

**HERENCIA**

**I.- Ficha técnica**

|  |  |
| --- | --- |
| CARRERA | Técnico de nivel superior en Análisis de sistemas |
| CURSO | Lenguaje de Programación II |
| ASOCIADA AL APRENDIZAJE ESPERADO | Aplicar principios fundamentales de la programación orientada a objetos |
| ASOCIADA A LOS CRITERIOS DE EVALUACIÓN | Identifica objetos, propiedades y métodos  Define clases, objetos, herencia, polimorfismo |
| CLASES (n°) | Clases 7-9 |
| TIPO | Personal |
| DESCRIPCIÓN | Herencias y tipos de herencias, aplicación de herencia entre clases |
| EVIDENCIA | Un programa que contenga herencia entre clases |
| ESPACIO | Laboratorio de computadores |
| INSUMOS | Visual Studio 2010, 2013 ó 2015 |

**Materiales/Equipos/Herramientas (M/E/H)**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Nombre** | **Tipo** | **Marca** | **M/E/H** | **Unidad de medida** | **Cantidad** | **Cada N alumnos** | **Observaciones** |
| Un computador |  |  |  |  | 1 | 1 | Operativo |
| Visual Studio 2010, 2013 ó 2015 |  |  |  |  | 1 | 1 | Instalado |

**II.- MATERIAL PARA EL ESTUDIANTE**

Se adjunta guía HERENCIA

III.- **PROCEDIMIENTO DE LA EXPERIENCIA**

1. Lea atentamente la guía, si tiene dudas consulte con el docente
2. La experiencia es individual. No requiere del uso de material ajeno a ella. Siga las instrucciones expuestas en el documento
3. La experiencia es formativa
4. El producto obtenido en el desarrollo de la experiencia debe ser enviado al docente quien indicará, oportunamente, la forma más apropiada de hacerlo.

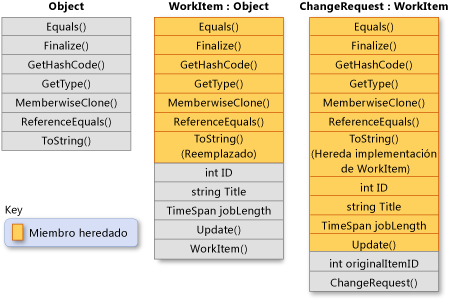
**HERENCIA**

La herencia, junto con la encapsulación y el polimorfismo, es una de las tres características principales (o *pilares*) de la programación orientada a objetos. La herencia permite crear nuevas clases que reutilizan, extienden y modifican el comportamiento que se define en otras clases. La clase cuyos miembros se heredan se denomina *clase base* y la clase que hereda esos miembros se denomina *clase derivada*. Una clase derivada solo puede tener una clase base directa. Sin embargo, la herencia es transitiva. Si ClassC se deriva de ClassB y ClassB se deriva de ClassA, ClassC hereda los miembros declarados en ClassB y ClassA.

Conceptualmente, una clase derivada es una especialización de la clase base. Por ejemplo, si tiene una clase base Animal, puede tener una clase derivada denominada Mammal y otra clase derivada denominada Reptile. Mammal esAnimal y Reptile es Animal, pero cada clase derivada representa especializaciones diferentes de la clase base.

Al definir una clase para derivar de otra clase, la clase derivada obtiene implícitamente todos los miembros de la clase base, salvo sus constructores y destructores. La clase derivada puede, por tanto, reutilizar el código de la clase base sin tener que volver a implementarlo. En la clase derivada, puede agregar más miembros. De esta manera, la clase derivada extiende la funcionalidad de la clase base.

