`switch` es una instrucción de selección que elige una sola *sección switch* para ejecutarla desde una lista de candidatos en función de una coincidencia de patrones con la *expresión de coincidencia*.

C#Copiar

```
using System;

public class Example
{
   public static void Main()
   {
      int caseSwitch = 1;

      switch (caseSwitch)
      {
          case 1:
              Console.WriteLine("Case 1");
              break;
          case 2:
              Console.WriteLine("Case 2");
              break;
          default:
              Console.WriteLine("Default case");
              break;
      }
   }
}
// The example displays the following output:
//       Case 1
```

La instrucción `switch` se suele usar como alternativa a un constructor if-else si una sola expresión se prueba con tres o más condiciones. Por ejemplo, la siguiente instrucción `switch` determina si una variable de tipo `Color` tiene uno de tres valores:

```
using System;

public enum Color { Red, Green, Blue }

public class Example
{
   public static void Main()
   {
      Color c = (Color) (new Random()).Next(0, 3);
      switch (c)
      {
         case Color.Red:
            Console.WriteLine("The color is red");
            break;
```

```
            case Color.Green:
               Console.WriteLine("The color is green");
               break;
            case Color.Blue:
               Console.WriteLine("The color is blue");
               break;
            default:
               Console.WriteLine("The color is unknown.");
               break;
         }
      }
}
```

Es equivalente al siguiente ejemplo que usa un constructor `if-else`.

```
using System;

public enum Color { Red, Green, Blue }

public class Example
{
   public static void Main()
   {
      Color c = (Color) (new Random()).Next(0, 3);
      if (c == Color.Red)
         Console.WriteLine("The color is red");
      else if (c == Color.Green)
         Console.WriteLine("The color is green");
      else if (c == Color.Blue)
         Console.WriteLine("The color is blue");
      else
         Console.WriteLine("The color is unknown.");
   }
}
// The example displays the following output:
//       The color is red
```

Expresión de coincidencia

La expresión de coincidencia proporciona el valor que debe coincidir con los patrones de las etiquetas `case`. Su sintaxis es:

```
   switch (expr)
```

En C# 6 y versiones anteriores, la expresión de coincidencia debe ser una expresión que devuelva un valor de los siguientes tipos:

- Un carácter.

- Una cadena.
- Un booleano.
- Un valor entero, como int o long.
- Un valor enum.

A partir de C# 7.0, la expresión de coincidencia puede ser cualquier expresión que no sea nula.

Sección switch

Una instrucción `switch` incluye una o más secciones switch. Cada sección switch contiene una o más *etiquetas case* (ya sea una etiqueta case o default) seguidas de una o más instrucciones. La instrucción `switch` puede incluir como máximo una etiqueta default colocada en cualquier sección switch. En el ejemplo siguiente se muestra una instrucción `switch` simple con tres secciones switch, cada una de ellas contiene dos instrucciones. La segunda sección switch contiene las etiquetas `case 2:` y `case 3:`.

Una instrucción `switch` puede incluir cualquier número de secciones switch y cada sección puede tener una o más etiquetas case, como se muestra en el ejemplo siguiente. Pero dos etiquetas case no pueden contener la misma expresión.

```
using System;

public class Example
{
   public static void Main()
   {
      Random rnd = new Random();
      int caseSwitch = rnd.Next(1,4);

      switch (caseSwitch)
      {
          case 1:
              Console.WriteLine("Case 1");
              break;
          case 2:
          case 3:
              Console.WriteLine($"Case {caseSwitch}");
              break;
          default:
              Console.WriteLine($"An unexpected value ({caseSwitch})");
              break;
      }
   }
}
```

```
// The example displays output like the following:
//       Case 1
```

Solo se ejecuta una sección switch en una instrucción switch. C# no permite que la ejecución continúe de una sección switch a la siguiente. Por eso, el código siguiente genera un error del compilador, CS0163: "El control no puede pasar explícitamente de una etiqueta de caso (<etiqueta de caso>) a otra".

```
switch (caseSwitch)
{
    // The following switch section causes an error.
    case 1:
        Console.WriteLine("Case 1...");
        // Add a break or other jump statement here.
    case 2:
        Console.WriteLine("... and/or Case 2");
        break;
}
```

