Instrucciones y excepciones

Descrición General

- Introducción a las instrucciones
- Uso de instrucciones condicionales
- Use instrucciones iterativas
- Uso de instrucciones de salto
- Tratamiento de excepciones básicas
- Lanzamiento de excepciones

Introducción a las instrucciones

- Bloques de instrucciones
- Tipos de instrucciones

Bloques de instrucciones

- Se usan llaves para delimitar bloques

```
{
   // code
}
```

- Un bloque y su bloque padre o pueden tener una variable con el mismo nombre

```
{
   int i;
   ...
   {
      int i;
      ...
   }
}
```

- Bloques hermanos pueden tener variables con el mismo nombre

```
{
   int i;
   ...
}
...
{
   int i;
   ...
}
```

Tipos de instrucciones

Instrucciones Condicionales

Las instrucciones if y switch

Instrucciones de iteración

Las instrucciones while, do, for, y foreach

Instrucciones de salto

Las instrucciones goto, break, y continue

Uso de instrucciones condicionales

- La instrucción if
- Instrucción if en cascada
- La instrucción switch
- Problema: ¿Dónde está el error?

La instrucción if

- **Sintaxis:**

```
if ( expresión-booleana )
  primera-instrucción-incrustada
else
  segunda-instrucción-incrustada
```

- **No hay conversión implicita de int a bool**

```
int x;
...
if (x) ...      // Debe ser if (x != 0) en C#
if (x = 0) ... // Debe ser if (x == 0) en C#
```

Instrucciones if en cascada

```
enum Palo { Treboles, Corazones, Diamantes, Picas}
Palo cartas = Palo.Corazones;
if (cartas == Palo.Treboles)
    color = “Negro”;
else if (cartas == Palo.Corazones)
    color = “Rojo";
else if (palo == Palo.Diamantes)
    color = "Rojo";
else
    color = “Negro";
```

La instrucción switch

- Las instrucciones switch se usan en bloques de varios casos
- Se usan instrucciones break para evitar caídas en

```
switch (palo) {
case Palo.Treboles :
case Palo.Picas :
    color = "Negro"; break;
case Palo.Corazones :
case Palo.Diamantes :
    color = "Rojo"; break;
default:
    color = "ERROR"; break;
}
```

Uso de instrucciones iterativas

- La instrucción while
- La instrucción do
- La instrucción for
- La instrucción foreach
- Problema: ¿Dónde está el error?

La instrucción while

- Ejecuta instrucciones en función de un valor booleano
- Evalúa la expresión booleana al principio del bucle
- Ejecuta las instrucciones mientras el valor booleano sea True

```
int i = 0;
while (i < 10) {
    Console.WriteLine(i);
    i++;
}
```

0 1 2 3 4 5 6 7 8 9

La instrucción do

- Ejecuta instrucciones en función de un valor booleano
- Evalúa la expresión booleana al final del bucle
- Ejecuta las instrucciones mientras el valor booleano sea True

```
int i = 0;
do {
    Console.WriteLine(i);
    i++;
} while (i < 10);
```

0 1 2 3 4 5 6 7 8 9

La instrucción for

- La información de actualización está al principio del bucle

```
for (int i = 0; i < 10; i++) {
    Console.WriteLine(i);
}
```

```
0 1 2 3 4 5 6 7 8 9
```

- Las variables de ur

```
for (int i = 0; i < 10; i++)
    Console.WriteLine(i);
Console.WriteLine(i); // Error: i está fuera de ámbito
```

```
for (int i = 0, j = 0; ... ; i++, j++)
```

La instrucción foreach

- Elige el tipo y el nombre de la variable de iteración
- Ejecuta instrucciones incrustadas para cada elemento de la clase collection

```
ArrayList numeros = new ArrayList( );
for (int i = 0; i < 10; i++ ) {
    numeros.Add(i);
}

foreach (int number in numeros) {
    Console.WriteLine(numero);
}
```

0 1 2 3 4 5 6 7 8 9

Uso de instrucciones de salto

- La instrucción goto
- Las instrucciones break y continue

La instrucción goto

- Transfiere el flujo de control a una instrucción con etiqueta
- Pueden dar lugar fácilmente a código “spaghetti” de difícil interpretación

```
if (numero % 2 == 0) goto Par;
Console.WriteLine(“impar");
goto Fin;
Par:
Console.WriteLine(“par");
Fin:;
```

