


# ALGORITMOS Y PROGRAMACIÓN

---

Clase 7

Manejo de excepciones

# Temario

- Excepciones
  - Manejo de excepciones
  - Disparando excepciones
  - Definiendo excepciones
- 

# Excepción

- Una **excepción es un acontecimiento**, que ocurre durante la **ejecución de un programa**, que **interrumpe el flujo normal** de las instrucciones de programa.
- Por lo general, las excepciones ocurren ante eventos no controlados. Son situaciones que no se previeron al momento de hacer el programa.
- Ejemplo de excepciones:
  - División por cero.
  - Acceder en un array a un elemento de una posición que no existe.
  - Error de casting.
  - Intentar abrir un archivo que no existe.

# Excepción - Ejemplos

```
public static void Main(string[] args) {
```

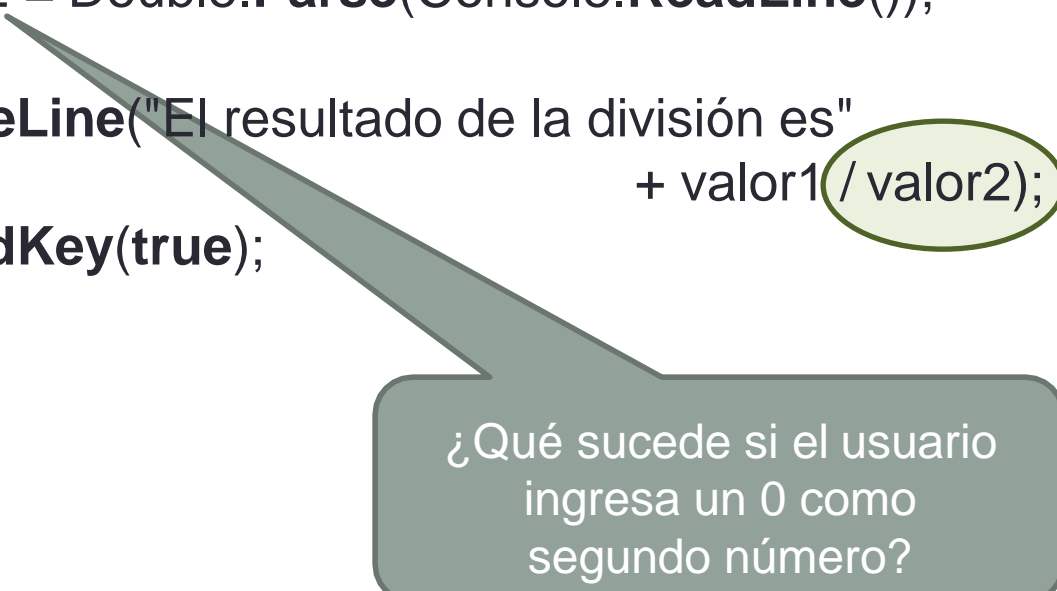
```
    Console.WriteLine("Ingrese un número");  
    double valor1 = Double.Parse(Console.ReadLine());
```

```
    Console.WriteLine("Ingrese otro número");  
    double valor2 = Double.Parse(Console.ReadLine());
```

```
    Console.WriteLine("El resultado de la división es"  
                      + valor1 / valor2);
```

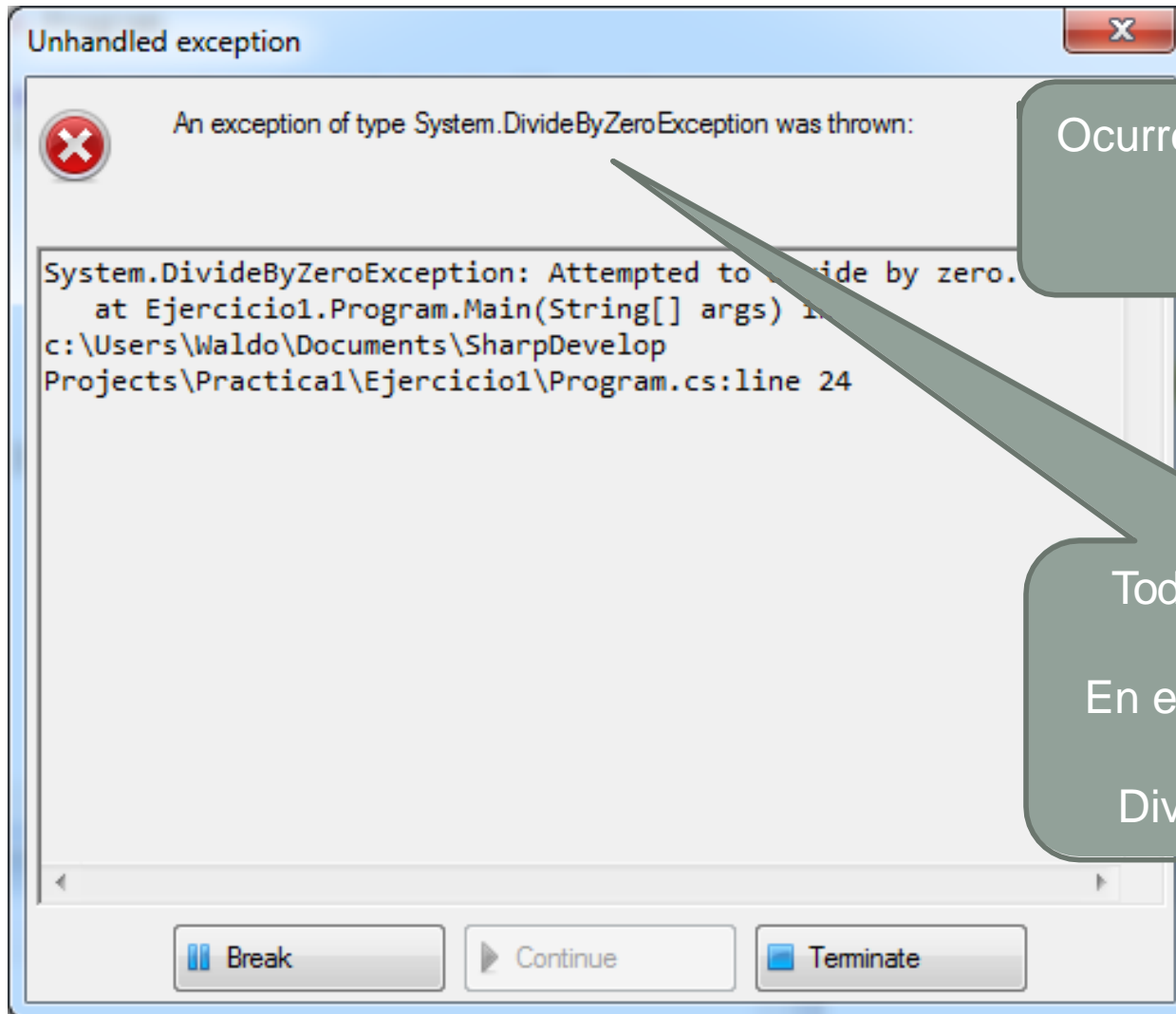
```
    Console.ReadKey(true);
```

```
}
```



