Unidad

4

DIPLOMATURA EN PROGRAMACION .NET

Ejercicios

Capítulo 8

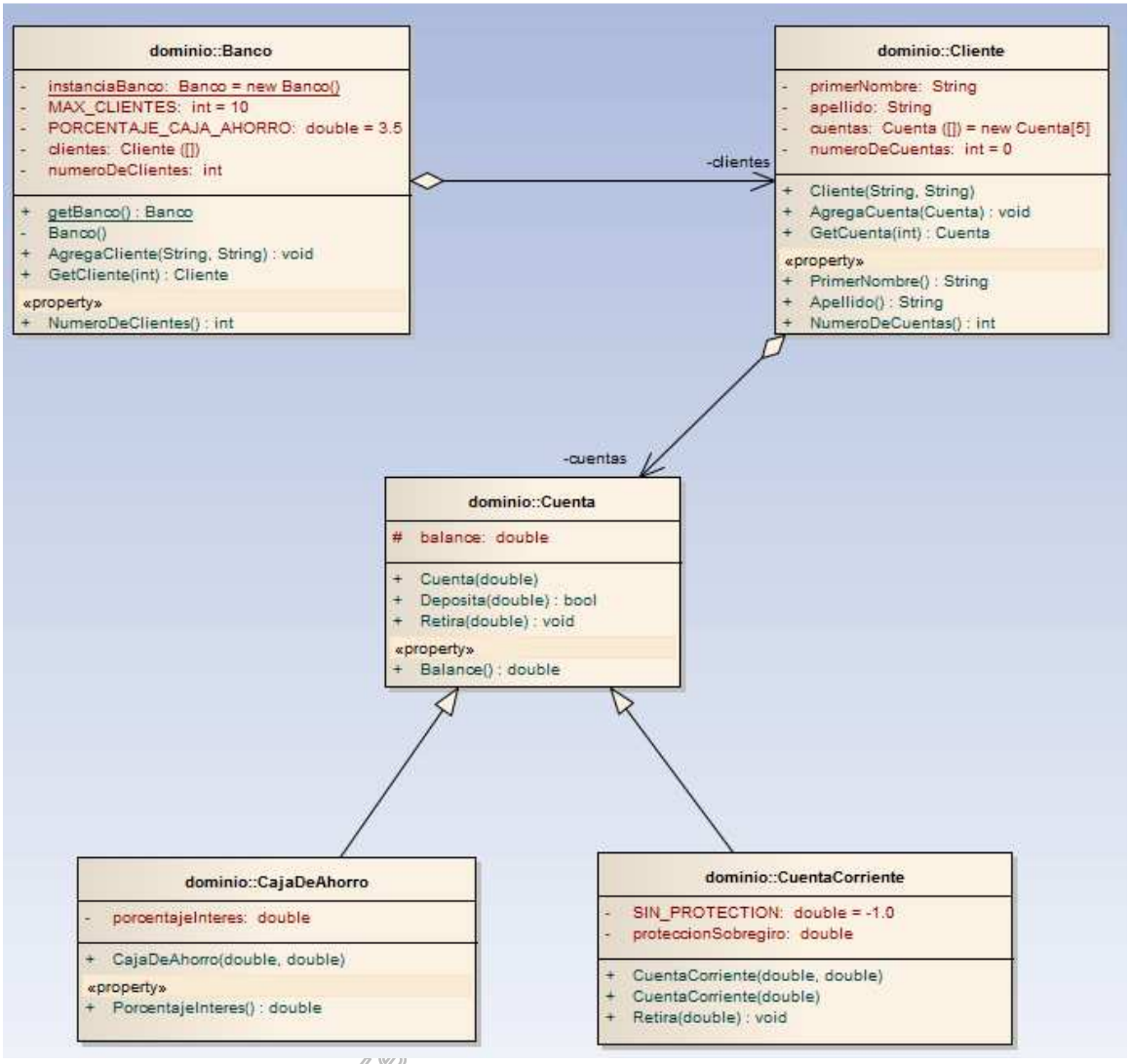
Colecciones

Ejercicio 1

En este ejercicio va a usar colecciones para representar asociaciones de clases en el modelo de dominio del proyecto de operaciones bancarias.

