2018

Programación II

Parte II - v.1

Ing. David CECCHI

UTN

PROGRAMACIÓN ORIENTADA A OBJETOS

CONCEPTOS

Objeto: *es una entidad que tiene un conjunto de responsabilidades y que encapsula un estado interno*. Este conjunto de responsabilidades se implementa a través de métodos los cuales determinan el comportamiento del objeto, mientras que el estado interno se implementa a través de un conjunto de propiedades o atributos encapsulados. Por lo tanto todo objeto posee un estado, un comportamiento y una identidad, esta última es la característica que permite distinguir al objeto entre todos los demás.

Programa: *un programa es una colección de objetos cooperantes*, esta cooperación se lleva a cabo a través de mensajes.

Mensaje: *consiste en el nombre de una operación junto con los argumentos necesarios*. Es el medio por el cual los objetos se comunican con el fin de lograr una acción.

Abstracción: "*una abstracción denota las características esenciales de un objeto que lo distinguen de todas las otras clases de objetos y que por lo tanto proporcionan limites conceptuales bien definidos, con relación a la perspectiva del observador*". [Booch].
La abstracción es una de las formas fundamentales con la que los seres humanos se enfrentan a la complejidad. Por lo tanto la abstracción captura el comportamiento completo de un objeto.

Encapsulamiento: "*es el proceso de esconder todos los detalles de un objeto que no contribuyen a sus características esenciales*". [Booch].
Un objeto pone a disposición del los objetos clientes lo que es capaz de realizar, pero nunca informa o les permite saber como lleva a cabo dichas tareas.
Esta característica es de suma importancia ya que asegura que la implementación en los objetos clientes es independiente de la implementación en el objeto servidor. Por lo tanto, los algoritmos contenidos en los métodos pueden ser modificados (perfeccionados) con tranquilidad sabiendo que no se requerirán cambios en los objetos clientes, siempre que no se haya modificado la interfaz del servidor. Pero no solo se trata de impedir que el cliente conozca los detalles del servidor, sino algo más importante aún que es evitar que el cliente se vea *forzado* a conocerlos.
La abstracción y el encapsulamiento son conceptos complementarios, ya que la abstracción se concentra en la visión exterior del objeto, mientras que el encapsulamiento evita que los objetos clientes accedan a su visión interna donde el comportamiento de la abstracción es implementado.

Clase: *se la puede considerar como el molde para un tipo determinado de objetos*. Un objeto es una instancia de la clase a la que pertenece, y todo objeto es instancia de alguna clase.

RELACIONES ENTRE CLASES

Relación de Herencia: *es un mecanismo que acepta que la definición de una clase incluya el comportamiento y la estructura interna definidos en otra clase más general*. Esto posibilita la reutilización de código.
Conceptos que se desprenden de este tipo de relaciones son:
SuperClase: es una clase de la cual otras clases heredan su comportamiento y estructura interna. Una clase puede tener una única superclase (herencia simple) o varias (herencia múltiple).
SubClase: es una clase que hereda el comportamiento y estructura interna de otra clase (SuperClase). La SubClase agregara su propio comportamiento y estructura interna para definir su propio y único tipo de objeto.
Redefinición: una subclase puede redefinir un método heredado simplemente definiendo un método con el mismo nombre. De esta forma, el método de la subclase reemplazara al método heredado. Así se podrá especificar un comportamiento mas adecuado para la subclase a través de un algoritmo más eficiente desde su punto de vista.
Clase Abstracta: es una clase que no ha sido creada para producir instancias de si misma. Este tipo de clases son creadas con la finalidad de agrupar comportamiento y estructura interna común a varias subclases.

El camino que se sigue para poder responder un mensaje es el siguiente:
1- El objeto recibe un mensaje.

2- Se busca el mensaje en la clase del objeto receptor.
3- Si el mensaje es encontrado, se ejecuta el método correspondiente.
4- Si el mensaje no fue encontrado, se lo busca en la superclase del objeto receptor. Si no se lo encuentra, se lo buscara en la superclase de su superclase, y así sucesivamente.
5- Si finalmente el mensaje no fue encontrado, se da un mensaje de error ya que el objeto no fue capaz de entender el mensaje.

Con la ayuda de la herencia el encapsulamiento puede ser violado ya que: la subclase puede acceder a un atributo de la superclase, llamar un método privado de la superclase, o referirse directamente a la superclase de su superclase.
Por lo tanto, en estos casos, un cambio en una superclase podría afectar a todas sus subclases.