¿Qué sucede si el usuario  
ingresa un 0 como  
segundo número?

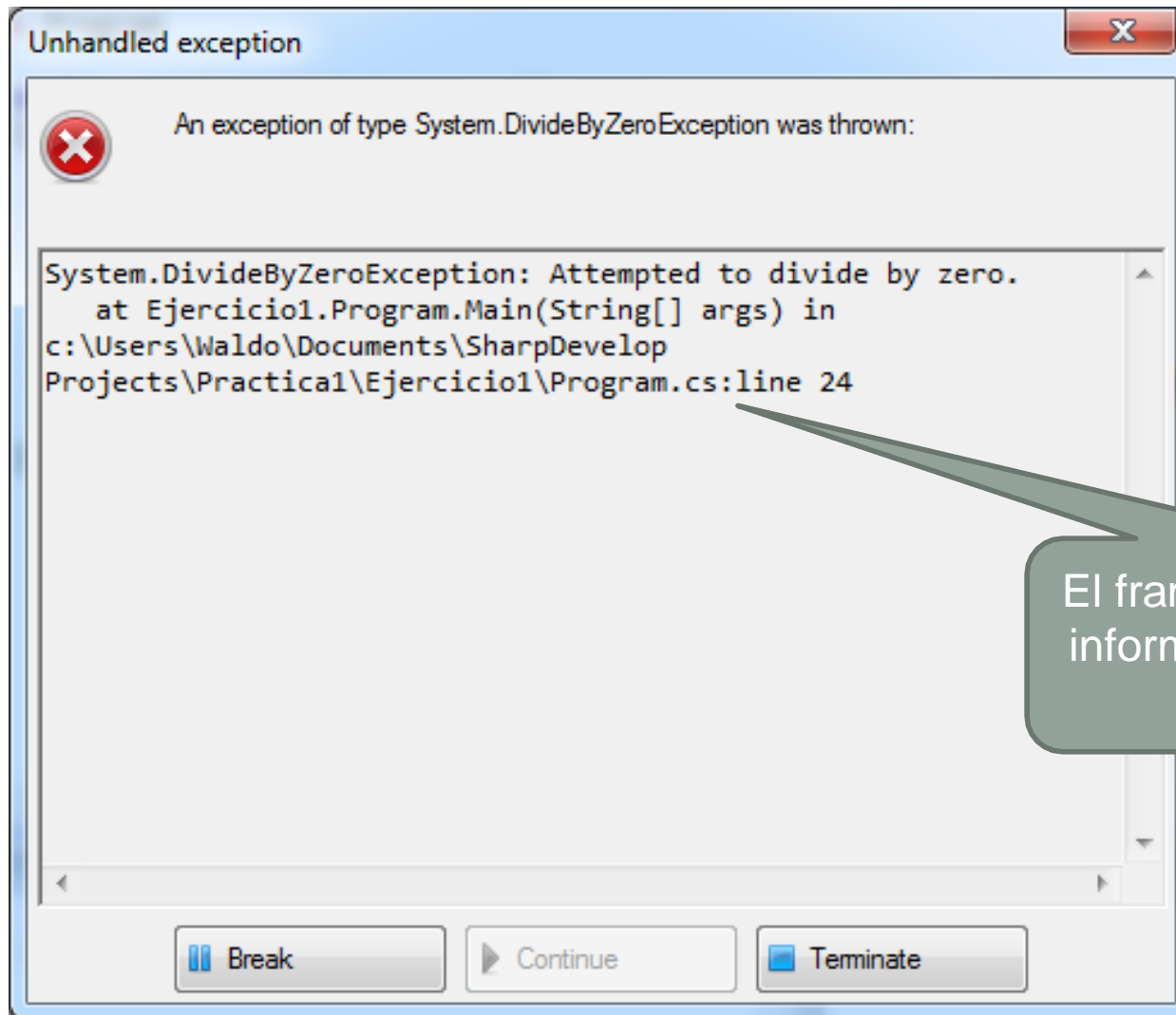
# Excepción - Ejemplo



Ocurre lo que se conoce con el nombre de:  
Excepción

Toda excepción tiene un nombre.  
En este caso el nombre de la excepción es:  
DivideByZeroException