En un diseño anterior se usaron vectores para implementar la multiplicidad en las relaciones entre el banco sus clientes, y entre los clientes y sus cuentas. Este diseño tiene varias limitaciones importantes, la principal es que el vector, una vez creado, tiene un tamaño fijo. Esta limitación de diseño se trató en el capítulo de genéricos, pero por cuestiones de compatibilidad de código, el área de diseño le solicita que nuevamente cambie el código implementando colecciones del Framework de .Net que no sean genéricas hasta que el código heredado permita implementar colecciones genéricas.

La siguiente figura muestra el modelo de dominio del proyecto de operaciones bancarias con las asociaciones de clases: un banco presta servicio (tiene) a muchos clientes y un cliente tiene muchas cuentas.



Abrir la solución con los ejercicios del capítulo 8

1. Abrir la solución en el directorio donde están instalados los ejercicios llamada Capitulo8.

Modificación de la clase Banco

2. Eliminar la declaración `private int MAX_CLIENTES = 10;` o `Private MAX_CLIENTES As Integer = 10` (C# o VB)
3. Modificar la declaración de la variable de instancia clientes para que sea del tipo ArrayList.
4. Borrar la inicialización de la variable de instancia numeroDeClientes en el constructor. Crear el objeto del tipo ArrayList en el constructor
5. Modificar el método AgregarCliente. Eliminar la declaración de la variable entera como el incremento y asignación a ésta de la variable de instancia numeroDeClientes. Reemplazar la asignación del nuevo cliente al vector por el

agregado de un nuevo nodo de la lista. Asignar la propiedad Count de la lista a la variable numeroDeClientes.

6. Modificar el método GetCliente para que retorne un objeto del tipo Cliente en lugar de un Object (que es lo que contiene el objeto del tipo ArrayList). Usar una conversión de tipo.

Modificación de la clase Cliente

7. Modificar la declaración de la variable de instancia cuentas para que sea del tipo ArrayList.
8. Crear el objeto del tipo del tipo ArrayList en el constructor.
9. Modificar el método AgregaCuenta para que utilice el objeto del tipo ArrayList para agregar cada cuenta y para que la variable de instancia numeroDeCuentas almacene el valor de la propiedad Count de dicho objeto, luego de incorporar la nueva cuenta.
10. Modificar el método GetCuenta para que retorne un objeto del tipo Cuenta en lugar de un Object (que es lo que contiene el objeto del tipo ArrayList). Usar una conversión de tipo.

Verificación de la clase ReporteCliente

11. Corroborar que la clase ReporteCliente no tiene ningún error de compilación

Ejecución del programa

12. Ejecutar el programa desde la clase PruebaOperacionesBancarias. La salida debe ser similar a la siguiente:

```
El Cliente [Perez, Juan] tiene un balance en cuenta corriente de 500 con
una protección por sobregiro de 500.00.
Cuenta Corriente [Juan Perez]: retira 150.00
Cuenta Corriente [Juan Perez]: deposita 22.50
Cuenta Corriente [Juan Perez]: retira 147.62
Cuenta Corriente [Juan Perez]: retira 470.00
Excepción: Fondos Insuficientes   Deficit: 245,12
El Cliente [Perez, Juan] tiene un balance en cuenta corriente de 224,88

El Cliente [Toma, Oscar] tiene un balance en cuenta corriente de 200
Cuenta Corriente [Oscar Toma]: retira 100.00
Cuenta Corriente [Oscar Toma]: deposita 25.00
Cuenta Corriente [Oscar Toma]: retira 175.00
Excepción: No hay protección por sobregiro   Deficit: 50
El Cliente [Toma, Oscar] tiene un balance en cuenta corriente de 125

El Cliente [García, Pedro] tiene un balance en cuenta corriente de 500
Cuenta Corriente [Pedro García]: retira 50.00
Cuenta Corriente [Pedro García]: deposita 250.00
Cuenta Corriente [Pedro García]: retira 400.00
El Cliente [García, Pedro] tiene un balance en cuenta corriente de 300


                    REPORTE DE CLIENTES
                    ===================

Cliente: Perez, Juan
```

```
    Cuenta Corriente: el balance actual es 200
    Caja de Ahorro: el balance actual es 224,88

Cliente: Toma, Oscar
    Cuenta Corriente: el balance actual es 125
    Caja de Ahorro: el balance actual es 700

Cliente: García, Pedro
    Cuenta Corriente: el balance actual es 300
    Caja de Ahorro: el balance actual es 900

Cliente: Soley, Maria
    Caja de Ahorro: el balance actual es 700
```

Ejercicio 2

En este ejercicio se ordenará la lista de los clientes del banco por sus nombres, lo cual requiere que la clase Cliente implemente la interfaz IComparable.

