1. **Introducción**

La reutilización y optimización de código son algunas de las principales ventajas de los lenguajes de programación orientados a objetos. A través de la herencia, uno de los cuatro principios de programación orientada a objetos, comprenderá por qué.

Supongamos que desea crear otra instancia de la aplicación LangotnsAnts. En lugar de duplicar el código y mantener varios conjuntos de código por separado, puede usar la herencia y la reutilización del código para crear otras instancias. Este enfoque permite crear y administrar varias instancias de forma más eficaz.

En este módulo, aprenderá a cambiar el código para que admita mejor la reutilización. Al igual que las funciones, las clases son fragmentos independientes de código que puede reutilizar para diferentes propósitos. Demostraremos este concepto mediante la herencia para crear una clase GeneralizedAnt. Esto le permitirá reutilizar el código de forma más eficaz.

1. **Objetivos de aprendizaje**

En este módulo, practicará cómo:

* Cómo usar la herencia.
* Entender las ventajas de la reutilización de código.

1. **Requisitos previos**

* Una instalación de Visual Studio Code configurada para el desarrollo de aplicaciones de C#, incluido el Kit de desarrollo de C#.
* Experiencia con el editor de VS Code.
* Capacidad para desarrollar aplicaciones de consola de C# que implementan la lógica de negocios mediante bucles, instrucciones if, propiedades y métodos.
* Experiencia con las reglas básicas de sintaxis de C#.

1. **Preparación para un proyecto guiado**

Usará el Editor de VS Code como entorno de desarrollo de código. Escribirá código que use la herencia para la reutilización y optimización del código.

1. **Información general del proyecto**

Este módulo le guía a través de los pasos necesarios para seguir con la creación de la demostración de LangtonsAnts. Le enseña a crear una clase que use la herencia.

1. **Configurar**

Siga los pasos que se indican a continuación para prepararse para los ejercicios del proyecto guiado:

1. Cree un directorio de trabajo con el Explorador de archivos en la carpeta de escritorio denominada **LangtonsAnts**.
2. Descargue el archivo zip que contiene el código del proyecto de inicio seleccionando el siguiente vínculo: [Archivos de inicio](https://github.com/MicrosoftLearning/Guided-Project-implement-inheritance-abstraction-polymorphism/tree/main/Starter).
3. Descomprima los archivos de descarga en el directorio de trabajo.
   1. En el equipo local, vaya a la carpeta de descargas en el Explorador de archivos.
   2. Haga clic con el botón derecho en **Guided-Project-implement-inheritance-abstraction-polymorphism-main.zip** y seleccione **Extraer todo**.
   3. Seleccione **Mostrar los archivos extraídos al completar** y, a continuación, **Extraer**.
   4. Abra la carpeta **Guided-Project-implement-inheritance-abstraction-polymorphism-main**.
   5. Escriba Ctrl-A para seleccionar todos los archivos de la carpeta.
   6. Escriba Ctrl-C para copiarlos en el portapapeles.
   7. Vaya a la carpeta LangtonsAnts que creó, escriba Ctrl-V para copiar los archivos en la carpeta.
4. Abra la nueva carpeta **LangstonsAnts** en VS Code.
   1. Haga clic con el botón derecho en un espacio vacío en la pantalla y seleccione **Mostrar más opciones**.
   2. Seleccione **Abrir con código**.

El VS Code debería mostrar las carpetas **Final**, **Solución** e **Inicio** en el panel del Explorador de VS Code.

1. Empiece a trabajar en los ejercicios usando el código de inicio.
   1. Seleccione Ver -> Terminal en el menú.
   2. En el periodo de terminal, escriba cd Starter para cambiar el directorio actual al código de inicio.
   3. Para compilar el código de inicio, escriba dotnet build y presione Entrar.
   4. Ejecute el código de inicio escribiendo dotnet run.

Si pudo ejecutar el código correctamente, ya está listo para comenzar los ejercicios del proyecto guiado. Buena suerte.

**Nota**

Si no pudo compilar y ejecutar el código de inicio, asegúrese de que ha pasado por los pasos 1-5 correctamente y de que se encuentra en el directorio de la ventana Terminal para el código de inicio. Recuerde que para compilar y ejecutar este código deberá estar en el directorio Starter.

1. **Ejercicio: Implementación de GeneralizedAnt**

En este ejercicio, verá cómo funciona la herencia y cómo implementarla. Creará una nueva clase denominada GeneralizedAnt que herede de la clase Ant original.

**Importante**

Debe haber completado las instrucciones de configuración de la unidad anterior, Preparación, antes de comenzar este ejercicio.