Este requisito se suele cumplir al salir explícitamente de la sección switch mediante una instrucción break, goto o return. Pero el código siguiente también es válido, porque garantiza que el control del programa no puede pasar explícitamente a la sección switch `default`.

```
switch (caseSwitch)
{
    // The following switch section causes an error.
    case 1:
        Console.WriteLine("Case 1...");
        break;
    case 2:
    case 3:
        Console.WriteLine("... and/or Case 2");
        break;
    case 4:
        while (true)
           Console.WriteLine("Endless looping. . . .");
    default:
        Console.WriteLine("Default value...");
        break;
}
```

La ejecución de la lista de instrucciones en la sección switch con una etiqueta case que coincide con la expresión de coincidencia comienza con la primera instrucción y continúa a lo largo de la lista de instrucciones, normalmente hasta que se alcanza una instrucción de salto, como `break`, `goto case`, `goto label`, `return` o `throw`. En este punto, el control se transfiere fuera de la

instrucción `switch` o a otra etiqueta case. Una instrucción `goto`, si se usa, debe transferir el control a una etiqueta de constante. Esta restricción es necesaria, ya que el intento de transferir el control a una etiqueta que no es de constante puede tener efectos secundarios no deseados, como la transferencia de control a una ubicación no deseada en el código o la creación de un bucle sin fin.

Etiquetas case

Cada etiqueta case especifica un patrón que se compara con la expresión de coincidencia (la variable `caseSwitch` en los ejemplos anteriores). Si coinciden, el control se transfiere a la sección switch que contiene la **primera** etiqueta case coincidente. Si ningún patrón de etiqueta case coincide con la expresión de coincidencia, el control se transfiere a la sección con la etiqueta case `default`, si la hubiera. Si no hay ninguna etiqueta case `default`, no se ejecuta ninguna instrucción de ninguna sección switch y el control se transfiere fuera de la instrucción `switch`.

Dado que C# 6 solo admite el patrón constante y no permite la repetición de valores constantes, las etiquetas case definen valores mutuamente exclusivos y solo un patrón puede coincidir con la expresión de coincidencia. Por este motivo, el orden en que aparezcan las instrucciones `case` no tiene importancia.

Pero en C# 7.0, dado que se admiten otros patrones, las etiquetas de caso no necesitan definir valores mutuamente exclusivos y varios patrones pueden coincidir con la expresión de coincidencia.

Puesto que solo se ejecutan las instrucciones de la primera sección switch que contiene el patrón coincidente, el orden en que aparecen las instrucciones `case` sí es importante. Si C# detecta una sección switch cuya instrucción o instrucciones case son equivalentes a o son subconjuntos de instrucciones anteriores, genera un error del compilador, CS8120: "El caso del modificador ya se ha gestionado en un caso anterior".

En el ejemplo siguiente se muestra una instrucción `switch` que usa una variedad de patrones que no son mutuamente excluyentes. Si mueve la sección switch `case 0:` de modo que ya no sea la primera sección de la instrucción `switch`, C# genera un error del compilador debido a que un entero cuyo valor es cero es un subconjunto de todos los enteros, que es el patrón definido por la instrucción `case int val`.

```
using System;
using System.Collections.Generic;
using System.Linq;

public class Example
```

```
{
   public static void Main()
   {
      var values = new List<object>();
      for (int ctr = 0; ctr <= 7; ctr++) {
         if (ctr == 2)
            values.Add(DiceLibrary.Roll2());
         else if (ctr == 4)
            values.Add(DiceLibrary.Pass());
         else
            values.Add(DiceLibrary.Roll());
      }

      Console.WriteLine($"The sum of { values.Count } die is {
DiceLibrary.DiceSum(values) }");
   }
}

public static class DiceLibrary
{
   // Random number generator to simulate dice rolls.
   static Random rnd = new Random();

   // Roll a single die.
   public static int Roll()
   {
      return rnd.Next(1, 7);
   }

   // Roll two dice.
   public static List<object> Roll2()
   {
      var rolls = new List<object>();
      rolls.Add(Roll());
      rolls.Add(Roll());
      return rolls;
   }

   // Calculate the sum of n dice rolls.
   public static int DiceSum(IEnumerable<object> values)
   {
      var sum = 0;
      foreach (var item in values)
      {
            switch (item)
            {
               // A single zero value.
               case 0:
                  break;
```

```
                // A single value.
                case int val:
                   sum += val;
                   break;
                // A non-empty collection.
                case IEnumerable<object> subList when subList.Any():
                   sum += DiceSum(subList);
                   break;
                // An empty collection.
                case IEnumerable<object> subList:
                   break;
                //  A null reference.
                case null:
                   break;
                // A value that is neither an integer nor a collection.
                default:
                   throw new InvalidOperationException("unknown item type");
             }
          }
          return sum;
       }

       public static object Pass()
       {
          if (rnd.Next(0, 2) == 0)
             return null;
          else
             return new List<object>();
       }
    }
```

Puede corregir este problema y eliminar la advertencia del compilador de alguna de estas dos formas:

- Si cambia el orden de las secciones switch.
- Si usa una cláusula when en la etiqueta case.