# Excepción - Ejemplo



El framework también nos da información donde ocurrió la excepción

# Excepción - Ejemplo

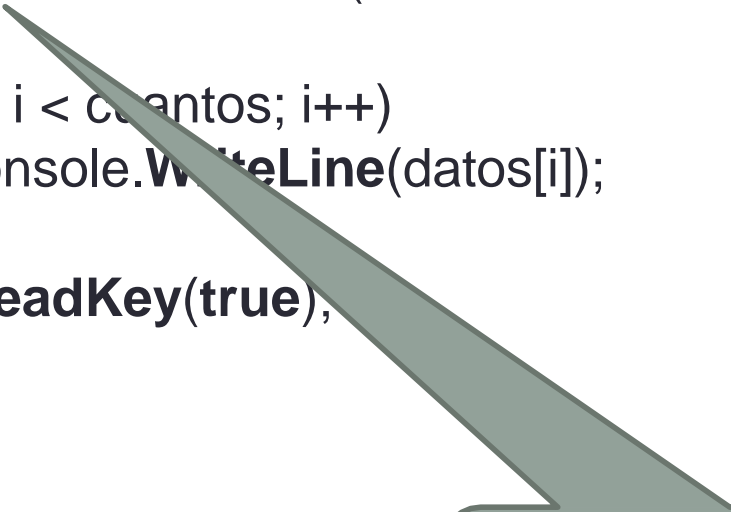
```
public static void Main(string[] args) {  
    int [] datos = new int [] {1,2,3,4,5,6,7,8,9,10};
```

```
    Console.WriteLine("¿Cuántos valores desea imprimir?");  
    int cuantos = Int32.Parse(Console.ReadLine());
```

```
    for(int i=0; i < cuantos; i++)  
        Console.WriteLine(datos[i]);
```

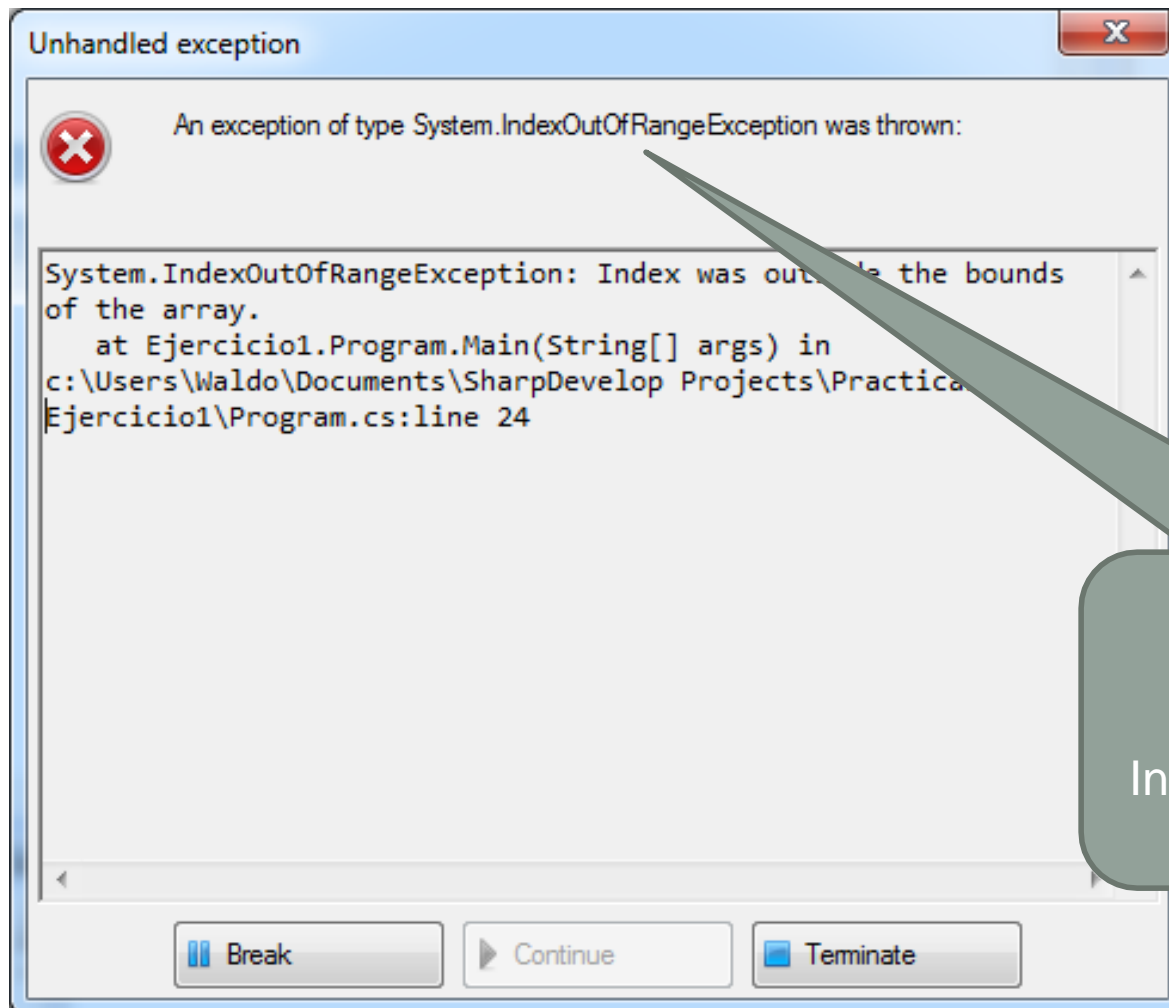
```
    Console.ReadKey(true),
```

```
}
```



¿Qué sucede si el usuario  
ingresa un valor mayor a 10?

