Unidad 2

Fundamentos de los Lenguajes

Capítulo 2

Orientación a Objetos en .Net

Contenidos

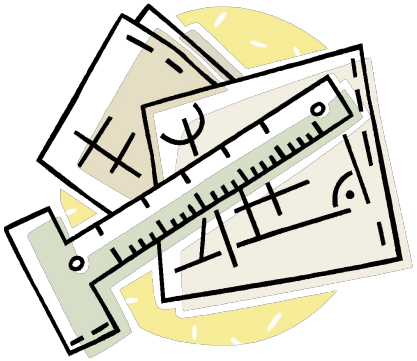
- Conceptos fundamentales del paradigma de OO
- Clases y Objetos
- Constructores y Destructores
- Métodos y Atributos
- Abstracción y Encapsulamiento
- Herencia Simple y Múltiple e Interfaces
- Polimorfismo
- Ocultamiento
- Clases Abstractas y Métodos Virtuales
- Clases Parciales (Partial Classes)
- Diseñador de Clases (Class Designer)

¿Qué es la Programación Orientada a Objetos?

- Es una manera de construir Software basada en el paradigma de orientación a objetos.
- Propone resolver problemas de la realidad a través de identificar objetos y relaciones de colaboración entre ellos.
- El Objeto y el mensaje son sus elementos fundamentales.

Clases

- Implica clasificación en base a comportamiento y atributos comunes
- Crea un vocabulario
  - La forma en que nos comunicamos
  - La forma en que nos expresamos
- Es una construcción estática
- Describe
  - Comportamiento común
  - Atributos (estado)
- Incluye
  - Datos
  - Funciones (métodos)



Constructor y Destructor

- Dos métodos de las clases, existen por defecto
- Constructor, inicializa valores
- Destructor, libera recursos al finalizar la vida de una instancia de una clase creada en memoria
- (en .Net) Existen constructores y destructores por defecto

¿Qué es un objeto?

- Instancia de una clase
- Un objeto posee:
  - Identidad: Relación única entre el objeto del modelo y el ente de la realidad que representa. Se implementa a través de un id único en el modelo.
  - Comportamiento: Resuelve un conjunto particular de problemas.
  - Estado: Almacena información
    - Fija
    - Variable

