Práctica A: Creación y uso de clases

Objetivos

Al final de esta práctica, usted será capaz de:

- Crear clases y hacer instancias de objetos.
- Usar datos y métodos no estáticos.
- Usar datos y métodos estáticos.

Requisitos previos

Antes de realizar la práctica debe estar familiarizado con los siguientes temas:

- Creación de métodos en C#
- Paso de argumentos como parámetros de métodos en C#

Ejercicio 1
Creación y uso de una clase

En este ejercicio retomará la estructura de cuenta bancaria desarrollada a lo largo del módulo y la convertirá en una clase. Declarará sus datos como privados, pero incluirá métodos públicos no estáticos para acceder a los datos. Escribirá un sistema de prueba que cree un objeto cuenta y le asigne un número de cuenta y un saldo indicados por el usuario. Finalmente, imprimirá los datos en la cuenta.

Cómo convertir la estructura CuentaBancaria en una clase

1. Abra el proyecto CreateAccount.sln en la carpeta Starter\CreateAccount dentro del fichero lab07.zip.
2. Estudie el código del archivo BankAccount.cs. Observe que *BankAccount* (CuentaBancaria) es un tipo struct.
3. Compile y ejecute el programa. Tendrá que introducir un número de cuenta y un saldo inicial. Repita este proceso para crear otra cuenta.
4. Modifique *BankAccount* en BankAccount.cs para que sea una clase en lugar de una estructura.
5. Compile el programa. No se compilará. Abra el archivo CreateAccount.cs y vea la clase **CreateAccount**. La clase tendrá este aspecto:

```
class CreateAccount
{
    ...
    static BankAccount NewBankAccount( )
    {
        BankAccount created;
        ...
        created.accNo = number; // Error aquí
        ...
    }
    ...
}
```

6. La asignación a *created.accNo* no daba error de compilación cuando **BankAccount** era una estructura, pero no se compila ahora que es una clase. Esto se debe a que, cuando **BankAccount** era una estructura, la declaración de la variable *created* creaba un *valor* **BankAccount** (en la pila). Ahora que **BankAccount** es una clase, la declaración de la variable *created* no crea un valor **BankAccount**; sino una *referencia* **BankAccount** que todavía no apunta a un *objeto* **BankAccount**.

7. Cambie la declaración de *created* para que se inicialice con un objeto **BankAccount** de nueva creación, como se indica:

```
class CreateAccount
{
    ...
    static BankAccount NewBankAccount( )
    {
        BankAccount created = new BankAccount( );
        ...
        created.accNo = number;
        ...
    }
    ...
}
```

8. Guarde el trabajo realizado.
9. Compile y ejecute el programa. Compruebe que los datos introducidos en la consola se leen correctamente y se muestran en el método **CreateAccount.Write**.

Cómo encapsular la clase BankAccount

1. Todos los datos de la clase **BankAccount** son aún públicos. Modifíquelos para que sean privados, como se indica:

```
class BankAccount
{
    private long accNo;
    private decimal accBal;
    private AccountType accType;
}
```

2. Compile el programa. No se compilará. El error se produce en la clase **CreateAccount**, como se muestra a continuación:

```
class CreateAccount
{
    ...
    static BankAccount NewBankAccount( )
    {
        BankAccount created = new BankAccount( );
        ...
        created.accNo = number; // Error aquí de nuevo
        ...

    }
    ...
}
```

3. Ahora las asignaciones de datos de **BankAccount** no se compilan porque los datos son privados y sólo los métodos de **BankAccount** pueden acceder a datos privados de **BankAccount**. Tiene que escribir un método público de **BankAccount** para que haga las asignaciones. Ejecute los siguientes pasos:

   Añada a **BankAccount** un método público no estático llamado **Populate**. Este método devolverá **void** y recibirá dos parámetros: un entero largo (el número de la cuenta bancaria) y un decimal (el saldo de la cuenta bancaria). El cuerpo de este método asignará el parámetro entero largo al campo *accNo* y el parámetro decimal al campo *accBal*. También dirigirá el campo *accType* a **AccountType.Checking**, como se muestra a continuación:

```
class BankAccount
{
    public void Populate(long number, decimal balance)
    {
        accNo = number;
        accBal = balance;
        accType = AccountType.Checking;
    }

    private long accNo;
    private decimal accBal;
    private AccountType accType;
}
```