# Excepción - Ejemplo



En este caso ocurre la  
excepción:  
IndexOutOfRangeException



# Excepciones

- Excepciones definidas por el framework:
- DivideByZeroException
- OverflowException
- InvalidCastException
- NullReferenceException
- IndexOutOfRangeException
- IO.IOException
- ... y muchas más

# Manejo de excepciones

- **Una excepción no controlada finaliza inmediatamente con la ejecución del programa.**
- C#, como otros lenguajes, provee un mecanismo para "capturar" excepciones y "hacer algo" cuando éstas ocurren.
- El poder "capturar" una excepción le permite al programador "manejar" la situación para que el programa siga ejecutándose con normalidad.

# Manejo de excepciones

```
try {  
  
}  
catch(){  
}  
catch(){  
}  
catch(){  
}  
finally{  
}
```

La estructura de control  
que permite manejar  
excepciones en C# se  
conoce como:  
try catch

# Manejo de excepciones

```
try {  
      
}
```

```
catch(){  
}
```

```
catch(){  
}
```

```
catch(){  
}
```

```
finally{  
}
```

Dentro del bloque try van las sentencias que se desean controlar. Aquellas que pueden producir una excepción.

# Manejo de excepciones

```
try {
```

```
}
```

```
catch(){
```

```
}
```

```
catch(){
```

```
}
```

```
catch(){
```

```
}
```

```
finally{
```

```
}
```

El bloque catch permite capturar y manejar las excepciones que ocurren en el bloque try.

Puede haber tantos bloques catch como se necesiten. Uno para cada tipo de excepción que se pueda producir. No es obligatorio, es decir puede no haber ningún bloque catch

# Manejo de excepciones

```
try {  
  
}  
catch(){  
}  
catch(){  
}  
catch(){  
}  
finally{  
}
```

El bloque finally es **único** y opcional. Este bloque se ejecuta **SIEMPRE**, independientemente de si hubo o no una excepción.

# Manejo de excepciones

## Estructuras válidas

**Un bloque try con  
más de un catch  
y un finally**

```
try {  
}  
catch(){  
}  
catch(){  
}  
finally{  
}
```

**Un bloque try con  
varios bloques  
catch**

```
try {  
}  
catch(){  
}  
catch(){  
}
```

**Un bloque try con  
un catch y un  
finally**

```
try {  
}  
catch(){  
}  
finally{  
}
```

# Manejo de excepciones

## Estructuras válidas

**Un bloque try con  
un único bloque  
catch**

```
try {  
}  
catch(){  
}
```

**Un bloque try un  
único bloque  
finally**

```
try {  
}  
finally{  
}
```



# Manejo de excepciones

## Estructuras NO válidas

**Un bloque try sin  
bloques catch ni  
bloques finally**

```
try {  
}
```

**El bloque finally siempre  
debe ir debajo de los  
bloques catch (si hubiera)**

```
try {  
}  
finally{  
}  
catch(){  
}
```

# Manejo de excepciones

```
try {
```

```
    try{  
    }  
    finally{  
    }
```

```
}
```

```
catch(){
```

```
}
```

```
finally{
```

```
}
```

Los bloques try pueden anidarse. Es decir, dentro de un bloque try puede haber otra estructura try-catch.

# Manejo de excepciones

```
public static void Main(string[] args) {  
    int a = 0, c = 1, i=0;  
    int [ ] b = new int [ ]{1};  
    try {  
        a = b[i] / c;  
        Console.WriteLine(a);  
    }  
    catch (Exception){  
        Console.WriteLine("Ha ocurrido un error");  
    }  
    Console.WriteLine("Fin");  
}
```

# Manejo de excepciones

```
public static void Main(string[] args) {  
    int a = 0, c = 1, i=0;  
    int [ ] b = new int [ ]{1};  
    try {  
        a = b[i] / c;  
        Console.WriteLine(a);  
    }  
    catch (Exception){  
        Console.WriteLine("Ha ocurrido un error");  
    }  
    Console.WriteLine("Fin");  
}
```

No se  
ejecuta

a = b[i] / c;  
Console.WriteLine(a);

Si las sentencias dentro del bloque try se ejecutan sin problema, el programa sigue su ejecución normalmente. En este caso el bloque catch **NO** se ejecuta.

# Manejo de excepciones

```
public static void Main(string[] args) {  
    int a = 0, c = 0, i = 0;  
    int [ ] b = new int [ ]{1};  
    try {  
        a = b[i] / c;  
        Console.WriteLine(a);  
    }  
    catch (Exception) {  
        Console.WriteLine("Ha ocurrido un error");  
    }  
    Console.WriteLine("Fin");  
}
```

En este caso, se producirá la excepción DivideByZeroException.

El bloque try deja de ejecutarse en este punto.

No se ejecuta

Luego se ejecuta el bloque catch.

Luego, el programa sigue ejecutándose normalmente,

# Manejo de excepciones

```
public static void Main(string[] args) {  
    int a = 0, c = 1, i = 1;  
    int [ ] b = new int [ ]{1};  
    try {  
        a = b[i] / c;  
        Console.WriteLine(a);  
    }  
    catch (Exception){  
        Console.WriteLine("Ha ocurrido un error");  
    }  
    Console.WriteLine("Fin");  
}
```

En este caso, se producirá la excepción `IndexOutOfRangeException`.

No se ejecuta

El bloque try deja de ejecutarse en este punto.

Luego se ejecuta el bloque catch.