Relación de Ensamble: *"es la relación "es parte de" en la cual los objetos que representan los componentes de algún conjunto son asociados a un objeto que representa el todo*". [Rumbaugh]. Es decir, un objeto compuesto esta formado por objetos componentes que son sus partes. Cada objeto parte tiene su propio comportamiento y el objeto compuesto es tratado como una unidad cuya responsabilidad es realizar la coordinación de las partes. Además, las partes pueden o no existir fuera del objeto compuesto, o pueden aparecer en varios objetos compuestos.
En síntesis, ensamble involucra dos objetos distintos donde uno es parte del otro.

Relación de Asociación: *se presenta cuando un objeto para llevar a cabo su comportamiento necesita un servicio provisto por otro objeto*.

Polimorfismo: *es la habilidad de dos o más objetos de responder a un mensaje con el mismo nombre, cada uno de su propia forma*. Esto significa que un objeto no necesita saber a quien le esta enviando un mensaje.

Binding Dinámico: significa que el enlace entre el mensaje y el método del receptor se realiza en tiempo de ejecución. Sin Binding Dinámico no seria posible implementar el polimorfismo.

Bibliografía:
Tecnología de Orientación a Objetos – Luciano Ripani – UTN – FRRo

EJEMPLO INTRODUCTORIO

A continuación se presenta un ejemplo introductorio a la POO, el cual se encuentra constituido por 2 clases: la clase Camión y la clase Caja. Desde el programa principal, el usuario contará con la posibilidad de agregar y quitar cajas al camión, además de listar las cajas contenidas.
Las cajas se encuentran almacenadas dentro de la propiedad P_Mercaderia la cual se corresponde con una colección List donde cada objeto contenido por la misma es una instancia de la clase Caja.
La propiedad P_Mercaderia es privada en la clase, de esta forma se impide el acceso directo a los métodos inherentes a toda colección List. El usuario de una instancia Camión, para manipular la List referida a las mercaderías del vehiculo, deberá limitarse al uso de los métodos y propiedades públicos definidos en la clase. Los métodos registrados con esta finalidad son: Agregar, Quitar y Recuperar. Además contará con 3 propiedades de solo lectura para obtener información relacionada a la mercadería cargada: Cantidad de Cajas Cargadas, Peso de la Carga y Peso Disponible para Carga.
La explicación detallada del ejercicio y conceptos aplicados, serán presentados en el aula. Queda en el lector ampliar la funcionalidad de las clases presentadas, agregando nuevos métodos o redefiniendo los ya existentes.

Clase Caja

```
class ClaseCaja
{
   private string P_CodigoInterno;
   private string P_Contenido;
   private float P_AltoCM;
   private float P_LargoCM;
   private float P_AnchoCM;
   private float P_PesoKG;
   private string P_Material;

   public ClaseCaja()
   {
      //Constructor
      this.CodigoInterno = "";
      this.Contenido = "";
      this.AltoCM = 0;
      this.LargoCM = 0;
      this.AnchoCM = 0;
      this.PesoKG = 0;
      this.Material = "";
   }

   public ClaseCaja(string N_CodigoInterno,string N_Contenido, float N_AltoCM, float N_LargoCM, float
N_AnchoCM, float N_PesoKG, string N_Material)
   {
      //Constructor
      this.CodigoInterno = N_CodigoInterno;
      this.Contenido = N_Contenido;
      this.AltoCM = N_AltoCM;
      this.LargoCM = N_LargoCM;
      this.AnchoCM = N_AnchoCM;
      this.PesoKG = N_PesoKG;
      this.Material = N_Material;
   }

   public string CodigoInterno
   {
      get
      {
         return P_CodigoInterno;
```

```
    }
    set
    {
      P_CodigoInterno = value;
    }
  }


  public string Contenido
  {
    get
    {
      return P_Contenido;
    }
    set
    {
      P_Contenido = value;
    }
  }
  public float AltoCM
  {
    get
    {
      return P_AltoCM;
    }
    set
    {
      P_AltoCM = value;
    }
  }
  public float AnchoCM
  {
    get
    {
      return P_AnchoCM;
    }
    set
    {
      P_AnchoCM = value;
    }
  }
  public float LargoCM
  {
    get
    {
      return P_LargoCM;
    }
    set
    {
      P_LargoCM = value;
    }
  }
  public float PesoKG
  {
    get
    {
      return P_PesoKG;
```

```
        }
        set
        {
            P_PesoKG = value;
        }
    }
    public string Material
    {
        get
        {
            return P_Material;
        }
        set
        {
            P_Material = value;
        }
    }



    public float VolumenCM3
    {
        //propiedad de solo lectura
        get
        {
            return this.AltoCM * this.AnchoCM * this.LargoCM;
        }
    }

    ~ClaseCaja()
    {
        //destructor
    }

}
```

