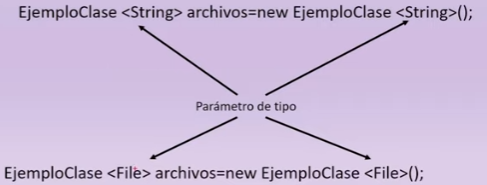
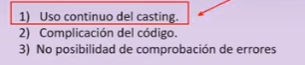
Programación genérica

Reutilización del código. La herencia también lo posibilita.

Se crean clases comodín que permiten usar cualquier clase de objeto, ya sean string, int, etc.



Se puede hacer lo mismo sin programación genérica, con la herencia. Tiene sus inconvenientes:



Lo del casting, el mas grave.

De esta forma, con genéricos, el código es mas sencillo, se reutiliza y hay comprobación de errores en tiempo de compilación.

Esta es la forma “incorrecta”. Usando la herencias de la superclase cósmica object:

namespace genericos

{

class Program

{

static void Main(string[] args)

{

AlmacenObjetos Archivos = new AlmacenObjetos(4);

/\*Archivos.Agregar("Juan");

Archivos.Agregar("Pedro");

Archivos.Agregar("Lucas");

Archivos.Agregar("Helena");

Console.WriteLine(Archivos.getElemento(2));

//El método no devuelve un string sino un object.

//Hay que hacer un casting

string NombrePersona = (string)Archivos.getElemento(3);

Console.WriteLine(NombrePersona);

//supongase que quiera almacenar objetos de tipo empleado. La cosa se complica un poco mas.\*/

Archivos.Agregar(new Empleado(1500)); //de esta forma se abrevia un poco

Archivos.Agregar(new Empleado(2330));

Archivos.Agregar(new Empleado(2670));

Archivos.Agregar(new Empleado(3200)); //hasta acá no da, hay que hacer un casting

Empleado salarioEmpleado = (Empleado)Archivos.getElemento(2);

Console.WriteLine(salarioEmpleado.getSalario());

}

class AlmacenObjetos

{

public AlmacenObjetos (int z) //le llegarían cuántos objetos va a almacenar el array

{

datosElemento = new Object[z];

}

public void Agregar (Object Obj)

{

datosElemento[i] = Obj; //este es el objeto que se le pasa también por parámetro.

i++;

}

public Object getElemento (int i) //este entero especifica la posición del array de la cual quiero obtener información

{

return datosElemento[i];

}

private Object[] datosElemento;

private int i = 0; //contador

}

class Empleado

{

public Empleado (double salario)

{

this.salario = salario;

}

public double getSalario()

{

return salario;

}

private double salario;

}

//Lo que dice es que todo hereda de Object

}

}

Permite usar objetos de tipo string y de tipo empleados, por eso es como “genérica”, pero tiene el problema de hacer mucho casting.

Para prescindir de la herencia. Borra todos los object.

Crear clase genérica:

class AlmacenObjetos <T> //Por convención es una T

Luego pone la misma T en todo lugar que había un object.

Esa clase manejará cualquier tipo de objetos.

AlmacenObjetos <Empleado> Archivos = new AlmacenObjetos<Empleado>(4);

Archivos.Agregar("Juan");

Archivos.Agregar("Pedro");

Archivos.Agregar("Lucas");

Archivos.Agregar("Helena");

Ahí, tiraría error en los strings. Lo que hay que hacer es cambiar el tipo de datos:

AlmacenObjetos <string> Archivos = new AlmacenObjetos<string>(4);

Archivos.Agregar("Juan");

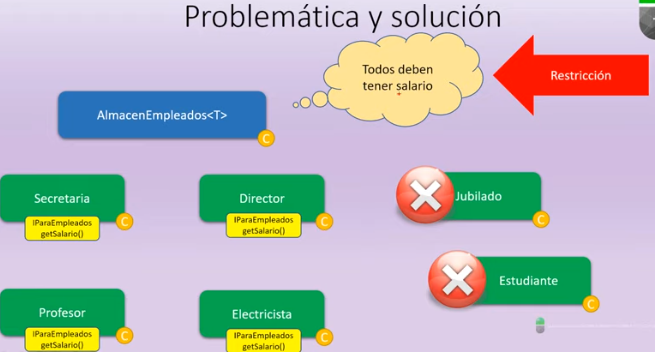
Archivos.Agregar("Pedro");

Archivos.Agregar("Lucas");

Archivos.Agregar("Helena");

Clases genéricas con restricciones

A veces se desea que una clase genérica no maneje determinados tipos de objetos.



class AlmacenEmpleados <T> where T:IParaEmpleados

Ahí se obliga a que los objetos tengan esa interfaz.

Solamente se podrán agregar personas que sean empleadas, es decir, que reciban un salario.

namespace GenericoConRestricciones

{

class Program

{

static void Main(string[] args)

{

AlmacenEmpleados<Director> Empleados = new AlmacenEmpleados<Director>(3);

Empleados.Agregar(new Director(4500));

Empleados.Agregar(new Director(5000));

Empleados.Agregar(new Director(15000));

AlmacenEmpleados<Secretaria> EmpleaSecre = new AlmacenEmpleados<Secretaria>(6);

EmpleaSecre.Agregar(new Secretaria(800));

EmpleaSecre.Agregar(new Secretaria(750));

EmpleaSecre.Agregar(new Secretaria(800));

EmpleaSecre.Agregar(new Secretaria(820));

EmpleaSecre.Agregar(new Secretaria(500));

EmpleaSecre.Agregar(new Secretaria(350));

}

}

interface IParaEmpleados

{

double getSalario();

}

class AlmacenEmpleados <T> where T:IParaEmpleados

{

//Esta clase es igual a la del video anterior de genericos, con la restricción de la lina anterior.

public AlmacenEmpleados (int z)

{

DatosEmpleado = new T[z];

}

public void Agregar (T obj)

{

DatosEmpleado[i] = obj;

i++;

}

public T GetEmpleado (int i)

{

return DatosEmpleado[i];

}

private int i = 0;

private T[] DatosEmpleado;

}

class Director:IParaEmpleados

{

public Director (double salario)

{

this.salario = salario;

}

private double salario;

public double getSalario()

{

return salario;

}

}

class Secretaria : IParaEmpleados

{

public Secretaria (double salario)

{

this.salario = salario;

}

public double getSalario()

{

return salario;

}

private double salario;

}

class Electricista : IParaEmpleados

{

public Electricista(double salario)

{

this.salario = salario;

}

public double getSalario()

{

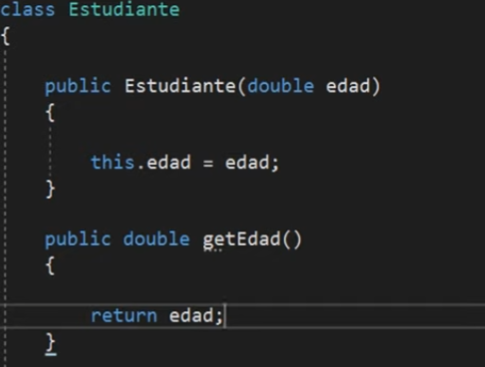
return salario;

}

private double salario;

}

}



Luego agrega eso, al que no le puso el :IParaEmpleados, por lo que tirará error.