Realizar las siguientes tareas:

1. Crear un nuevo proyecto y llamarlo ejercicio2.
2. Copiar las clases del proyecto del ejercicio anterior (en el caso de VB se deberá actualizar el espacio de nombres)

Clase Cliente

3. Modificar la clase Clientes para implementar la interfaz IComparable. Se necesitará especificar el método CompareTo. Hacer que este método compare dos clientes en orden lexicográfico tomando como precedencia el apellido antes del nombre

Clase Banco

4. Agregar el método OrdenarClientes. En él ordenar la lista del tipo ArrayList invocando al método Sort, el cual utilizará la implementación de la interfaz IComparable realizada en el punto anterior.

Clase PruebaOperacionesBancarias

5. Modificar la clase PruebaOperacionesBancarias para llamar al método ordenarClientes antes de generar el reporte.
6. Compilar y ejecutar el programa PruebaOperacionesBancarias y verificar que los clientes están ordenados por apellido y nombre según se estableció en la implementación de la interfaz y que se obtenga la siguiente salida:

```
El Cliente [Perez, Juan] tiene un balance en cuenta corriente de 500 con
una pro
tección por sobregiro de 500.00.
Cuenta Corriente [Juan Perez]: retira 150.00
Cuenta Corriente [Juan Perez]: deposita 22.50
Cuenta Corriente [Juan Perez]: retira 147.62
Cuenta Corriente [Juan Perez]: retira 470.00
Excepción: Fondos Insuficientes   Deficit: 245,12
```

```
El Cliente [Perez, Juan] tiene un balance en cuenta corriente de 224,88

El Cliente [Toma, Oscar] tiene un balance en cuenta corriente de 200
Cuenta Corriente [Oscar Toma]: retira 100.00
Cuenta Corriente [Oscar Toma]: deposita 25.00
Cuenta Corriente [Oscar Toma]: retira 175.00
Excepción: No hay protección por sobregiro   Deficit: 50
El Cliente [Toma, Oscar] tiene un balance en cuenta corriente de 125

El Cliente [García, Pedro] tiene un balance en cuenta corriente de 500
Cuenta Corriente [Pedro García]: retira 50.00
Cuenta Corriente [Pedro García]: deposita 250.00
Cuenta Corriente [Pedro García]: retira 400.00
El Cliente [García, Pedro] tiene un balance en cuenta corriente de 300

                        REPORTE DE CLIENTES
                        ===================

Cliente: García, Pedro
    Cuenta Corriente: el balance actual es 300
    Caja de Ahorro: el balance actual es 900

Cliente: Perez, Juan
    Cuenta Corriente: el balance actual es 200
    Caja de Ahorro: el balance actual es 224,88

Cliente: Soley, Maria
    Caja de Ahorro: el balance actual es 700

Cliente: Toma, Oscar
    Cuenta Corriente: el balance actual es 125
    Caja de Ahorro: el balance actual es 700
```


Ejercicio 3

El equipo de diseño le acaba de informar que el código ahora soporta el uso de colecciones e interfaces genéricas, razón por la cual le solicitan modificar nuevamente el código para asegurar los tipos que se emplean. Para ello, deberá realizar las siguientes tareas:

1. Modificar la interfaz implementada en la clase Cliente para que admita genéricos (usar IComparable<T> en lugar de IComparable). Tener en cuenta que para implementar el parámetro genérico de la interfaz debe utilizarse el tipo Cliente, que es el tipo de objeto que se desea comparar. Sacar todas las verificaciones y conversiones de tipo que se realizan en el método CompareTo ya que al ser un tipo asegurado no es necesario verificar el tipo de objeto recibido como parámetro.
2. También modificar el ArrayList para que se utilice en su lugar la clase genérica List<T>. Esto requiere que se realicen los correspondientes cambios tanto en la clase Banco como en la clase Cliente.
3. Sacar la conversión de tipo en el método GetCliente porque ya no es necesaria al asegurar el tipo.

4. Sacar la conversión de tipo en el método GetCuenta porque ya no es necesaria al asegurar el tipo.
5. Verificar que la salida queda inalterable y sólo mejoró la calidad del código porque se aseguró el tipo.