4. Marque como comentarios las tres asignaciones a la variable *created* en el método **CreateAccount.NewbankAccount** y añada en su lugar una instrucción que llame al método **Populate** sobre la variable *created*, pasando como argumentos **number** y **balance**. El código será el siguiente:

```
class CreateAccount
{
    ...
    static BankAccount NewBankAccount( )
    {
        BankAccount created = new BankAccount( );
        ...
        // created.accNo = number;
        // created.accBal = balance;
        // created.accType = AccountType.Checking;

        created.Populate(number, balance);
        ...
    }
    ...
}
```

5. Guarde el trabajo realizado.

6. Compile el programa. No se compilará. En el método **CreateAccount**.**Write** quedan todavía tres instrucciones que intentan acceder directamente a los campos privados de **BankAccount**. Tiene que escribir tres métodos públicos de **BankAccount** que devuelvan los valores de estos tres campos. Ejecute los siguientes pasos:

   a. Añada a **BankAccount** un método público no estático llamado **Number**. Este método devolverá un entero largo y no recibirá ningún parámetro. Devolverá el valor del campo *accNo*, como se muestra a continuación:

```
class BankAccount
{
    public void Populate(...) ...

    public long Number( )
    {
        return accNo;
    }
    ...
}
```

   b. Añada a **BankAccount** un método público no estático llamado **Balance**, como muestra el siguiente código. Este método devolverá un decimal y no recibirá ningún parámetro. Devolverá el valor del campo *accBal*.

```
class BankAccount
{
    public void Populate(...) ...

    ...
    public decimal Balance( )
    {
        return accBal;
    }
    ...
}
```

   c. Añada a **BankAccount** un método público no estático llamado **Type**, como muestra el siguiente código. Este método devolverá un **AccountType** y no recibirá ningún parámetro. Devolverá el valor del campo *accType*.

```
class BankAccount
{
    public void Populate(...) ...

    ...
    public AccountType Type( )
    {
        return accType;
    }
    ...
}
```

d. Finalmente, sustituya las tres instrucciones del método **CreateAccount.Write** que intentan acceder directamente a los campos privados de **BankAccount** por llamadas a los tres métodos públicos que acaba de crear, como se muestra a continuación:

```
class CreateAccount
{
    ...
    static void Write(BankAccount toWrite)
    {
        Console.WriteLine("Account number is {0}",
➥toWrite.Number( ));
        Console.WriteLine("Account balance is {0}",
➥toWrite.Balance( ));
        Console.WriteLine("Account type is {0}",
➥toWrite.Type( ));
    }
}
```

7. Guarde el trabajo realizado.
8. Compile el programa y corrija los posibles errores. Ejecute el programa. Compruebe que los datos introducidos en la consola y pasados al método **BankAccount.Populate** se leen correctamente y se muestran en el método **CreateAccount.Write**.

Cómo encapsular aún más la clase BankAccount

1. Cambie el método **BankAccount.Type** para que devuelva el tipo de la cuenta como una cadena de caracteres en lugar de cómo una enumeración **AccountType**, como se muestra a continuación:

```
class BankAccount
{
    ...
    public string Type( )
    {
        return accType.ToString( );
    }
    ...
    private AccountType accType;
}
```

2. Guarde el trabajo realizado.
3. Compile el programa y corrija los posibles errores. Ejecute el programa. Compruebe que los datos introducidos en la consola y pasados al método **BankAccount.Populate** se leen correctamente y se muestran en el método **CreateAccount.Write**.

Ejercicio 2
Generación de números de cuenta

En este ejercicio modificará la clase **BankAccount** del Ejercicio 1 para que genere números de cuenta exclusivos. Para ello usará una variable estática en la clase **BankAccount** y un método que incrementa y devuelve el valor de esta variable. Cuando cree una nueva cuenta, el sistema de prueba llamará a este método para generar el número de cuenta. Luego llamará al método de la clase **BankAccount** que fija el número para la cuenta, pasando este valor como parámetro.

Cómo comprobar que cada número de BankAccount es único

1. Abra el proyecto UniqueNumbers.sln en la carpeta Starter\UniqueNumbers dentro del fichero lab07.zip.

   Nota Este proyecto es el mismo que el proyecto CreateAccount finalizado en el Ejercicio 1.