Luego, el programa sigue ejecutándose normalmente,

# Manejo de excepciones

```
public static void Main(string[] args) {  
    int a = 0, c = 1, i=1;  
    int [ ] b = new int [ ]{1};  
    try {  
        a = b[i] / c;  
        Console.WriteLine(a);  
    }  
    catch (Exception){  
        Console.WriteLine("Ha ocurrido un error");  
    }  
    Console.WriteLine("Fin");  
}
```

En este ejemplo, al momento de atrapar una excepción, es imposible saber que excepción ocurrió.

Si se usa una cláusula catch sin parámetro se atrapa cualquier tipo de excepción, pero es imposible saber que tipo de excepción ocurrió.

# Manejo de excepciones

```
public static void Main(string[] args) {  
    int a = 0, c = 1, i=0;  
    int [ ] b = new int [ ]{1};  
    try {  
        a = b[i] / c;  
        Console.WriteLine(a);  
    }  
    catch(DivideByZeroException){  
        Console.WriteLine("Error de división por cero");  
    }  
    catch(IndexOutOfRangeException){  
        Console.WriteLine("Índice no válido");  
    }  
    Console.WriteLine("Fin");  
}
```



# Manejo de excepciones

```
public static void Main(string[] args) {  
    int a = 0, c = 0, i=0;  
    int [ ] b = new int [ ]{1};  
    try {
```

No se  
ejecuta

```
        a = b[i] / c;  
        Console.WriteLine(a);  
    }
```

```
    catch(DivideByZeroException){  
        Console.WriteLine("Error de división por cero");  
    }
```

```
    catch(IndexOutOfRangeException){  
        Console.WriteLine("Índice no válido");  
    }
```

```
    Console.WriteLine("Fin");  
}
```

Si la excepción que  
ocurre es del tipo  
DivideByZeroException  
se ejecuta este bloque  
catch

Luego, el programa sigue  
ejecutándose  
normalmente,

# Manejo de excepciones

```
public static void Main(string[] args) {  
    int a = 0, c = 0, i=1;  
    int [ ] b = new int [ ]{1};  
    try {  
        a = b[i] / c;  
        Console.WriteLine(a);  
    }  
    catch(DivideByZeroException){  
        Console.WriteLine("Error de división por cero");  
    }  
    catch(IndexOutOfRangeException){  
        Console.WriteLine("Índice no válido");  
    }  
    Console.WriteLine("Fin");  
}
```

No se ejecuta

Si la excepción que ocurre es del tipo `IndexOutOfRangeException` se ejecuta este bloque catch

Luego, el programa sigue ejecutándose normalmente,

# Manejo de excepciones

```
public static void Main(string[] args)
{
    int a = 0, c = 0, i = 0;
    int [ ] b = new int [ ]{1};
    try {
        a = b[i] / c;
        Console.WriteLine(a);
    }
    catch(IndexOutOfRangeException){
        Console.WriteLine("Índice no válido");
    }
    Console.WriteLine("Fin");
}
```

En este caso ocurrirá la excepción `DivideByZeroException`. Pero no hay un manejador para esta excepción.

# Manejo de excepciones

```
public static void Main(string[] args) {  
    int a = 0, c = 0, i = 0;  
    int [ ] b = new int [ ]{1};  
    try {  
        a = b[i] / c;  
        Console.WriteLine(a);  
    }  
    catch(IndexOutOfRangeException){  
        Console.WriteLine("Índice no válido");  
    }  
    Console.WriteLine("Fin");  
}
```

No se  
ejecuta

En este caso la  
excepción causa el fin  
de la ejecución del  
Programa.

# Manejo de excepciones

```
public static void Main(string[] args) {  
    int a = 0, c = 1, i=0;  
    int [ ] b = new int [ ]{1};  
    while (true) {  
        try {  
            a = b[i] / c; Console.WriteLine(a);  
            break;  
        }  
        catch(DivideByZeroException){  
            Console.WriteLine("Error de división por cero");  
        }  
        catch(IndexOutOfRangeException){  
            Console.WriteLine("Índice no válido");  
        }  
        catch (Exception) {  
            Console.WriteLine("Ha ocurrido otro error");  
        }  
    }  
    Console.WriteLine("Fin");  
}
```

En algunos casos resulta útil atrapar tipos de excepciones específicas y tener un manejador general para cualquier otro tipo de excepción que pueda ocurrir.

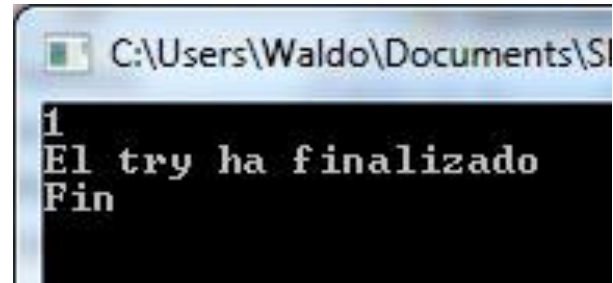
# Manejo de excepciones

- Si se codifica la cláusula `finally`, la misma se **ejecuta SIEMPRE**, independientemente de si se produjo una excepción o no.
- Se ejecuta incluso si la excepción no se pudo manejar.
- La cláusula `finally` se ejecuta aún cuando en la función exista un `return` dentro del `try` o de un `catch`.