Las instrucciones break and continue

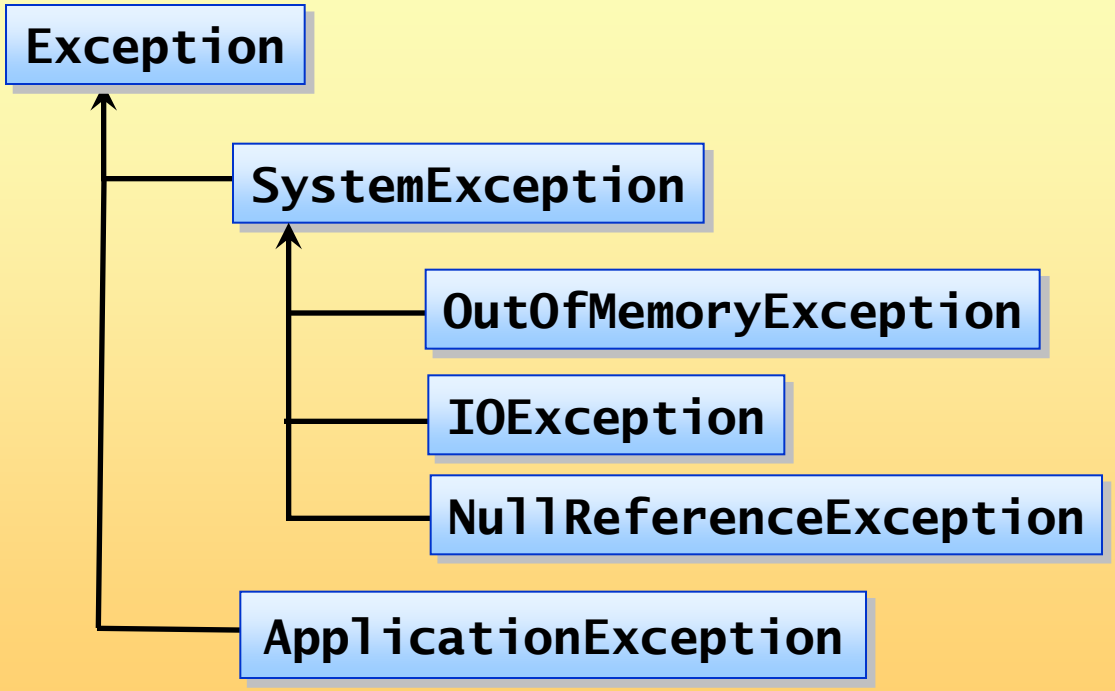
- La instrucción break salta fuera de una iteración
- La instrucción continue salta a la siguiente iteración

```
int i = 0;
while (true) {
    Console.WriteLine(i);
    i++;
    if (i < 10)
        continue;
    else
        break;
}
```

Tratamiento de excepciones básicas

- ¿Por qué se emplean excepciones?
- Objetos excepción
- Uso de bloques try-catch
- Bloques catch múltiples

Obejetos Excepción



Uso de bloques try-catch

- **Solución orientada a objetos para el tratamiento de errores**
  - Poner el código normal en un bloque **try**
  - Tratar las excepciones en un bloque catch aparte

```
try {
      Console.WriteLine("Escriba un número");
      int i = int.Parse(Console.ReadLine());
}
catch (OverflowException capturada)
{
      Console.WriteLine(capturada);
}
```

Lógica del programa

Tratamiento de errores

Bloques catch múltiples

- Cada bloque catch captura una clase de excepcion
- Un bloque try puede tener un bloque catch general
- Un bloque try no puede capturar una clase derivada de una clase capturada en un bloque catch anterior

```
try
{
      Console.WriteLine(“Escriba el primer número");
      int i = int.Parse(Console.ReadLine());
      Console.WriteLine("Escriba el segundo número");
      int j = int.Parse(Console.ReadLine());
      int k = i / j;
}
catch (OverflowException capturada) {…}
catch (DivideByZeroException capturada) {…}
```

Lanzamiento de excepciones

- La instrucción throw
- La cláusula finally
- Comprobación de desbordamiento aritmético
- Normas para el tratamiento de excepciones

La instrucción throw

- Lanza una excepción apropiada
- Asigna a la excepción un mensaje significativo

```
throw expression ;
```

```
if (minuto < 1 || minuto >= 60) {
  throw new InvalidTimeException(minuto +
                                 " no es un minuto válido");
  // !! Not alcanzado !!
}
```

La cláusula finally

- **Las instrucciones de un bloque finally se ejecutan**

```
Monitor.Enter(x);
try {
    ...
}
finally {
    Monitor.Exit(x);
}
```

Bloques catch opcionales

Comprobación de desbordamiento aritmético

- **Por defecto, el desbordamiento aritmético no se comprueba**
  - Un comando checked activa la comprobación de desbordamiento

```
checked {
    int numero = int.MaxValue;
    Console.WriteLine(++numero);
}
```

`OverflowException`

Se lanza un objeto excepción. WriteLine no se ejecuta

```
unchecked {
    int numero = int.MaxValue;
    Console.WriteLine(++numero);
}
```

MaxValue + 1 es negativo?

`-2147483648`

Normas para el tratamiento de excepciones

- **Lanzamiento**
  - Evitar excepciones para casos normales o esperados
  - Nunca crear ni lanzar objetos de clase **Exception**
  - Incluir una cadena de descripción en un objeto **Exception**
  - Lanzar objetos de la clase más específica posible
- **Captura**
  - Ordenar los bloques **catch** de lo específico a lo general
  - No permitir que salgan excepciones de **Main**

Prácticas