Clase Camion

```
class ClaseCamion
{
    private string P_Patente;
    private string P_Chofer;
    private float P_MaxCargaKG;
    private List<ClaseCaja> P_Mercaderia;

    public ClaseCamion()
    {
        //Constructor
        this.Patente = "";
        this.Chofer = "";
        this.MaxCargaKG = 0;
        this.P_Mercaderia = new List<ClaseCaja>();
    }

    public ClaseCamion(string N_Patente, string N_Chofer, float N_MaxCargaKG)
    {
        //Constructor
        this.Patente = N_Patente;
```

```
    this.Chofer = N_Chofer;
    this.MaxCargaKG = N_MaxCargaKG;
    this.P_Mercaderia = new List<ClaseCaja>();
  }

  public string Patente
  {
    get
    {
      return P_Patente;
    }
    set
    {
      P_Patente = value;
    }
  }
  public string Chofer
  {
    get
    {
      return P_Chofer;
    }
    set
    {
      P_Chofer = value;
    }
  }




  public float MaxCargaKG
  {
    get
    {
      return P_MaxCargaKG;
    }
    set
    {
      P_MaxCargaKG = value;
    }
  }

  public int CantCajasCargadas
  {
    //propiedad de solo lectura
    get
    {
      return this.P_Mercaderia.Count;
    }
  }

  public float PesoKGMercaderia
  {
    //propiedad de solo lectura
    get
```

```
    {
      float TotalCargaKG;
      TotalCargaKG = 0;
      foreach (ClaseCaja CajaAux in this.P_Mercaderia)
      {
        TotalCargaKG = TotalCargaKG + CajaAux.PesoKG;
      }
      return TotalCargaKG;
    }
  }

  public float PesoKGDisponible
  {
    //propiedad de solo lectura
    get
    {
      return this.MaxCargaKG - this.PesoKGMercaderia;
    }
  }

  public bool AgregarCaja(ClaseCaja N_Caja)
  {
    if (N_Caja.PesoKG <= this.PesoKGDisponible)
    {
      this.P_Mercaderia.Add(N_Caja);
      return true;
    }
    else
    {
      return false;
    }
  }
  public bool QuitarCaja(int PosCaja)
  {
    if (PosCaja >= 0 && PosCaja < this.CantCajasCargadas)
    {
      this.P_Mercaderia.RemoveAt(PosCaja);
      return true;
    }
    else
    {
      return false;
    }
  }
  public bool RecuperarDatosCaja(int PosCaja, out ClaseCaja CajaRecuperada)
  {
    CajaRecuperada = null;
    if (PosCaja >= 0 && PosCaja < this.CantCajasCargadas)
    {
      CajaRecuperada = this.P_Mercaderia[PosCaja];
      return true;
    }
    else
    {
      return false;
    }
  }
```

```
   ~ClaseCamion()
   {
      //destructor
   }
}
```