# Manejo de excepciones

```
public static void Main(string[] args) {  
    int a = 0, c = 1, i=0;  
    int [ ] b = new int [ ]{1};  
    try {  
        a = b[i] / c;  
        Console.WriteLine(a);  
    }  
    finally{  
        Console.WriteLine("El try ha finalizado");  
    }  
    Console.WriteLine("Fin");  
}
```

Sin excepciones ni  
cláusula catch



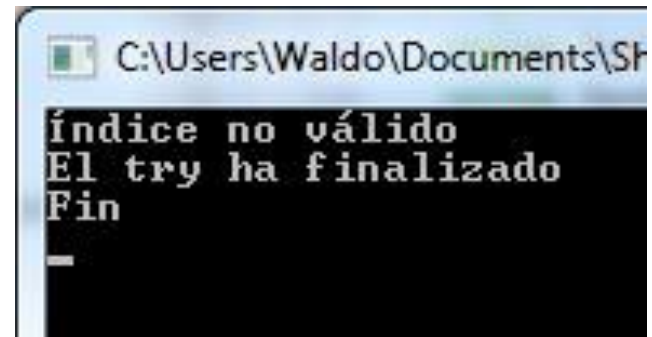
C:\Users\Waldo\Documents\SI

```
1  
El try ha finalizado  
Fin
```

# Manejo de excepciones

```
public static void Main(string[] args) {  
    int a = 0, c = 1, i=1;  
    int [ ] b = new int [ ]{1};  
    try {  
        a = b[i] / c;  
        Console.WriteLine(a);  
    }  
    catch(DivideByZeroException){  
        Console.WriteLine("Error de división");  
    }  
    catch(IndexOutOfRangeException){  
        Console.WriteLine("Índice no válido");  
    }  
    finally{  
        Console.WriteLine("El try ha finalizado");  
    }  
    Console.WriteLine("Fin");  
}
```

Con excepciones y  
con cláusula catch



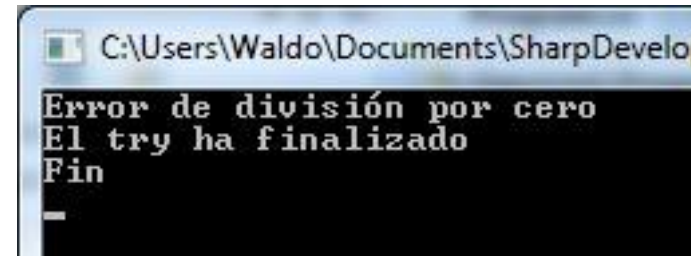


# Manejo de excepciones

```
public static void Main(string[] args) {  
    int a = 0, c = 0, i=0;  
    int[ ] b = new int[ ]{1};  
    try {  
        a = b[i] / c;  
        Console.WriteLine(a);  
        return;  
    }  
    catch(DivideByZeroException){  
        Console.WriteLine("Error de división por cero");  
    }  
    catch(IndexOutOfRangeException){  
        Console.WriteLine("Índice no válido");  
    }  
    finally{  
        Console.WriteLine("El try ha finalizado");  
    }  
    Console.WriteLine("Fin");  
}
```

No se  
ejecuta

Con excepciones  
y catch



```
C:\Users\Waldo\Documents\SharpDevelo  
Error de división por cero  
El try ha finalizado  
Fin  
-
```

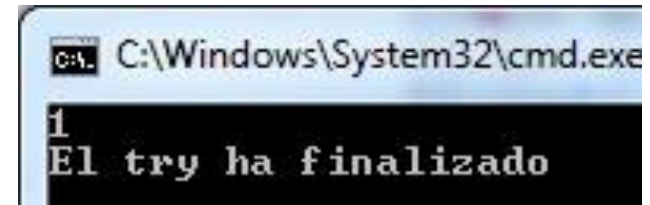
# Manejo de excepciones

```
public static void Main(string[] args) {  
    int a = 0, c = 1, i=0;  
    int [ ] b = new int [ ]{1};  
    try {  
        a = b[i] / c;  
        Console.WriteLine(a);  
        return;  
    }  
    catch(DivideByZeroException){  
        Console.WriteLine("Error de división por cero");  
    }  
    catch(IndexOutOfRangeException){  
        Console.WriteLine("Índice no válido");  
    }  
    finally{  
        Console.WriteLine("El try ha finalizado");  
    }  
    Console.WriteLine("Fin");  
}
```

El  
return  
hace  
que la  
función  
finalice  
en este  
punto

No se  
ejecuta

Sin excepciones y con  
return en el try



```
C:\Windows\System32\cmd.exe  
1  
El try ha finalizado
```

# Propagación de excepciones

- Si en una función ocurre una excepción que no es manejada, esta se propaga a quién la invocó.
- Solo cuando la propagación llega a la función Main y no es manejada, el programa finaliza su ejecución con error.

# Propagación de excepciones

```
public static int dividir(int t, int u){  
    int r = t / u;  
    return r;  
}  
  
public static void Main(string[] args) {  
    int a = 0, c = 0, i = 0;  
    int [ ] b = new int [ ]{1};  
    try {  
        a = dividir(b[i], c);  
        Console.WriteLine(a);  
    }  
    catch(DivideByZeroException){  
        Console.WriteLine("Error de división por cero");  
    }  
    catch(IndexOutOfRangeException){  
        Console.WriteLine("Índice no válido");  
    }  
    Console.WriteLine("Fin");  
}
```

La excepción  
DivideByZeroException  
no es manejada dentro  
de esta función

# Propagación de excepciones

```
public static int dividir(int t, int u){  
    int r = t / u;  
    return r;  
}  
public static void Main(string[] args) {  
    int a = 0, c = 0, i = 0;  
    int [ ] b = new int [ ]{1};  
    try {  
        a = dividir(b[i], c);  
        Console.WriteLine(a);  
    }  
    catch(DivideByZeroException){  
        Console.WriteLine("Error de división por cero");  
    }  
    catch(IndexOutOfRangeException){  
        Console.WriteLine("Índice no válido");  
    }  
    Console.WriteLine("Fin");  
}
```

No se  
ejecuta

Si ocurre, la función  
deja de ejecutarse y la  
excepción se propaga a  
la función que hizo la  
invocación.