En la ilustración siguiente se muestra una clase WorkItem que representa un elemento de trabajo en algún proceso de negocio. Como todas las clases, deriva de [System.Object](https://msdn.microsoft.com/es-cl/library/system.object.aspx) y hereda de todos sus métodos. WorkItem agrega cinco miembros propios. Incluyen un constructor, porque los constructores no se heredan. La clase ChangeRequest hereda de WorkItem y representa un tipo determinado de elemento de trabajo.  ChangeRequest agrega dos o más miembros a los miembros que hereda de WorkItem y [Object](https://msdn.microsoft.com/es-cl/library/system.object.aspx). Debe agregar su propio constructor, y también agrega originalItemID.La propiedad originalItemID permite que la instancia de ChangeRequest se asocie al WorkItem original al que se aplica la solicitud de cambio.



**Herencia de clases**

En el ejemplo siguiente se muestra cómo se expresan en C# las relaciones de clase presentadas en la ilustración anterior. En el ejemplo también se muestra cómo WorkItem invalida el método virtual [Object.ToString](https://msdn.microsoft.com/es-cl/library/system.object.tostring.aspx) y cómo la claseChangeRequest hereda la implementación de WorkItem del método.

C#

// WorkItem implicitly inherits from the Object class.

public class WorkItem

{

// Static field currentID stores the job ID of the last WorkItem that

// has been created.

private static int currentID;

//Properties.

protected int ID { get; set; }

protected string Title { get; set; }

protected string Description { get; set; }

protected TimeSpan jobLength { get; set; }

// Default constructor. If a derived class does not invoke a base-

// class constructor explicitly, the default constructor is called

// implicitly.

public WorkItem()

{

ID = 0;

Title = "Default title";

Description = "Default description.";

jobLength = new TimeSpan();

}

// Instance constructor that has three parameters.

public WorkItem(string title, string desc, TimeSpan joblen)

{

this.ID = GetNextID();

this.Title = title;

this.Description = desc;

this.jobLength = joblen;

}

// Static constructor to initialize the static member, currentID. This

// constructor is called one time, automatically, before any instance

// of WorkItem or ChangeRequest is created, or currentID is referenced.

static WorkItem()

{

currentID = 0;

}

protected int GetNextID()

{

// currentID is a static field. It is incremented each time a new

// instance of WorkItem is created.

return ++currentID;

}

// Method Update enables you to update the title and job length of an

// existing WorkItem object.

public void Update(string title, TimeSpan joblen)

{

this.Title = title;

this.jobLength = joblen;

}

// Virtual method override of the ToString method that is inherited

// from System.Object.

public override string ToString()

{

return String.Format("{0} - {1}", this.ID, this.Title);

}

}

// ChangeRequest derives from WorkItem and adds a property (originalItemID)

// and two constructors.

public class ChangeRequest : WorkItem

{

protected int originalItemID { get; set; }

// Constructors. Because neither constructor calls a base-class

// constructor explicitly, the default constructor in the base class

// is called implicitly. The base class must contain a default

// constructor.

// Default constructor for the derived class.

public ChangeRequest() { }

// Instance constructor that has four parameters.

public ChangeRequest(string title, string desc, TimeSpan jobLen,

int originalID)

{

// The following properties and the GetNexID method are inherited

// from WorkItem.

this.ID = GetNextID();

this.Title = title;

this.Description = desc;

this.jobLength = jobLen;

// Property originalItemId is a member of ChangeRequest, but not

// of WorkItem.

this.originalItemID = originalID;

}

}

class Program

{

static void Main()

{

// Create an instance of WorkItem by using the constructor in the

// base class that takes three arguments.

WorkItem item = new WorkItem("Fix Bugs",

"Fix all bugs in my code branch",

new TimeSpan(3, 4, 0, 0));

// Create an instance of ChangeRequest by using the constructor in

// the derived class that takes four arguments.

ChangeRequest change = new ChangeRequest("Change Base Class Design",

"Add members to the class",

new TimeSpan(4, 0, 0),

1);

// Use the ToString method defined in WorkItem.

Console.WriteLine(item.ToString());

// Use the inherited Update method to change the title of the

// ChangeRequest object.

change.Update("Change the Design of the Base Class",

new TimeSpan(4, 0, 0));

// ChangeRequest inherits WorkItem's override of ToString.

Console.WriteLine(change.ToString());

// Keep the console open in debug mode.