Pilares de POO

Herencia

Polimorfismo

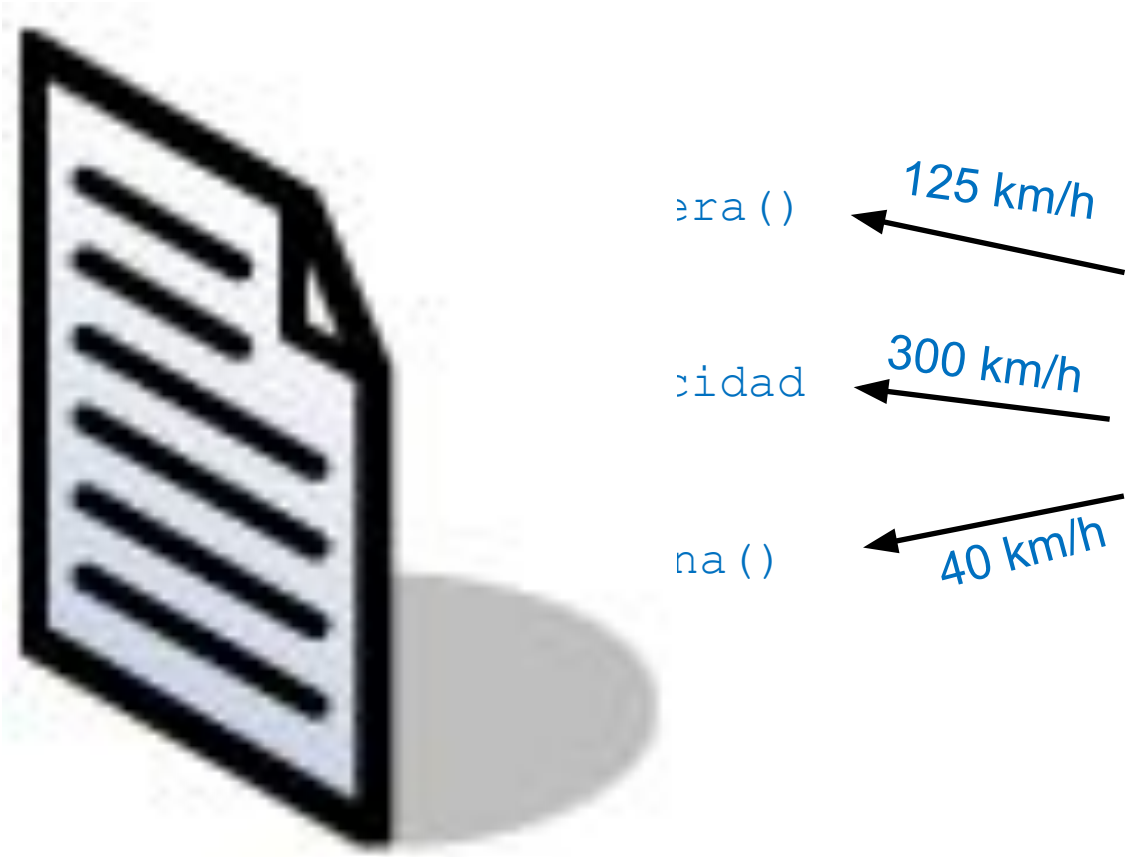
Encapsulamiento

Abstracción

Abstracción

- Ignorancia selectiva
- Decide que es importante y que no lo es
- Se enfoca [depende] en lo que es importante
- Ignora [no depende] de lo que no es importante
- Utiliza la encapsulación para reforzar la abstracción

Encapsulamiento

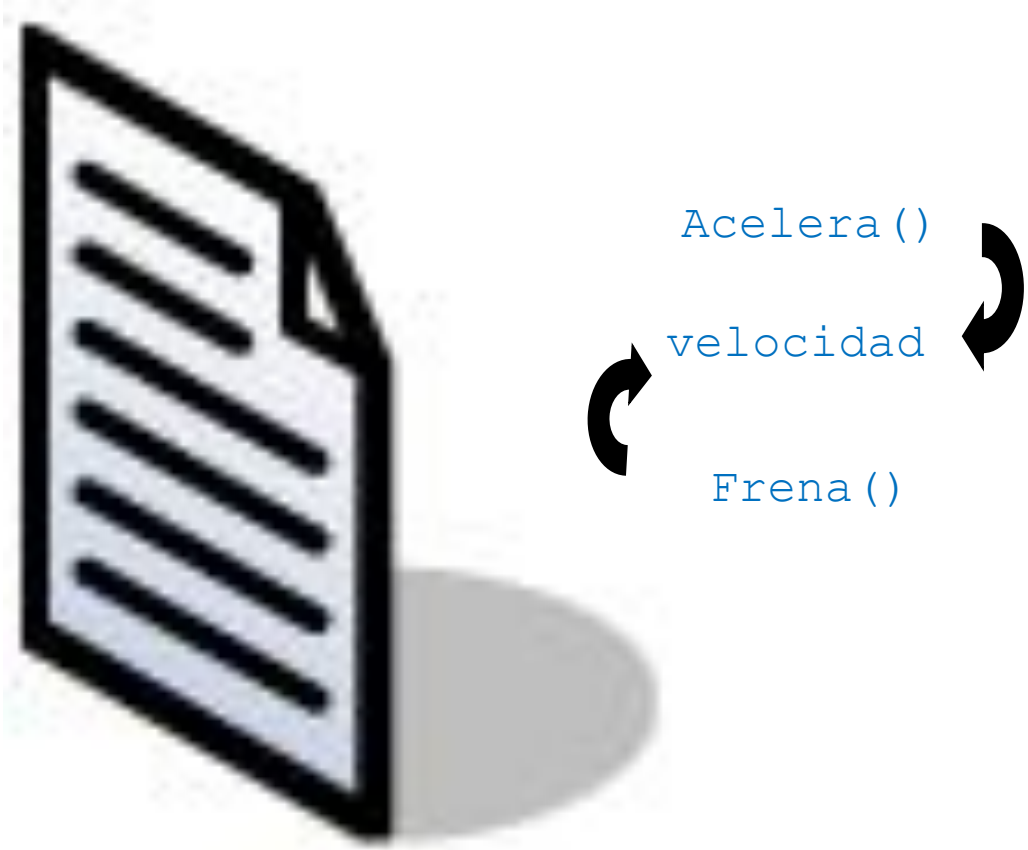


¿Por qué usar Encapsulamiento?