# Propagación de excepciones

```
public static int dividir(int t, int u){  
    int r = t / u;  
    return r;  
}
```

```
public static void Main(string[] args) {  
    int a = 0, c = 0, i = 0;  
    int [ ] b = new int [ ]{1};  
    try {
```

```
        a = dividir(b[i], c);  
        Console.WriteLine(a);  
    }
```

```
    catch(DivideByZeroException){  
        Console.WriteLine("Error de división por cero");  
    }
```

```
    catch(IndexOutOfRangeException){  
        Console.WriteLine("Índice no válido");  
    }
```

```
    Console.WriteLine("Fin");  
}
```

No se  
ejecuta

Dentro de Main si existe  
un manejador para la  
excepción  
DivideByZeroException

Luego el  
programa sigue  
su ejecución  
normalmente

# Excepciones definidas por el usuario

- Una excepción en realidad es un objeto de la clase `Exception`.
- En C#, la clase `Exception` es la superclase para cualquier excepción.
- C# permite la definición de nuevas excepciones.
- Para ello hay que crear subclases de `Exception`.

# Excepciones definidas por el usuario

- Definición: se define como cualquier subclase.
- En general por convención el nombre de la clase termina con la palabra “Exception”

```
public class MiPropiaException : Exception  
{  
}
```

- El nuevo tipo de excepción creada se puede usar en cualquier cláusula catch

```
try { ... }  
catch (MiPropiaException) { ... }
```

Dentro del catch se codifican las acciones que queremos que se lleven a cabo si se produce esta excepción



# Excepciones definidas por el usuario

Las excepciones creadas por el usuario deben ser levantadas o disparadas por el propio usuario cuando sea necesario.

Para ello se usa la siguiente sintaxis:



```
throw new MiPropiaException();
```

El comando  
throw dispara  
la excepción.

Crea una instancia de la  
clase  
**MiPropiaException**

# Excepciones definidas por el usuario

```
public class NoEsVocalException : Exception { }  
public static void Main(string[] args) {  
    string vocal;  
    while (true)  
    {    try {  
        Console.WriteLine("Ingrese una vocal");  
        vocal = Console.ReadLine();  
        if((vocal != "A") && (vocal != "E") && (vocal != "I") && ....  
            throw new NoEsVocalException();  
        else break;  
    }  
    catch(NoEsVocalException){  
        Console.WriteLine("No ingresó una vocal.  
            Intentelo nuevamente"); }  
    }  
    Console.WriteLine("Fin");}
```

¿Qué hace este programa?

# Excepciones definidas por el usuario

```
public static void Main(string[] args) {  
    ...  
    try {  
        Usuario u;  
        u.recuperarUsuario(numero);  
        u.validarPassword(password);  
        u.login();  
    }  
    catch(UsuarioInexistenteException){  
        Console.WriteLine("El usuario no existe en la base de datos");  
    }  
    catch>PasswordInvalidoException){  
        Console.WriteLine("El número de usuario con la contraseña no coinciden");  
    }  
    catch(CuentaDeshabilitadaException){  
        Console.WriteLine("La cuenta del usuario fue deshabilitada");  
    }  
    catch(MultipleAccesoException){  
        Console.WriteLine("El usuario ya ingresó al sistema");  
    }  
    catch(DataBaseException){  
        Console.WriteLine("No es posible conectarse con la base de datos");  
    }  
    ...  
}
```

# Puesta en común

Dadas las clases *Alumno* y *Aula*, implemente en esta última un método público llamado *ingresaAlumno(Alumno a)* que reciba una instancia de la clase *Alumno* y lo almacene en la variable *alumnos*. La clase *aula* tiene una capacidad limitada.

En un main crear un aula y simular la inscripción de varios alumnos. Si al momento de ingresar un alumno, la cantidad de inscriptos es mayor o igual a la *capacidad* del aula se debe levantar la excepción *AulaLlena*.  
Definir el manejador de la excepción.

```

public class Aula{
    private ArrayList alumnos;
    private int capacidad;

    public Aula ()
    { alumnos=new ArrayList();
      capacidad=30;
    }
    public void ingresaAlumno(Alumno a)
    {
        alumnos.Add(a);
    }
    public int cantidadInscriptos()
    { return alumnos.Count;
    }
    public int Capacidad
    { set {capacidad=value;}
      get {return capacidad;}
    } }

```

```

public class Alumno{
    private int dni,legajo;
    private string nomape;

    public Alumno (int doc, int leg, string nom) {
        dni=doc;
        legajo=leg;
        nomape=nom;
    }
}

```

```

public class AulaLlena : Exception {}

```

```

public static void main(){
    Aula a = new Aula();
    int docu, lega;
    string name;
    try {
        while (true){
            .....
            if (a.cantidadInscriptos() < a.Capacidad )
                a.ingresaAlumno(new Alumno(docu,lega,name));
            else
                throw new AulaLlena();
            .....
        }
    }
    catch(AulaLlena){
        Console.WriteLine("Cupo insuficiente"); } }

```

Vamos ingresando alumnos hasta agotar el cupo, en ese momento se dispara la excepción que es atendida por AulaLlena.