2. Añada un entero largo privado estático llamado *nextAccNo* a la clase **BankAccount**, como se muestra a continuación:

```
class BankAccount
{
    ...
    private long accNo;
    private decimal accBal;
    private AccountType accType;

    private static long nextAccNo;
}
```

3. Añada a la clase **BankAccount** un método público estático llamado **NextNumber**, como se ve en el siguiente código. Este método devolverá un entero largo y no recibirá ningún parámetro. Devolverá el valor del campo *nextAccNo* e incrementará ese campo.

```
class BankAccount
{
    ...
    public static long NextNumber( )
    {
        return nextAccNo++;
    }

    private long accNo;
    private decimal accBal;
    private AccountType accType;

    private static long nextAccNo;
}
```

4. Marque como comentarios la instrucción del método **CreateAccount.NewBankAccount** que escribe en la consola un mensaje para pedir el número de la cuenta bancaria, como se indica:

```
//Console.Write("Enter the account number: ");
```

5. Sustituya la inicialización de *number* en el método **CreateAccount.NewBankAccount** por una llamada al método **BankAccount.NextNumber** que acaba de crear:

```
//long number = long.Parse(Console.ReadLine( ));
long number = BankAccount.NextNumber( );
```

6. Guarde el trabajo realizado.
7. Compile el programa y corrija los posibles errores. Ejecute el programa. Compruebe que los dos números de cuenta son 0 y 1.
8. El valor de inicialización por defecto del campo estático **BankAccount.nextAccNo** es cero en estos momentos. Inicialice este campo de forma explícita a 123.
9. Compile y ejecute el programa. Compruebe que los números de las dos cuentas creadas son 123 y 124.

Cómo encapsular aún más la clase BankAccount

1. Cambie el método **BankAccount.Populate** para que sólo reciba como parámetro el *balance* decimal. Dentro del método, asigne el campo *accNo* utilizando el método estático **BankAccount.NextNumber**, como se muestra a continuación:

```
class BankAccount
{
    public void Populate(decimal balance)
    {
        accNo = NextNumber( );
        accBal = balance;
        accType = AccountType.Checking;
    }
    ...
}
```

2. Convierta **BankAccount.NextNumber** en un método privado:

```
class BankAccount
{
    ...
    private static long NextNumber( ) ...
}
```

3. Marque como comentarios la declaración e inicialización de *number* en el método **CreateAccount.NewBankAccount**. Cambie la llamada al método **created.Populate** para que pase un solo parámetro, como se indica:

```
class CreateAccount
{
    ...
    static BankAccount NewBankAccount( )
    {
        BankAccount created = new BankAccount( );

        //long number = BankAccount.NextNumber( );
        ...
        created.Populate(balance);
        ...
    }
    ...
}
```

4. Guarde el trabajo realizado.
5. Compile el programa y corrija los posibles errores. Ejecute el programa. Compruebe que los dos números de cuenta siguen siendo 123 y 124.

Ejercicio 3
Inclusión de más métodos públicos

En este ejercicio añadirá dos métodos a la clase **Account**: **Withdraw** (Retirar) y **Deposit** (Ingresar).

Withdraw recibirá un parámetro decimal y deducirá del saldo la cantidad indicada. Sin embargo, antes comprobará que se dispone de suficientes fondos, ya que las cuentas no pueden quedar en números rojos. Devolverá un valor booleano que indique si la retirada de fondos ha sido correcta.

Deposit recibirá también un parámetro decimal cuyo valor sumará al saldo de la cuenta. Devolverá el nuevo valor del saldo.

Cómo añadir un método Deposit a la clase BankAccount

1. Abra el proyecto MoreMethods.sln en la carpeta Starter\MoreMethods dentro del fichero lab07.zip.

   Nota Este proyecto es el mismo que el proyecto UniqueNumbers finalizado en el Ejercicio 2.