Etiqueta case `default`

La etiqueta case `default` especifica la sección switch que se va a ejecutar si la expresión de coincidencia no coincide con ninguna otra etiqueta `case`. Si no hay ninguna etiqueta case `default` y la expresión de coincidencia no coincide con ninguna otra etiqueta `case`, el flujo del programa pasa a la instrucción `switch`.

La etiqueta case `default` puede aparecer en cualquier orden en la instrucción `switch`.Independientemente de su orden en el código fuente,

siempre se evalúa en último lugar, después de que se hayan evaluado las demás etiquetas `case`.

Coincidencia de patrones con la instrucción `switch`

Cada instrucción `case` define un patrón que, si coincide con la expresión de coincidencia, provoca la ejecución de su sección switch contenedora. Todas las versiones de C# admiten el patrón de constante. Los demás patrones se admiten a partir de C# 7.0.

Patrón de constante

El patrón de constante comprueba si la expresión de coincidencia es igual a una constante especificada. Su sintaxis es:

```
case constant:
```

donde *constant* es el valor que se va a comprobar. *constant* puede ser *cualquiera de las expresiones de constante siguientes:*

- Un literal booleano, ya sea true o false.
- Cualquier constante entera, como int, long o byte.
- El nombre de una variable const declarada.
- Una constante de enumeración.
- Un literal de carácter.
- Un literal de cadena.

La expresión de constante se evalúa de la siguiente forma:

- Si expr y constant son tipos enteros, el operador de igualdad de C# determina si la expresión devuelve true (es decir, si expr == constant).
- De lo contrario, el valor de la expresión se determina mediante una llamada al método estático Object.Equals(expr, constant).

En el ejemplo siguiente se usa el patrón de constante para determinar si una fecha determinada es un fin de semana, el primer día, el último día o la mitad de la semana laboral. Evalúa la propiedad DateTime.DayOfWeek del día actual con los miembros de la enumeración DayOfWeek.

```
using System;

class Program
{
```

```
    static void Main()
    {
        switch (DateTime.Now.DayOfWeek)
        {
           case DayOfWeek.Sunday:
           case DayOfWeek.Saturday:
              Console.WriteLine("The weekend");
              break;
           case DayOfWeek.Monday:
              Console.WriteLine("The first day of the work week.");
              break;
           case DayOfWeek.Friday:
              Console.WriteLine("The last day of the work week.");
              break;
           default:
              Console.WriteLine("The middle of the work week.");
              break;
        }
    }
}
// The example displays output like the following:
//       The middle of the work week.
```

En el ejemplo siguiente se usa el patrón de constante para controlar la entrada del usuario en una aplicación de consola que simula una cafetera automática.

```
using System;

class Example
{
   static void Main()
   {
       Console.WriteLine("Coffee sizes: 1=small 2=medium 3=large");
       Console.Write("Please enter your selection: ");
       string str = Console.ReadLine();
       int cost = 0;

       // Because of the goto statements in cases 2 and 3, the base cost of
25
       // cents is added to the additional cost for the medium and large
sizes.
       switch (str)
       {
          case "1":
          case "small":
              cost += 25;
              break;
          case "2":
```

```
            case "medium":
                cost += 25;
                goto case "1";
            case "3":
            case "large":
                cost += 50;
                goto case "1";
            default:
                Console.WriteLine("Invalid selection. Please select 1, 2, or
3.");
                break;
        }
        if (cost != 0)
        {
            Console.WriteLine("Please insert {0} cents.", cost);
        }
        Console.WriteLine("Thank you for your business.");
    }
}
// The example displays output like the following:
//         Coffee sizes: 1=small 2=medium 3=large
//         Please enter your selection: 2
//         Please insert 50 cents.
//         Thank you for your business.
```