1. **Crear una clase de ant GeneralizedAnt que herede de Ant**
2. Seleccione la carpeta Game.
3. Seleccione el icono Nuevo archivo.
4. Escriba GeneralizedAnt.cs para el nombre del archivo.
5. Presione ENTRAR.
6. Cree la clase GeneralizedAnt.

C#Copiar

namespace LangtonsAnts

{

public class GeneralizedAnt

{

}

}

1. Haga que GeneralizedAnt herede de la clase Ant.

C#Copiar

namespace LangtonsAnts

{

public class GeneralizedAnt : Ant

{

}

}

Ahora debe pasar al constructor GeneralizedAnt los mismos parámetros que estaba pasando al constructor de clase Ant.

1. Agregue un constructor que tome los mismos parámetros que la clase original Ant.

C#Copiar

namespace LangtonsAnts

{

public class GeneralizedAnt : Ant

{

public GeneralizedAnt(int i, int j, AntDirection direction)

{

}

}

}

Aunque pasará estos parámetros de constructor a GeralizedAnt, va a usar el constructor de la clase Ant para inicializar GeneralizedAnt.

La clase Ant amplía la clase base GeneralizedAnt. Para que GeneralizedAnt use el constructor Ant, modifique el constructor GeneralizedAnt, indicándole que pase los parámetros al constructor de clase base Ant.

1. Haga que GeneralizedAnt use el constructor de clase base Ant, como se muestra.

C#Copiar

namespace LangtonsAnts

{

public class GeneralizedAnt : Ant

{

public GeneralizedAnt(int i, int j, AntDirection direction) : base(i, j, direction)

{

}

}

}

Básicamente, estamos pasando los mismos parámetros que pasó a la clase Ant en su nueva clase GeneralizedAnt. A continuación, se hereda el constructor de clase base de GeneralizedAnt (el constructor de clase Ant) para inicializar GeneralizedAnt.

1. Reemplace Ant por GeneralizedAnt.

El último paso es usar la nueva clase. Para ello, modificará la clase Game para que inicialice la matriz Ants mediante la nueva clase GeneralizedAnt.

* 1. Abra el archivo Game.cs en la carpeta Juegos.
  2. Reemplace la línea del constructor para la creación de la matriz Ants que crea un Ant para que ahora cree un GenerlizedAnt en su lugar.

C#Copiar

public Game(int size = 64)

{

Field = new byte[size, size];

Ants = new IAnt[] {

new GeneralizedAnt(i: size/2 + 2, j: size / 2, direction: AntDirection.Up),

new MirrorAnt(i: size/2 - 2, j: size / 2, direction: AntDirection.Down)

};

}

1. Para guardar el trabajo, presione las teclas Ctrl+S (esto guardará todos los archivos abiertos)
2. **Comprobar el trabajo**

Ahora puede comprobar su trabajo. El programa debería funcionar exactamente como antes. Realmente no hemos cambiado nada en términos de jugar al juego o lo que se ve en la pantalla. El único cambio producido al agregar herencia es *cómo* funciona el juego internamente.

1. Seleccione el menú Vista de VS Code.
2. Seleccione Términal.
3. Compile el código, escriba "dotnet build" y presione ENTRAR.
4. Ejecute el código, escriba "dotnet run" y presione ENTRAR.
5. **Ejercicio: Agregar regla a GeneralizedAnt**

En este ejercicio, agregará una regla de movimiento a la clase GeneralizedAnt. Actualmente, LangtonsAnts es sencillo. Tiene dos colores, 0 para negro y 1 para blanco. Una hormiga gira en el sentido de las agujas del reloj cuando está de pie en una celda con color 0 y en sentido contrario al reloj cuando está de pie en 1. Cuando una hormiga llega a una celda 0 o 1, el color de la celda cambia de negro a blanco o de blanco a negro.

Vamos a hacer una regla GeneralizedAnt para que una hormiga pueda girar en cualquier dirección, cuando esté de pie en una celda negra o blanca.

1. **Implementar la propiedad Rule**

Para empezar, agregue una propiedad de valor de cadena pública denominada Rule a GeneralizedAnt. Usará una cadena que asigna la dirección en la que se activa la hormiga, en función del color de su celda anterior. Por ejemplo, la cadena "R" significaría que giramos "derecha" (en el sentido de las agujas del reloj) en el color 0 e "izquierda" (en sentido contrario a las agujas del reloj) en el color 1.