- Control
- Cambios

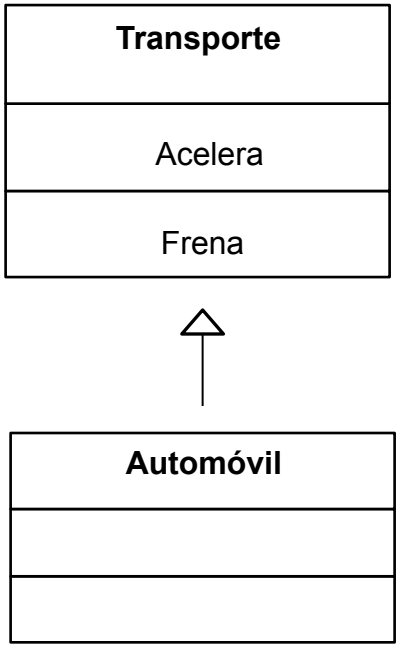
Encapsulamiento

- Métodos públicos: accesibles desde afuera
- Métodos privados: accesibles desde adentro

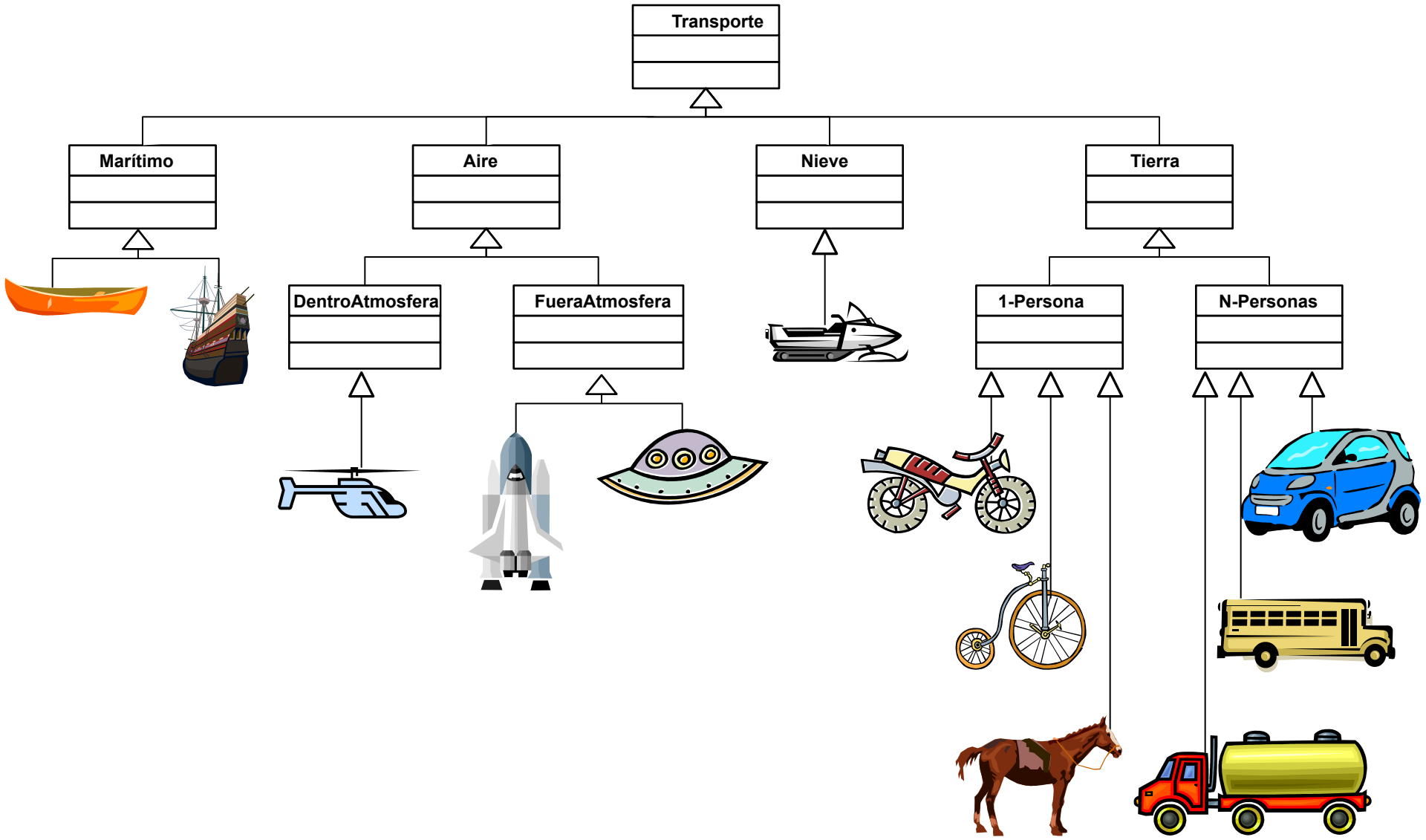


Herencia

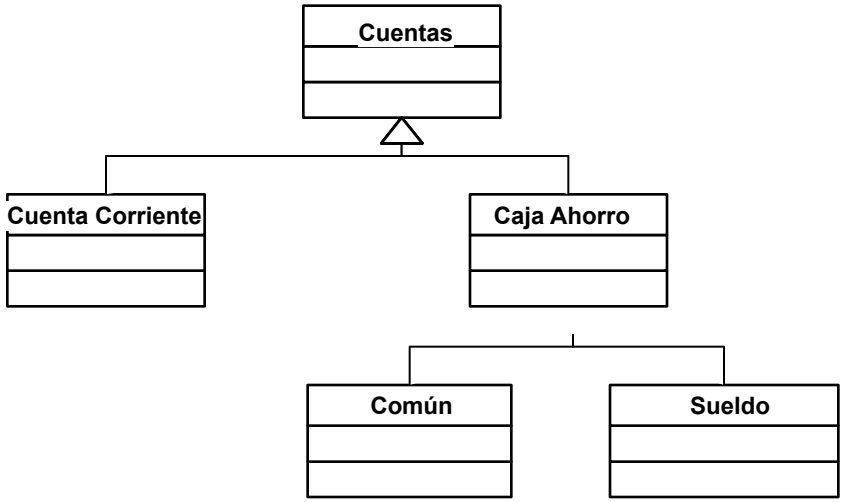
- Es una relación “un tipo de” entre clases
- Va de la generalización a la especialización
- Clase Base / Clase Derivada
- Hereda la implementación