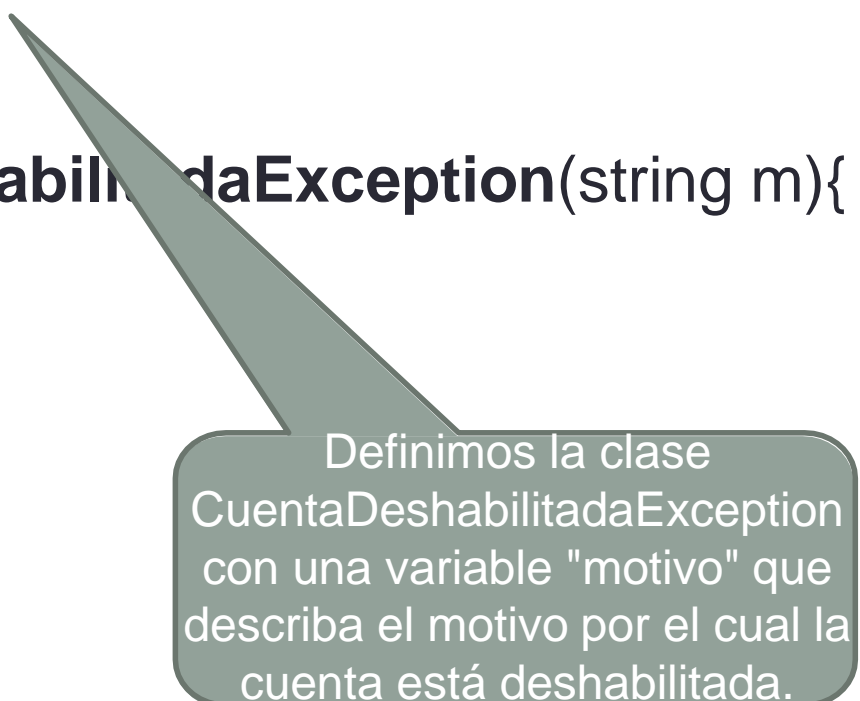
# Enviando información con las excepciones

- Muchas veces resulta útil devolver información extra al momento de levantar una excepción.
- Para hacer esto se debe ampliar el estado de la clase que representa la excepción. Se le agrega una variable de instancia y se agrega un constructor.

# Enviando información con las excepciones

```
public class CuentaDeshabilitadaException : Exception
{
    public string motivo;

    public CuentaDeshabilitadaException(string m){
        motivo = m;
    }
}
```




Definimos la clase CuentaDeshabilitadaException con una variable "motivo" que describa el motivo por el cual la cuenta está deshabilitada.



# Enviando información con las excepciones

```
class CuentaDeshabilitadaException : Exception
{
    public string motivo;

    public CuentaDeshabilitadaException(string m){
        motivo = m;
    }
}
```



El "motivo" es pasado al constructor de la clase en el momento en que ocurre la excepción.

# Enviando información con las excepciones

**Al momento de levantar la excepción se le pasa el motivo como parámetro.**

```
public static void Main() {  
    public string razon = "Usuario moroso";  
  
    try { ...  
        throw new CuentaDeshabilitadaException(razon)  
    }  
  
    catch (CuentaDeshabilitadaException e){  
        Console.WriteLine(e.motivo);  
    }  
}
```


**La instancia es creada.**

**El capturador la atrapa y es asignada a la variable e**

**Se envía por parámetro el motivo de la excepción**

# Enviando información con las excepciones

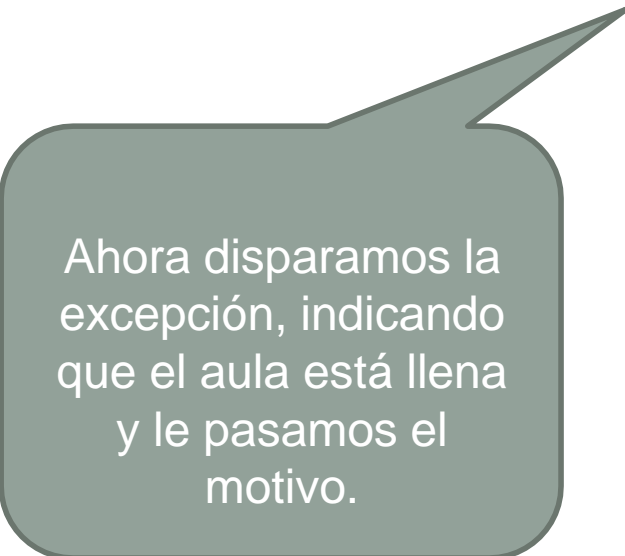
```
public static void Main() {  
    public string razon = "Usuario moroso";  
  
    try { ...  
        throw new CuentaDeshabilitadaException(razon)  
    }  
  
    catch (CuentaDeshabilitadaException e){  
  
        Console.WriteLine(e.motivo);  
    }  
}
```



**Luego se puede utilizar  
motivo, ya que es un  
miembro público**

## Puesta en común: enviando información a la excepción

```
if (a.CantidadInscriptos() < a.Capacidad )  
    a. ingresaAlumno(new Alumno(docu,lega,name));  
else  
    throw new AulaLlena("Cupo insuficiente");
```



Ahora disparamos la excepción, indicando que el aula está llena y le pasamos el motivo.

# Cambia la definición de la clase AulaLlena....

```
public class AulaLlena : Exception{  
    public string motivo;  
  
    public AulaLlena (string m){  
        motivo = m;  
    }  
}
```

```
public static void main(){
    Aula a = new Aula();
    int docu, lega;
    string name;
    try {
        while (true){
            .....
            if (a.cantidadInscriptos() a.Capacidad )
                a. ingresaAlumno(new Alumno(docu,lega,name));
            else
                throw new AulaLlena("Cupo insuficiente");
            ..... }
        }
    catch(AulaLlena ele){
        Console.WriteLine(ele.motivo); }
}
```