1. Abra el proyecto LangtonsAnts en VS Code.
2. Seleccione la carpeta Game.
3. Abra el archivo GeneralizedAnt.cs .
4. Debajo del constructor, escriba prop y presione Entrar.
5. Para el tipo de valor devuelto, escriba string y presione Entrar dos veces.
6. Escriba Rule para el nombre y presione Entrar para pasar al final de la línea.
7. Al final de la propiedad, agregue = "RL"

C#Copiar

namespace LangtonsAnt

{

public class GeneralizedAnt : Ant

{

public GeneralizedAnt(int i, int j, AntDirection direction) : base(i, j, direction)

{

}

public string Rule { get; set; } = "RL";

}

}

1. **Implementación del método Act**

Ahora debe implementar una nueva versión del método Act que usa la propiedad Rule. El método Act heredado de la clase Ant siempre activa una hormiga en el color 0 y en sentido contrario a las agujas del reloj en 1.

Implemente una nueva versión en la clase GeneralizedAnt que cambie la dirección de la instancia GeneralizedAnt, según la regla establecida por la propiedad Rule. En lugar de cambiar la dirección en función del color de la celda sobre la que se ha colocado, cambiará la dirección en función de la dirección anterior. Si una hormiga fue a la derecha anteriormente, ahora irá a la izquierda, etcétera.

También hará que esta versión del método Act invalide la versión heredada original en la clase Ant. Para ello, realice la versión de la clase original Ant del método virtual y, a continuación, invalide el método en GeneralizedAnt.

1. Cree un método Act debajo de la propiedad Rule.

C#Copiar

public byte Act(byte oldValue)

{

}

**Nota**

No olvide, puede hacer clic con el botón derecho en el editor y seleccionar Aplicar formato al documento para corregir los problemas de formato. Mayús+Alt+F también funcionará.

1. Haga que se "invalide" agregando la palabra clave override.

C#Copiar

public override byte Act(byte oldValue)

{

}

1. Agregue la lógica de la regla.

C#Copiar

public override byte Act(byte oldValue)

{

public override byte Act(byte oldValue)

{

if (Rule[oldValue] == 'R')

{

RotateCW();

}

else

{

RotateCCW();

}

Move();

byte ret;

if (oldValue == 0)

ret = 1;

else // oldValue = 1

ret = 0;

return ret;

}

}

1. Abra el archivo Ant.cs en la carpeta Game.
2. Busque el método Act y agregue la palabra clave virtual.

C#Copiar

public virtual byte Act(byte oldValue)

{

byte ret;

if (oldValue == 0)

{

ret = 1;

RotateCW();

}

else

{

ret = 0;

RotateCCW();

}

Move();

return ret;

}

**Nota**

Al convertir un método virtual en una clase base y reemplazarlo en una clase derivada, se llama a la versión derivada.

1. Para guardar el trabajo, presione las teclas Ctrl+S (esto guardará todos los archivos abiertos)
2. **Comprobar el trabajo**

Ahora puede comprobar su trabajo. El programa debería funcionar exactamente como antes. Realmente no hemos cambiado nada en términos de jugar o lo que se ve en la pantalla. El único cambio producido al agregar herencia es *cómo* funciona el juego internamente.

1. Seleccione el menú Vista de VS Code.
2. Seleccione Terminal.
3. Escriba cd y presione Entrar para ir al directorio Starter, si aún no está en él.
4. Escriba "dotnet build" para compilar el código y presione ENTRAR.
5. Escriba "dotnet run" y presione ENTRAR.
6. **Ejercicio: Reemplazar MirrorAnt**

En esta lección reemplazará MirrorAnt por GeneralizedAnt, de modo que también use el nuevo método Act.

1. **Reemplazar MirrorAnt con GeneralizedAnt**
2. En la carpeta Juego, abra el archivo Game.cs.
3. Reemplace la línea que hace referencia a MirrorAnt por GenralizedAnt.

C#Copiar

Ants = new IAnt[] {

new Ant(i: size/2 + 2, j: size / 2, direction: AntDirection.Up),

new GeneralizedAnt(i: size/2 + 2, j: size / 2, direction: AntDirection.Up),

new GeneralizedAnt(i: size/2 - 2, j: size / 2, direction: AntDirection.Down)

};

1. **Comprobar el trabajo**

Ahora puede volver a comprobar el trabajo. El programa debería funcionar exactamente como lo hizo anteriormente.

1. Seleccione el menú Vista de VS Code.
2. Seleccione Terminal.
3. Para guardar el trabajo, presione las teclas Ctrl+S (esto guardará todos los archivos abiertos)
4. Compile el código, escriba "dotnet build" y presione ENTRAR.
5. Ejecute el código, escriba "dotnet run" y presione ENTRAR.