2. Añada a la clase **BankAccount** un método público no estático llamado **Deposit**, como se ve en el siguiente código. Este método recibirá un parámetro decimal cuyo valor sumará al saldo de la cuenta. Devolverá el nuevo valor del saldo.

```
class BankAccount
{
    ...
    public decimal Deposit(decimal amount)
    {
        accBal += amount;
        return accBal;
    }
    ...
}
```

3. Añada a la clase **CreateAccount** un método público estático llamado **TestDeposit**, como se ve en el siguiente código. Este método devolverá **void** y recibirá un parámetro de **BankAccount**. El método escribirá en la consola un mensaje que pregunte al usuario la cantidad del depósito, capturará como decimal la cantidad introducida y llamará al método **Deposit** sobre el parámetro **BankAccount**, pasando la cantidad como argumento.

```
class CreateAccount
{
    ...
    public static void TestDeposit(BankAccount acc)
    {
        Console.Write("Enter amount to deposit: ");
        decimal amount = decimal.Parse(Console.ReadLine());
        acc.Deposit(amount);
    }
    ...
}
```

4. Añada a **CreateAccount.Main** instrucciones que llamen al método **TestDeposit** que acaba de crear, como muestra el siguiente código. Compruebe que la llamada a **TestDeposit** se hace para los dos objetos de cuenta. Use el método **CreateAccount.Write** para mostrar la cuenta una vez hecho el depósito.

```
class CreateAccount
{
    static void Main( )
    {
        BankAccount berts = NewBankAccount( );
        Write(berts);
        TestDeposit(berts);
        Write(berts);

        BankAccount freds = NewBankAccount( );
        Write(freds);
        TestDeposit(freds);
        Write(freds);
    }
}
```

5. Guarde el trabajo realizado.
6. Compile el programa y corrija los posibles errores. Ejecute el programa. Compruebe que los depósitos funcionan correctamente.

Nota Si le queda tiempo, puede añadir a **Deposit** una comprobación más para asegurarse de que el parámetro decimal que se pasa no es negativo.

Cómo añadir un método Withdraw a la clase BankAccount

7. Añada a la clase **BankAccount** un método público no estático llamado **Withdraw**, como se ve en el siguiente código. Este método recibirá un parámetro decimal con la cantidad que se desea retirar, y la deducirá del saldo sólo si se dispone de suficientes fondos, ya que las cuentas no pueden quedar en números rojos. Devolverá un valor booleano que indique si la retirada de fondos ha sido correcta.

```
class BankAccount
{
    ...
    public bool Withdraw(decimal amount)
    {
        bool sufficientFunds = accBal >= amount;
        if (sufficientFunds) {
            accBal -= amount;
        }
        return sufficientFunds;
    }
    ...
}
```

8. Añada a la clase **CreateAccount** un método público estático llamado **TestWithdraw**, como se ve en el siguiente código. Este método devolverá **void** y recibirá un parámetro de **BankAccount**. El método escribirá en la consola un mensaje que pregunte al usuario la cantidad que desea retirar, capturará como decimal la cantidad introducida y llamará al método **Withdraw** sobre el parámetro **BankAccount**, pasando la cantidad como argumento. Capturará el resultado booleano devuelto por **Withdraw** y escribirá un mensaje en la consola si no ha sido posible retirar la cantidad.

```
class CreateAccount
{
    ...
    public static void TestWithdraw(BankAccount acc)
    {
        Console.Write("Enter amount to withdraw: ");
        decimal amount = decimal.Parse(Console.ReadLine());
        if (!acc.Withdraw(amount)) {
            Console.WriteLine("Insufficient funds.");
        }
    }
    ...
}
```

9. Añada a **CreateAccount.Main** instrucciones que llamen al método **TestWithdraw** que acaba de crear, como muestra el siguiente código. Compruebe que la llamada a **TestWithdraw** se hace para los dos objetos de cuenta. Use el método **CreateAccount.Write** para mostrar la cuenta una vez retirada la cantidad.

```
class CreateAccount
{
    static void Main( )
    {
        BankAccount berts = NewBankAccount( );
        Write(berts);
        TestDeposit(berts);
        Write(berts);
        TestWithdraw(berts);
        Write(berts);

        BankAccount freds = NewBankAccount( );
        Write(freds);
        TestDeposit(freds);
        Write(freds);
        TestWithdraw(freds);
        Write(freds);
    }
 }
```

10. Guarde el trabajo realizado.
11. Compile el programa y corrija los posibles errores. Ejecute el programa. Compruebe que las cantidades se retiran correctamente. Haga pruebas con cantidades que den resultados correctos e incorrectos.