Jerarquías de Clases – Ejemplo I

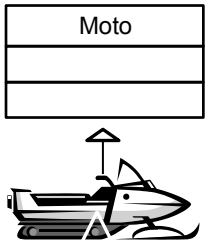


Jerarquía de Clases – Ejemplo II

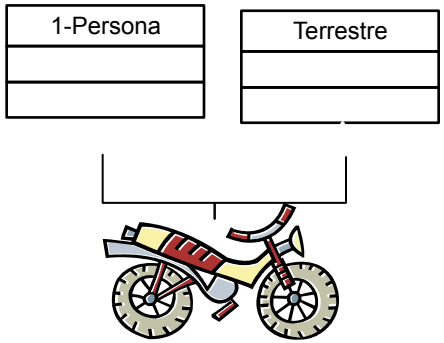


Herencia Simple y Múltiple

- Simple: La clase hija deriva de una única clase padre

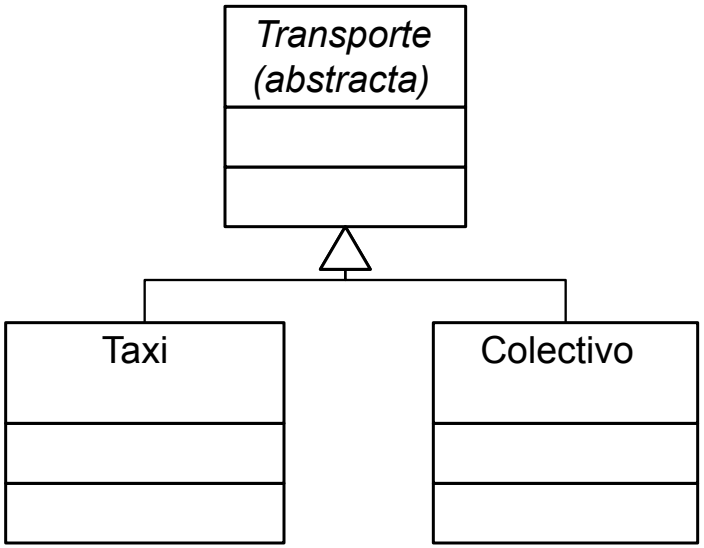


- Múltiple: La clase hija deriva de varias clases padre
  - No es soportada por todos los lenguajes
  - Puede ser confusa



