**ACTIVIDAD DE APRENDIZAJE**

|  |  |
| --- | --- |
| **Carrera/s** | Ingeniería en Informática  Ingeniería en Gestión de tecnologías de información  Analista programador Computacional |
| **Sigla Curso** | POO3501 |
| **Modalidad** | Presencial |
| **Versión PDA** | 2015 |
| **Material de apoyo (insumos y equipamiento) para la actividad** | Herencia |

|  |
| --- |
| **NOMBRE DE LA ACTIVIDAD:**  **HerenciaTeleCell** |
| Herencia de Clases  Como hemos visto, C# nos permite realizar la herencia de clases de manera declarativa y con ello se nos brinda la posibilidad de sacar provecho de las ventajas de esta característica esencial de la Programación Orientada a Objetos.  Se cuenta con una aplicación WPF en Visual Studio, que nos permite ingresar la información de varias personas, y se necesita construir una especialización de la clase para los datos de Trabajadores y Clientes, que comparten los datos de una Persona, pero que requieren sus propios datos particulares. Esta aplicación se llama “**SolucionHerencia**” y forma parte de la experiencia N°9.  Las nuevas clases están definidas en el siguiente diagrama de Clases:    Se necesita crear las nuevas enumeraciones y clases, permitiendo su creación y registro en la interfaz de la Aplicación WPF. Se adjunta una solución inicial.  Se debe permitir especificar qué tipo se desea crear y llevar el registro de personas tal como está actualmente. Al seleccionar alguno de los registros, se debe desplegar el detalle particular acorde al tipo (Trabajador o Cliente) en un DataGrid adicional.  Se deben conservar los botones de acción actual, no permitiéndose la creación de botones especiales por tipo de dato.  Nuevas enumeraciones  1. Tanto para la definición de la clase Cliente como para la clase Trabajador, vemos que hay nuevas enumeraciones a utilizar. Por lo tanto lo primero será incluir estas enumeraciones en el archivo de código respectivo (Enumeraciones.cs)  2. Agregamos entonces la definición de la enumeración CargoTrabajador, la cual tendrá los valores que utilizaremos en la propiedad Cargo de la clase Trabajador:    3. Luego agregamos la definición de la enumeración ***TipoCliente***, la cual tendrá los valores que utilizaremos en la propiedad ***Tipo*** de la clase ***Cliente***:    Creando la clase hija Cliente  1. Agregue al proyecto de consola un nuevo elemento del tipo clase Agregar > nuevo Elemento > Código > Clase, a esta nueva clase le daremos el nombre Cliente. No olvidemos definir la clase de acceso público incluyendo el modificador public.  2. Para declarar herencia de esta clase desde la clase Persona, debemos hacer esta definición en la declaración de la clase:    3. Realice la declaración de las propiedades en base a la definición visual de la clase:    4. El constructor por defecto y el sobrecargado con el nombre y apellido, tienen el mismo comportamiento de la sobrecarga de la clase padre ***Persona***, pero en este caso el constructor por defecto llamará al constructor respectivo de ***Persona*** mediante la palabra reservada ***base***, con la que se puede acceder a los constructores y métodos de la clase padre.    5. Otro ejemplo de utilización de base, lo vemos en la implementación del método ***ObtenerInformacion()*** de la clase Cliente:    6. Aquí hemos redefinido la función del método con el uso de ***new*** y para aprovechar el método de la clase padre, usamos su resultado como entrada inicial de la información del cliente mediante ***base.ObtenereInformacion()***. Luego simplemente agregamos los datos propios de la clase ***Cliente***.  Creando la clase hija Trabajador  1. Agregue al proyecto de consola un nuevo elemento del tipo clase con el nombre **Trabajador**. No olvidar definir la clase de acceso público incluyendo el modificador ***public***.  2. Declaramos la clase como hija de la clase ***Persona***:    3. Realice la declaración de las propiedades en base a la definición visual de la clase:      4. En el constructor por defecto aseguramos la inicialización de los elementos propios de la clase, esta no tiene constructores sobrecargados.    5. Al igual que en la clase cliente, realizamos la implementación del método ***ObtenerInformacion()*** reutilizando el método original de la clase ***Persona*** e incluimos la información propia de la clase ***Trabajador***:    Prueba de las clases heredadas  1. Para hacer uso de nuestras clases y crear instancias de ellos, agregamos un nuevo proyecto del tipo Aplicación de Consola de nombre “***ClienteConsola*”**. Recuerde que para hacer uso de las clases creadas en la biblioteca, debemos hacer la referencia a ese proyecto desde nuestro proyecto de consola.  2. Incluiremos un nuevo método de nombre **UsoHerencia()**, en el que generaremos una instancia de cada clase hija y desplegaremos su información por consola utilizando el método **ObtenerInformacion()**:    3. El método programado hace uso directo de las nuevas implementaciones del método ObtenerInformacion() que han redefinido las clases hijas. Pero es posible hacer uso del método original con las instancias de las clases hijas, para ello crearemos un nuevo método auxiliar de nombre MostrarInformacionPersona(), el cual recibe como parámetro un objeto de la clase Persona:  4. Incluimos un nuevo método que llamaremos desde **Main()** de nombre **UsoPolimorfismo()**, en él se crearán las instancias de **Cliente** y **Trabajador**. Luego llamaremos al método **MostrarInformacionPersona()** y enviaremos como parámetro las respectivas instancias de las clases hijas, para ver el comportamiento deseado:    5. Compilamos y ejecutamos el programa con depuración para ver los resultados y el tratamiento de las instancias.  Utilizando la clase desde una aplicación cliente WPF  Incorporaremos un cliente WPF, para capturar y tratar las instancias de las clases heredadas separando la captura de las propiedades base de las propiedades particulares de las clases hijas:  1. En la solución trabajaremos con el proyecto de tipo Aplicación WPF y nombre “***ClienteWPF***” que será nuestro proyecto de partida.  2. Tenemos en la parte superior los controles para la captura de los datos comunes:    3. A continuación definiremos 2 áreas para capturar los datos particulares de cada clase hija. El control GroupBox nos permite separar y agrupar los datos:    4. Al pie del formulario agregaremos 2 controles DataGrid, para desplegar la información de las instancias creadas:    5. A nivel del formulario existe un arreglo para capturar las instancias creadas en los botones, este arreglo se declara de tamaño cero (0) por que se irá redimensionando por cada nueva persona:    6. En el método CargarListas() agregaremos la carga de los ComboBox para las nuevas enumeraciones y les asignaremos un valor por defecto:    7. Considerando que los datos comunes de Trabajador y Cliente corresponden a su clase base Persona, incluiremos un método que cargue los datos comunes, más los datos de la especialización en una instancia de persona que llegará como parámetro:    8. Como necesitamos agregar elementos de manera dinámica, se ha incluido código que nos permite realizar esta acción redimensionando el tamaño del arreglo:    9. Existe un método que nos permite ir visualizando las personas agregadas en el DataGrid:    10. Los otros botones ya se encuentran codificados y no necesitan modificaciones para continuar funcionando.  11. El método CargarDatos() que se llama cada vez que se selecciona una persona desde el DataGrid principal, debe ser modificado para que ahora muestre el detalle de la herencia de cada persona.    12. Al seleccionar una persona en el DataGrid, sus datos se cargan en la interfaz y además en el DataGrid segundario. |