilogramos. El método debe verificar que si se agrega la caja no supere la carga máxima. Si se excediera el peso, no se agrega la caja y se deberá retornar un valor falso. En caso contrario, se agregará el peso y se sumará a la carga total que soporta el vehículo y se retorna un valor verdadero. Sugerencia: necesitará utilizar una sentencia if, el formato básico de la misma es el siguiente:

```
if ( <expresionBooleana> ) {

  <sentencia>*

} else {

  <sentencia>*

}
```

   Nota: todos los pesos se asumen en kilogramos

5. Leer el código de la clase PruebaVehiculos. Notar que el programa no se agrega la última caja con la que se carga el vehículo porque supera la capacidad máxima.
6. Compilar y ejecutar las clases
7. Verificar que la salida sea igual que la que se muestra:

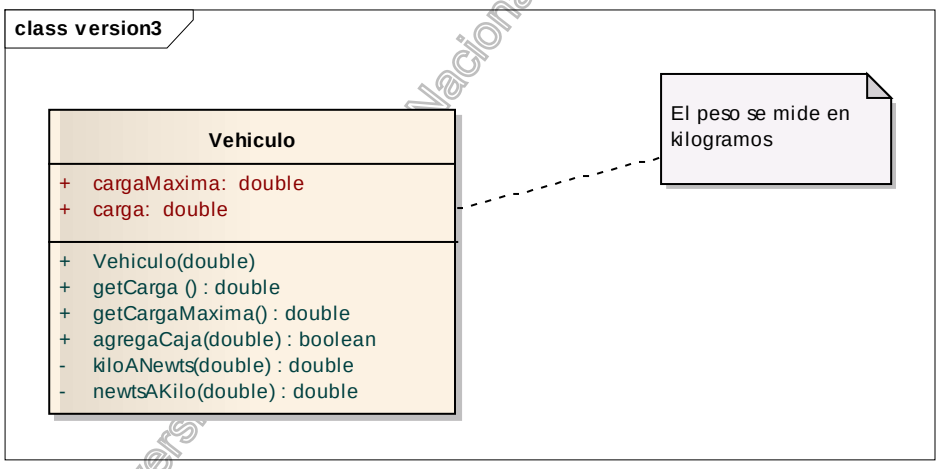
```
Creando un a vehículo con un máximo de 10,000kg de carga.
Agregar caja #1 (500kg)true
Agregar caja #2 (250kg)true
Agregar caja #3 (5000kg)true
Agregar caja #4 (4000kg)true
Agregar caja #5 (300kg)false
La carga del vehículo es 9750.0 kg
```

Versión 3

Cambiar la representación interna del peso a Newtons

En esta versión se debe suponer que se van a escribir ciertos cálculos para determinar el desgaste del motor y la estructura del vehículo. Estos cálculos son más fáciles de realizar si las medidas se hacen en Newtons

1. Utilizar en el proyecto Ejercicio2_3 de la solución Capitulo1
2. Crear la clase Vehiculo que refleje el siguiente diagrama UML (puede copiar la clase del ejercicio anterior y modificarla)



Nota: los métodos getXXX representan los get de las propiedades. La variable en mayúscula representa a una variable a la que no se le va a cambiar su valor

3. Modificar el constructor, las propiedades que poseen los métodos getCarga y getCargaMaxima junto con el método agregaCaja para que utilicen la conversión de kilogramos (la medida del peso en el parámetro) a Newtons (la medida de la variable de instancia. Utilice los siguientes métodos:

CS

```
private double kiloANewts(double peso)
{
```

```
            return (peso * KG_A_NEWTON);
        }
        private double newtsAKilo(double peso)
        {
            return (peso / KG_A_NEWTON);
        }
```

VB

```
    Private Function kiloANewts(ByVal peso As Double) As Double
        Return (peso * KG_A_NEWTON)
    End Function

    Private Function newtsAKilo(ByVal peso As Double) As Double
        Return (peso / KG_A_NEWTON)
    End Function
```

Notar que ahora todos los datos internos del vehículo deben estar en Newtons y los datos externos (los que retornan los métodos) deben estar en kilogramos.

4. Compilar y ejecutar las clases
5. Verificar que la salida sea igual que la que se muestra:

```
Creando un a vehículo con un máximo de 10,000kg de carga.
Agregar caja #1 (500kg)true
Agregar caja #2 (250kg)true
Agregar caja #3 (5000kg)true
Agregar caja #4 (4000kg)true
Agregar caja #5 (300kg)false
La carga del vehículo es 9750.0 kg
```

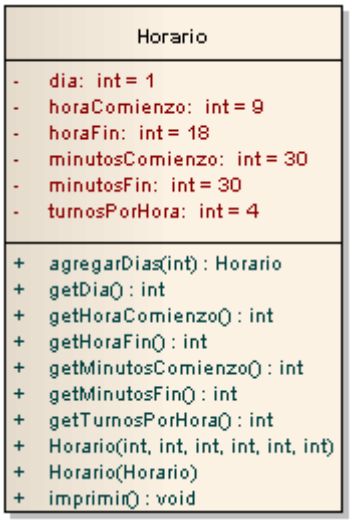
Notar que la salida del programa se mantiene. Esto demuestra que los cambios internos (privados) no afectan a las clases clientes.

Ejercicio 3

Este ejercicio introduce al proyecto Clínica, al cual se utilizará en capítulos futuros. El mismo consiste de una clínica con diferentes médicos y pacientes. Estas clases se reutilizarán en los sucesivos ejercicios

Se deberá crear una versión simple de la clase Horario la cual se colocará en el paquete utilidades. Un programa de prueba que se encuentra en la clase VerificaHorarios, se escribió en el espacio de nombres verificaciones para crear un único horario.

1. Utilizar en el proyecto Ejercicio3 de la solución Capitulo1
2. Crear la clase Horario que refleje el siguiente diagrama UML. Declarar cuando se cree la clase Horario se encuentra el espacio de nombres ejercicio3.Utilidades



Nota: los métodos getXXX representan los get de las propiedades. La variable en mayúscula representa a una variable a la que no se le va a cambiar su valor

3. Declarar como privado todos los atributos del diagrama UML.
4. Declara un constructor público que recibe parámetros para cada una de las variables de instancia.
5. Declarar métodos públicos para obtener y modificar cada valor en cada variable de instancia
6. Declarar un método público imprimir, que genere una salida por pantalla del valor que almacenan todas las variables cuando se lo invoca
7. Leer el código de la clase VerificaHorarios.
8. Compilar y ejecutar las clases
9. Verificar que la salida sea igual que la que se muestra:

```
Horario:
Día: 1
Hora de comienzo: 9
Minutos de comienzo: 30
Hora de fin: 18
Minutos de fin: 30
Día del horario: 1
Hora de comienzo del horario: 9
Minutos de comienzo del horario: 30
Hora de fin del horario: 18
Minutos de fin del horario: 30
Turnos por hora para el horario: 4
```