Clases Abstractas

- Proveen una implementación parcial para que sea heredada por las clases derivadas
- No pueden ser instanciadas



Métodos Abstractos

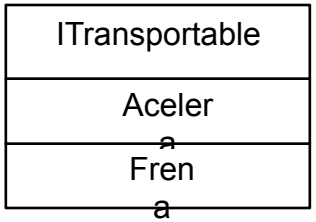
- Solo en clases abstractas
- No pueden contener implementación
- Deben ser implementados por las clases derivadas
- Los métodos abstractos son virtuales
- Los métodos abstractos pueden sobrescribir métodos de la clase base declarados como virtuales
- Los métodos abstractos pueden sobrescribir métodos de la clase base declarados como “override”

Métodos Virtuales

- Es un método que la clase base permite que sea sobrescrito en una clase derivada
- Un método no-virtual es la UNICA implementación posible para este método

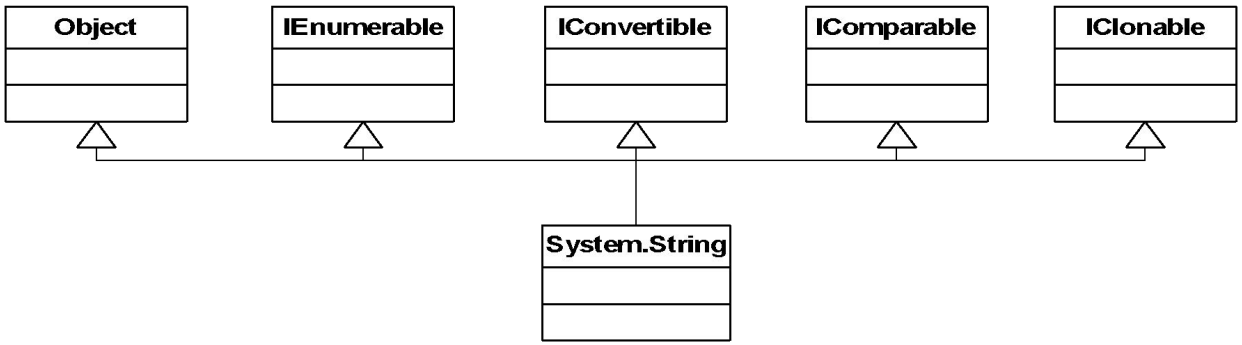
Interfases

- Definen un “contrato”
- Contienen solo métodos sin implementación
- No heredan atributos
- No se pueden crear instancias de una “interfase”
- Las clases derivadas deben de implementar todas las operaciones heredadas



Interfases

- Una clase puede implementar cero, una o más interfases
- Deben de implementarse todos los métodos heredados por la interfase
- Las interfases a su vez pueden heredar de múltiples interfases