Programa Principal

```
class Program
{
   static void Main(string[] args)
   {
      char Opcion;
      ClaseCamion MiCamion = new ClaseCamion("HFS-521", "Gomez, Jorge", 20000);

      Console.Clear();
      Console.WriteLine("Ingrese Opcion: \n(A)-Agregar Mercaderia \n(Q)-Quitar Mercaderia \n(L)-Listar Mercaderia
\n(S)-Salir");
      Opcion = char.Parse(Console.ReadKey().KeyChar.ToString().ToUpper());
      while (Opcion == 'A' || Opcion == 'Q' || Opcion == 'L')
      {
         switch(Opcion)
         {
            case 'A':
               {
                  Agregar(MiCamion);
                  break;
               }
            case 'Q':
               {
                  Quitar(MiCamion);
                  break;
               }
            case 'L':
               {
                  Listar(MiCamion);
                  break;
               }
         }
         Console.Clear();
         Console.WriteLine("Ingrese Opcion: \n(A)-Agregar Mercaderia \n(Q)-Quitar Mercaderia \n(L)-Listar
Mercaderia \n(S)-Salir");
         Opcion = char.Parse(Console.ReadKey().KeyChar.ToString().ToUpper());
      }
   }
```

```
   private static void Agregar(ClaseCamion Camion)
```

```
{
    string Cadena;
    string Codigo;
    string Contenido;
    float Alto;
    float Largo;
    float Ancho;
    float Peso;
    string Material;
    ClaseCaja NuevaCaja;

    Console.Clear();
    Console.WriteLine("Codigo: ");
    Codigo = Console.ReadLine();

    Console.WriteLine("Contenido: ");
    Contenido = Console.ReadLine();

    Console.WriteLine("Alto en CM: ");
    Cadena = Console.ReadLine();
    while (float.TryParse(Cadena, out Alto) == false)
    {
        Console.WriteLine("Alto en CM: ");
        Cadena = Console.ReadLine();
    }

    Console.WriteLine("Largo en CM: ");
    Cadena = Console.ReadLine();
    while (float.TryParse(Cadena, out Largo) == false)
    {
        Console.WriteLine("Largo en CM: ");
        Cadena = Console.ReadLine();
    }

    Console.WriteLine("Ancho en CM: ");
    Cadena = Console.ReadLine();
    while (float.TryParse(Cadena, out Ancho) == false)
    {
        Console.WriteLine("Ancho en CM: ");
        Cadena = Console.ReadLine();
    }

    Console.WriteLine("Peso en KG: ");
    Cadena = Console.ReadLine();
    while (float.TryParse(Cadena, out Peso) == false)
    {
        Console.WriteLine("Peso en KG: ");
        Cadena = Console.ReadLine();
    }

    Console.WriteLine("Material: ");
    Material = Console.ReadLine();

    NuevaCaja=new ClaseCaja(Codigo,Contenido,Alto,Largo,Ancho,Peso,Material);
    if (Camion.AgregarCaja(NuevaCaja))
    {
        Console.WriteLine("Caja agregada con exito!");
```

```
        }
        else
        {
            Console.WriteLine("Imposible agregar caja: \nse superaria el peso maximo permitido por el vehiculo.");
        }
        Console.ReadKey();
    }



    private static void Quitar(ClaseCamion Camion)
    {
        int Posicion;
        string Cadena;

        Console.Clear();
        Console.WriteLine("Posicion de la caja a quitar: ");
        Cadena = Console.ReadLine();
        while (int.TryParse(Cadena,out Posicion) == false)
        {
            Console.WriteLine("Posicion de la caja a quitar: ");
            Cadena = Console.ReadLine();
        }

        if (Camion.QuitarCaja(Posicion))
        {
            Console.WriteLine("Caja quitada con exito!");
        }
        else
        {
            Console.WriteLine("Imposible quitar caja: la posicion ingresada no es valida.");
        }
        Console.ReadKey();
    }

    private static void Listar(ClaseCamion Camion)
    {
        int CantCajas;
        ClaseCaja CajaRecup;

        Console.Clear();
        Console.WriteLine("----------------------- LISTADO DE CAJAS CONTENIDAS -----------------------");
        Console.WriteLine("PATENTE: {0} CHOFER: {1} MAX. CARGA: {2} LIBRES:
{3}",Camion.Patente,Camion.Chofer,Camion.MaxCargaKG,Camion.PesoKGDisponible);
        Console.WriteLine("--------------------------------------------------------------------------------");

        CantCajas = Camion.CantCajasCargadas;
        for (int Posicion = 0; Posicion < CantCajas; Posicion++)
        {
            Camion.RecuperarDatosCaja(Posicion, out CajaRecup);
            Console.WriteLine("Posicion: {0}", Posicion);
            Console.WriteLine("Codigo Interno: {0}", CajaRecup.CodigoInterno);
            Console.WriteLine("Contenido: {0}", CajaRecup.Contenido);
            Console.WriteLine("Peso: {0}", CajaRecup.PesoKG);
            Console.WriteLine("Volumen: {0}", CajaRecup.VolumenCM3);
            Console.WriteLine("--------------------------------------------------------------------------------");
        }
```

```
        Console.ReadKey();
    }
}
```