Console.WriteLine("Press any key to exit.");

Console.ReadKey();

}

}

/\* Output:

1 - Fix Bugs

2 - Change the Design of the Base Class

\*/

[**Métodos abstractos y virtuales**](javascript:void(0))

Cuando una clase base declara un método como [virtual](https://msdn.microsoft.com/es-cl/library/9fkccyh4.aspx), una clase derivada puede [invalidar](https://msdn.microsoft.com/es-cl/library/ebca9ah3.aspx) el método con su propia implementación. Si una clase base declara un miembro como [abstracto](https://msdn.microsoft.com/es-cl/library/sf985hc5.aspx), ese método se debe sobrescribir en cualquier clase no abstracta que herede directamente de dicha clase. Si una clase derivada es abstracta en sí misma, hereda los miembros abstractos sin implementarlos. Los miembros abstractos y virtuales son la base para el polimorfismo, la segunda característica principal de la programación orientada a objetos.

[**Clases base abstractas**](javascript:void(0))

Puede declarar una clase como abstracta si desea evitar la creación directa de instancias por medio de la palabra clave [new](https://msdn.microsoft.com/es-cl/library/51y09td4.aspx). Si hace esto, la clase solo se puede utilizar si una nueva clase se deriva de ella.Una clase abstracta puede contener una o más firmas de método que se declaran a sí mismas como abstractas. Estas firmas especifican los parámetros y el valor devuelto, pero no tienen ninguna implementación (cuerpo del método). Una clase abstracta no tiene que contener miembros abstractos; sin embargo, si una clase contiene un miembro abstracto, la propia clase se debe declarar como abstracta. Las clases derivadas que no son abstractas por sí mismas deben proporcionar la implementación de cualquier método abstracto de una clase base abstracta…

[Interfaces](javascript:void(0))

Una *interfaz* es un tipo de referencia similar en cierto modo a una clase base abstracta compuesta únicamente por miembros abstractos. Cuando una clase implementa una interfaz, debe proporcionar una implementación para todos los miembros de la interfaz. Una clase puede implementar varias interfaces aunque solo puede derivar de una única clase base directa.

Las interfaces se utilizan para definir funciones específicas para las clases que no tienen necesariamente una relación de identidad.Por ejemplo, la interfaz [System.IEquatable<T>](https://msdn.microsoft.com/es-cl/library/ms131187.aspx) puede ser implementada por cualquier clase o struct que deba permitir al código de cliente determinar si dos objetos del tipo son equivalentes (sin embargo, el tipo define la equivalencia). [IEquatable<T>](https://msdn.microsoft.com/es-cl/library/ms131187.aspx) no implica el mismo tipo de relación "es un(a)" que existe entre una clase base y una clase derivada (por ejemplo, un Mammal es un Animal).

[**Acceso de la clase derivada a los miembros de la clase base**](javascript:void(0))

Una clase derivada tiene acceso a los miembros public, protected, internal y protected internal de una clase base. Aunque una clase derivada hereda los miembros privados de una clase base, no puede tener acceso a estos miembros.Sin embargo, todos los miembros privados siguen presentes en la clase derivada y pueden hacer el mismo trabajo que harían en la propia clase base.Por ejemplo, supongamos que un método protegido de la clase base tiene acceso a un campo privado. Este campo debe estar presente en la clase derivada para que el método heredado de la clase base funcione correctamente.

[Evitar la derivación adicional](javascript:void(0))

Una clase puede evitar que otras clases hereden de ella, o de cualquiera de sus miembros, declarándose a sí misma o al miembro como [sealed](https://msdn.microsoft.com/es-cl/library/88c54tsw.aspx).

[**Ocultar miembros de la clase base en la clase derivada**](javascript:void(0))

Una clase derivada puede ocultar miembros de la clase base si los declara con el mismo nombre y firma. Se puede utilizar el modificador new para indicar explícitamente que no se pretende que el miembro sea una invalidación del miembro base. No es necesario utilizar new, pero se generará una advertencia del compilador si no se usa new.