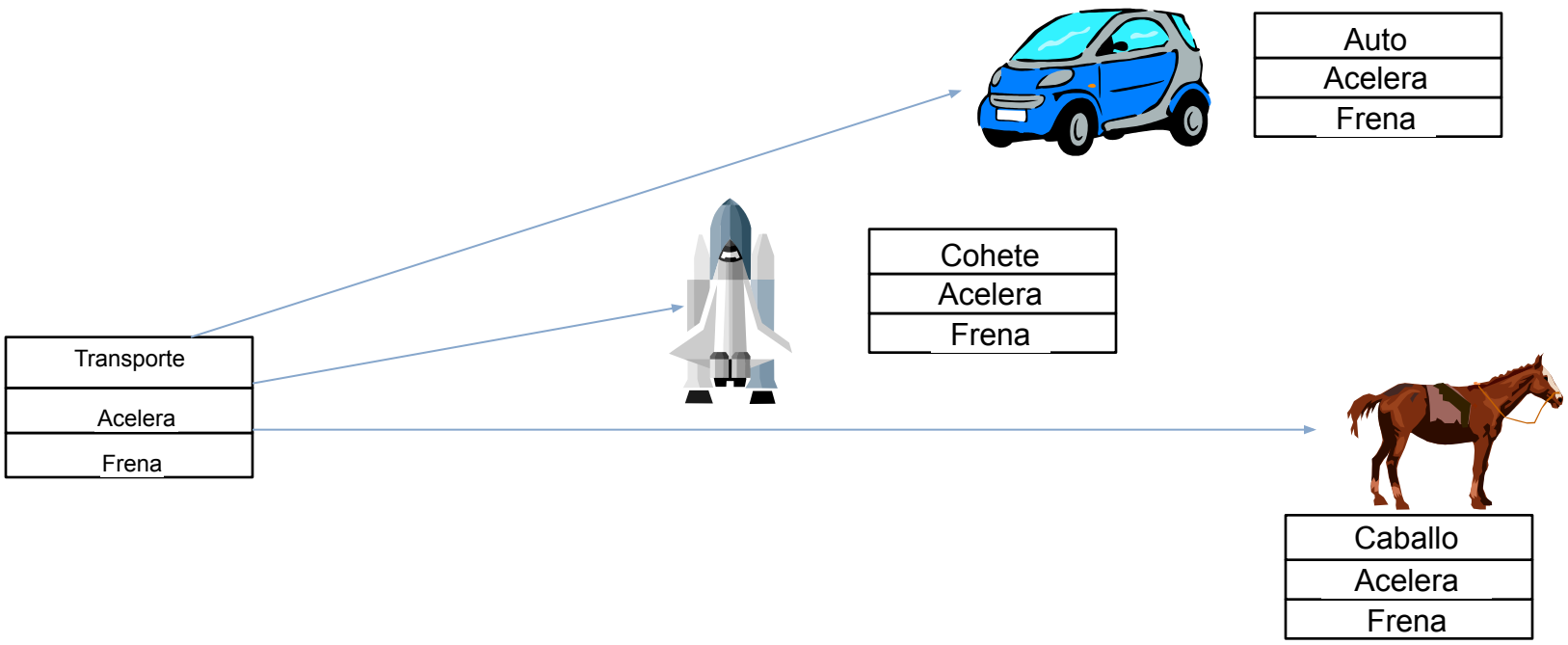


Polimorfismo - Definición

- Dos o más objetos son polimórficos con respecto a un conjunto de mensajes, si todos ellos pueden responder a esos mensajes, aún cuando cada uno lo haga de un modo diferente.

Polimorfismo

- La definición del método reside en la clase base
- La implementación del método reside en la clase derivada



Sobrecarga de Métodos

- Definir más de un método por cada mensaje, los tipos de los argumentos ayudan a decidir a qué mensaje se invoca
- Tareas similares son realizadas por métodos con mismo nombre
- Simplifican la tarea del desarrollador, al no tener que recordar distintos nombres para comportamientos iguales

Clases

- Clase: es la definición de las características de un determinado tipo de objeto.
- Son declaradas mediante la palabra class

```
//Definición de la clase CtaCte
class CtaCte
{
    //Definición de miembros
}
```

Constructores

- Constructor: funciones dentro de la clase, que son llamadas cuando se crea una instancia de dicha clase.

```
class CtaCte
{
    public CtaCte(){...} //Contructor por default
    public CtaCte(int i){...} //Constructor con un parametro
}
```

Propiedades

- Propiedad: característica o atributo de un objeto

```
class CtaCte
{
    int _balance;

    public int Balance
    {
        get
        {
            return _balance;
    }
        set
        {
            _balance = value;
        }
    }
}

CtaCte co = new CtaCte();
Co.Balance = 100; //Asignación
Mostrar(co.Balance) //Obtención
```

Métodos

- Métodos: acciones que un objeto puede llevar a cabo.
- Todo método es una función

```
public void HacerDeposito(int importe) //No devuelve valor
{
}

public int ObtenerInventario(int codArticulo) //Devuelve un entero
{
}
```

Sobrecarga de Métodos

- Sobrecarga: varios métodos con el mismo nombre pero diferentes parámetros.

```
public void HacerDeposito(int importe)
{
}

public void HacerDeposito(int importe, bool acreditar)
{
}
```

Namespaces

- Namespace: grupo de clases que tienen el mismo prefijo

```
namespace Banco
{
    namespace Gestion
    {
        public class CtaCte
        {
        }
        public class CajaAhorro
        {
        }
    }
}

//Referencia "full"
Banco.Gestion.CtaCte;
Banco.Gestion.CajaAhorro;

//Referencia "corta"
using Banco.Gestion;
CtaCte cc = new CtaCte();
CajaAhorro ca = new CajaAhorro();
```

Herencia

- Herencia: mecanismo por el cual una clase (hija) hereda de otra (padre) para extender su funcionalidad.

```
class Cuenta     //Clase Padre
{
}

class CtaCte : Cuenta     //Clase Hija
{
}
```

