
Programación II

Overriding - Hiding

Es una técnica que permite redefinir alguna propiedad o implementación de método en subclases.

Para hacer uso de la misma, en la clase base hay que definir el miembro como sobreescribible.

En la subclase se puede:

- Derivar los miembros de la clase base. Esto ocurre por defecto.
- Sobreescribir estos miembros.
- Ocultar los miembros

Override vs Hide

Sobreescribir miembros.

- En la clase base los miembros deben tener el modificador **virtual**
- En la clase derivada el miembro es marcado con la palabra **override**
- El miembro está sujeto a las reglas del polimorfismo

Ocultar miembros

- **No** hace falta que los miembros en la clase base tengan el modificador **virtual**
- En la clase derivada el miembro es marcado con la palabra **new**
- El miembro **no** está sujeto a las reglas de polimorfismo

```
class CuentaBancaria
{
    public virtual string Extraer(decimal Monto)
    { return "CuentaBancaria:Extraer(decimal Monto)"; }

    public string Extraer(decimal Monto, MonedaEnum Moneda)
    { return "CuentaBancaria:Extraer(decimal Monto, MonedaEnum Moneda)"; }
}
class CajaAhorro : CuentaBancaria
{
    public override string Extraer(decimal Monto)
    { return "CajaAhorro:" + base.Extraer(Monto).Split(':')[1]; }
}
class CuentaCorriente : CuentaBancaria
{
    public override string Extraer(decimal Monto)
    { return "CuentaCorriente:" + base.Extraer(Monto).Split(':')[1]; }

    public new string Extraer(decimal Monto, MonedaEnum Moneda)
    { return "CuentaCorriente:" + base.Extraer(Monto).Split(':')[1]; }
}
```

```
enum MonedaEnum
{
    Pesos = 1, Dolares = 2, Reales= 3, Euros = 4
}
```

Una enumeración es un tipo distinto que consiste en un conjunto de constantes. Tiene un tipo subyacente que por default es int.

```
CuentaBancaria cb = new CuentaBancaria();
CajaAhorro ca = new CajaAhorro();
CuentaCorriente cc = new CuentaCorriente();

Console.WriteLine(cb.Extraer(10));                   //CuentaBancaria:Extraer
Console.WriteLine(cb.Extraer(10, MonedaEnum.Pesos)); //CuentaBancaria:Extraer
Console.WriteLine(ca.Extraer(10));                   //CajaAhorro:Extraer
Console.WriteLine(ca.Extraer(10, MonedaEnum.Pesos)); //CuentaBancaria:Extraer
Console.WriteLine(cc.Extraer(10));                   //CuentaCorriente:Extraer
Console.WriteLine(cc.Extraer(10, MonedaEnum.Pesos)); //CuentaCorriente:Extraer
```

Polimorfismo

Permite que una aplicación cree una instancia de una clase derivada y la asigne a una variable de la clase base.

Si la aplicación invoca a un método sobreescrito a traves de esta variable, el polimorfismo asegura que se ejecute la versión correcta del miembro invocado.

Beneficios

- Consistencia y simplicidad: la aplicación puede lidiar con instancias de distintas subclases de una manera consistente
- Extensibilidad y resiliencia: si se agregan nuevas subclases el trabajo necesario para mantener el código es mínimo

Probar y comparar con los resultados obtenidos en el ejemplo anterior.

¿Que diferencia hay? **¿Por qué?**

```
List<CuentaBancaria> MisCuentas = new List<CuentaBancaria>();
MisCuentas.Add(new CuentaBancaria());
MisCuentas.Add(new CajaAhorro());
MisCuentas.Add(new CuentaCorriente());

foreach (CuentaBancaria c in MisCuentas)
{
    Console.WriteLine(c.Extraer(10));
    Console.WriteLine(c.Extraer(10, MonedaEnum.Pesos));
}
```

Sobrecarga de operadores

Permite definir el comportamiento de los operadores del lenguaje cuando interactuan con nuestras clases

Retomamos el problema de las posiciones, si tengo dos objetos posición y ambos referencian a las mismas coordenadas queremos que el operador de igualdad devuelva verdadero.

```
public static bool operator ==(Posicion pos1, Posicion pos2)
{
    return pos1.Equals(pos2);
}

public static bool operator !=(Posicion pos1, Posicion pos2)
{
    return ! (pos1 == pos2);
}

public override bool Equals(object obj)
{
    if (obj == null || GetType() != obj.GetType())     { return false; }
    Posicion o = (obj as Posicion);
    return this.x == o.x && this.y == o.y;
}

public override int GetHashCode()
{
    return base.GetHashCode();
}
```

Ejercicios

- Terminar con las clases de Sokoban.
- Agregar la sobrecarga de operadores para Posicion y sobreescribir el método Equals
- Cambiar el evento de juego terminado por una property booleana
- **Agregar las imagenes como recursos.**
    - Seleccionamos el proyecto en el explorador de soluciones, y vamos a propiedades
    - Vamos a la pestaña recursos
    - Elegimos Agregar Recurso -> Nueva Imagen
- Agregar la lógica del juego
- **¿Como lo podemos probar?**