Herencia

- Dos keywords que afectan la “posiblidad” de heredar desde una clase base.

```
Public sealed class Cuenta
{
}

Public abstract class Cuenta
{
}
```

Invocando al Constructor Base

- El constructor “default” siempre invoca al constructor de la clase base

```
class MyBaseClass
{
    public MyBaseClass(int i)
    {
    }
    protected MyBaseClass(string s)
    {
    }
}

class MyDerivedClass: MyBaseClass
{
    public MyDerivedClass(int i) : base(i)
    {
    }
    public MyDerivedClass(): base(“Test”)
    {
    }
}
```

Protegiendo el Acceso a Miembros - C#

```
class MyBaseClass
{
    protected string field;
}

class MyDerivedClass: MyBaseClass
{}

class ThirdLevel : MyDerivedClass
{
    public string MyField()
    {
        return field;
    }
}
```

- “Publico” a las clases derivadas
- “Privado” a las clases externas
- No puede ser usado en estructuras

Interfaces: Implementación

- Declaradas mediante palabra interface

```
//Definición de la clase CtaCte
interface ICtaCte
{
    //Definición de miembros
}
```

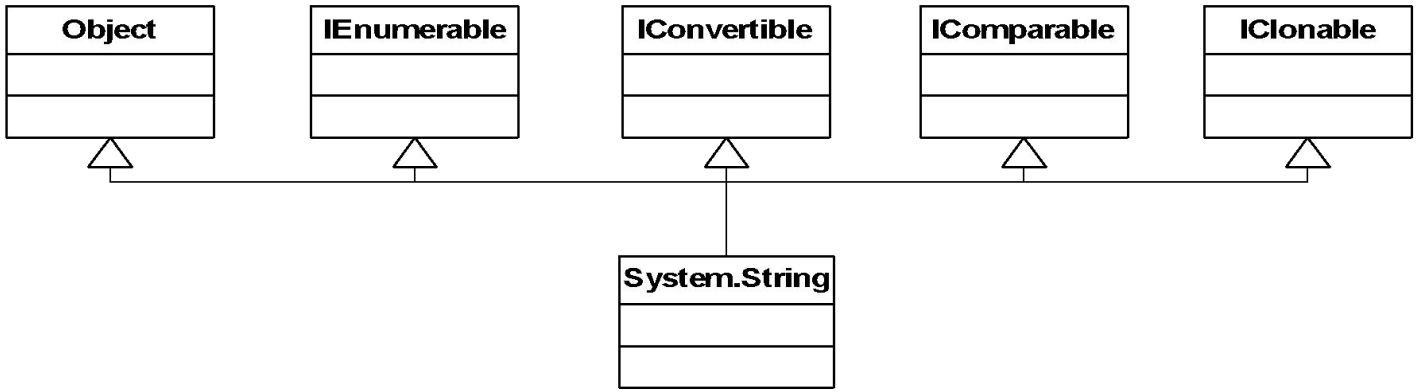
Interfaces

- Los métodos son implícitamente públicos
- Los métodos no tienen implementación
- No se declaran “access modifiers”
- Estándar Se les agrega el prefijo “I”

```
interface IMyInterface
{
    void MyMethod1();
    bool MyMethod2(string s);
}
```

Métodos de las Interfaces

- Una clase puede implementar cero, una o más interfaces
- Deben de implementarse todos los métodos heredados por la interface
- Las interfaces a su vez pueden heredar de múltiples interfaces



Implementando una Interface (implícitamente)

- Escribe el método exactamente de la misma forma que el método de la interfase
- Tienen el mismo: tipo de retorno, nombre y parámetros.
- El método implementado puede ser virtual o no virtual

```
interface IMyInterface
{
    void MyMethod1();
    bool MyMethod2(string s);
}

class MyClass: IMyInterface
{
    public virtual void MyMethod1() {}
    public bool MyMethod2(string s) {}
    public void OtherMethod() {}
}
```

Invocando métodos de una Interface (implícitamente)

- Puede ser invocada directamente a través de una clase o estructura.
- Puede realizarse un “cast” al tipo de la interfase.

```
MyClass mc = new MyClass();
mc.MyMethod1();
mc.OtherMethod();

IMyInterface mi = mc
bool b = mi.MyMethod2(“Hello”);
```

Implementando una Interface (explícitamente)

- Debe usarse el nombre completo del método para su acceso
- No puede ser declarada como virtual ni especificar un “access modifier”
- Solo puede ser accesado a través de la interfase

```
interface IMyInterface
{
    void MyMethod1();
    bool MyMethod2(string s);
}

class MyClass: IMyInterface
{
    void IMyInterface.MyMethod1() {}
    bool IMyInterface.MyMethod2(string s) {}
}
```

Invocando Métodos de una Interfase (explícitamente)

- No puede ser invocado de forma directa
- No es parte pública de una clase
- Debe ser aplicado un “cast” al tipo de la interfase y llamado desde el tipo de la interfase

```
MyClass mc = new MyClass();
//mc.MyMethod1(); 'mc.MyMethod1' does not exist in the current context.
IMyInterface mi = new MyClass();
mi.MyMethod1();
bool b = mi.MyMethod2(“Hello”);
```

Ventajas Implementación Explícita

- Permite que la implementación de la interfase sea excluida de la cara pública de la clase o estructura
- Resuelve conflictos de nombre

```
interface IMyInterface1
{
    void MyMethod();
}

interface IMyInterface2
{
    void MyMethod();
}

class MyClass: IMyInterface1, IMyInterface2
{
    void IMyInterface1.MyMethod() {}
    void IMyInterface2.MyMethod() {}
}
```

Clases Abstractas

- Proveen una implementación parcial para que sea heredada por las clases derivadas
- No pueden ser instanciadas
- Utiliza el calificador abstract

Clases Abstractas – C#

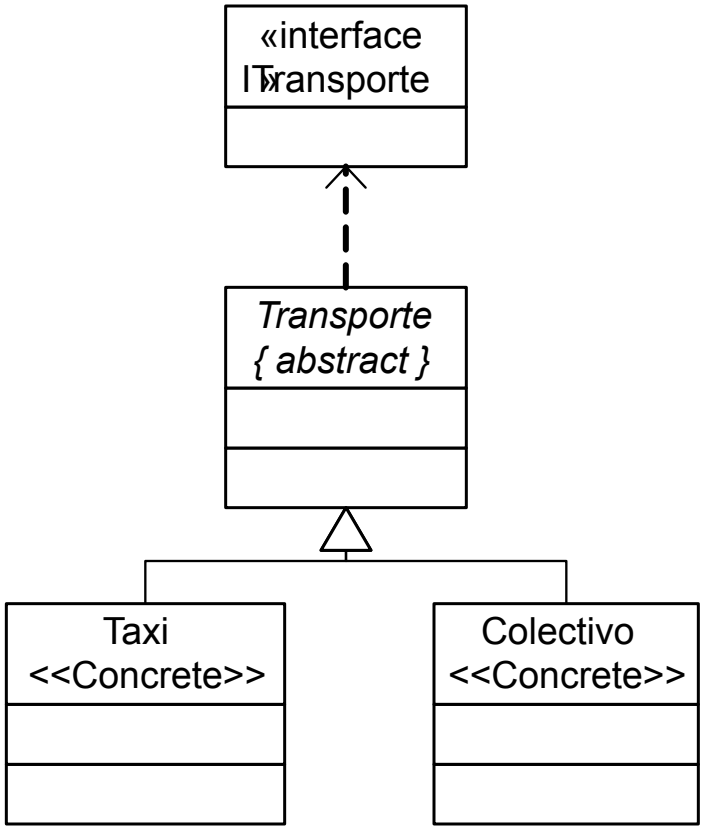
```
interface ITransporte
{
    string Name();
}

abstract class Transporte : ITransporte
{
    string Name()
    {...}
}

class Taxi: Transporte
{...}

class Colectivo: Transporte
{...}

}
```



Métodos Abstractos

- Solo en clases abstractas
- No pueden contener implementación
- Deben ser implementados por las clases derivadas
- Utiliza el calificador abstract
- Son virtuales
- Pueden sobrescribir métodos de la clase base declarados como virtuales
- Pueden sobrescribir métodos de la clase base declarados como “override”

Métodos Estáticos

- Miembros que no requieren de una instancia para ser invocados

```
public static void HacerDeposito(int importe)
{

}
```

Demostracion

Demostración 1

Ejemplos trabajando con objetos en C#

Laboratorios

Laboratorio 1

Clases y herencia

Metodos virtuales y ocultamiento

Clase Persona

Adivine el número

Adivine el número con ayuda