DIGITAL ARTS & ENTERTAINMENT

TOOL DEVELOPMENT

C# INHERITANCE & INTERFACES

A DIGITAL ARTS AND ENTERTAINMENT PRESENTATION

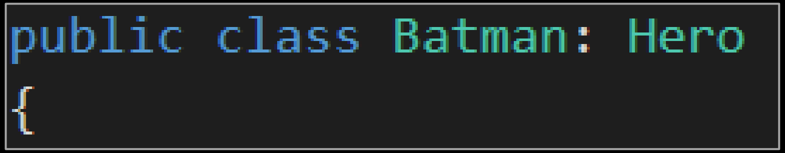
INHERITANCE

C# SYNTAX

c# syntax: inheritance

INHERITANCE BASICS

- Struct = value type → **no** inheritance
- Class = reference type → inheritance
  - in fact, every class implicitely inherits System.object



```
public class Batman: Hero
{
```

- Batman → Hero → System.object

- Restricted in C#: you can only inherit from 1 class!
  - no multiple inheritance
  - (part of) solution: interfaces

    we will get to that in a moment☺

c# syntax: inheritance

WHAT IS INHERITED?

- Fields and properties
  - public/protected fields & props are inherited

```
public class Batman: Hero
{
```

- Methods
  - public/protected methods are inherited
- Constructors
  - **not** inherited!
  - but can be accessed through base;
    - "base" gives you access to the base Class

```
public Batman(int level) : base(level)
```

➢ calls Hero's constructor

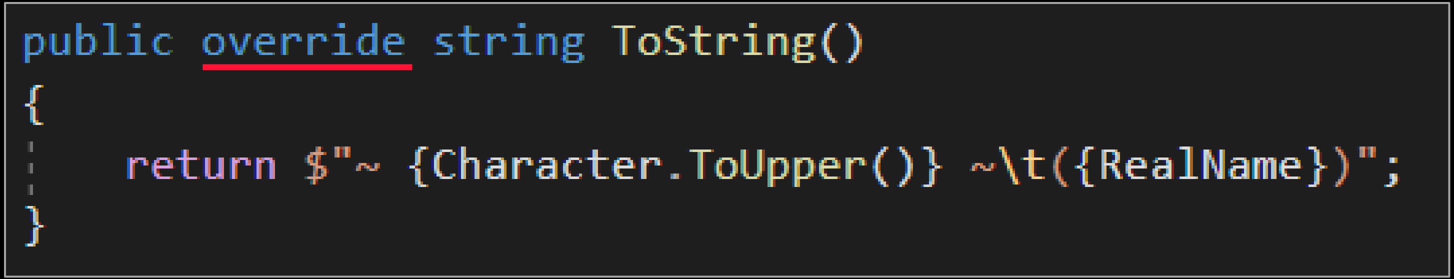
  - in fact, base class constructor must be called
    if the base class does not have a default constructor!

```
CS7036: There is no argument given that corresponds to the required formal parameter 'level' of 'Hero.Hero(int)'
```

c# syntax: inheritance

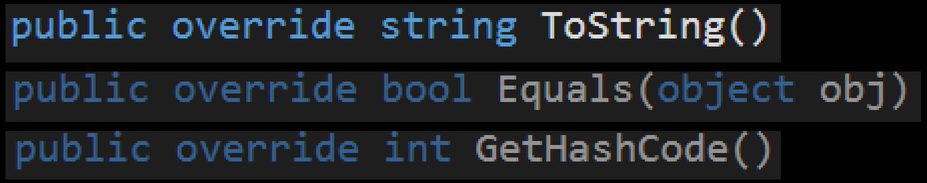
METHOD OVERRIDING

- Remember ToString()?



```
public override string ToString()
{
    return $"~ {Character.ToUpper()} ~\t({RealName})";
}
```

  - ToString() is virtual in Sytem.object,
  - therefore, we can override it

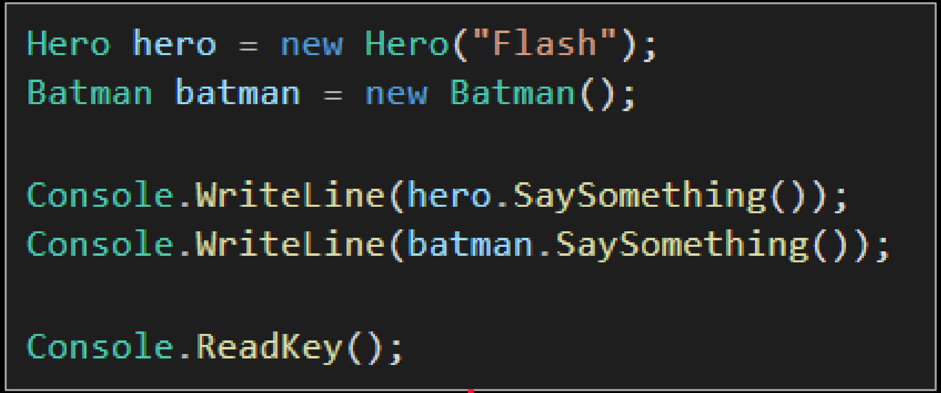


```
public override string ToString()
public override bool Equals(object obj)
public override int GetHashCode()
```

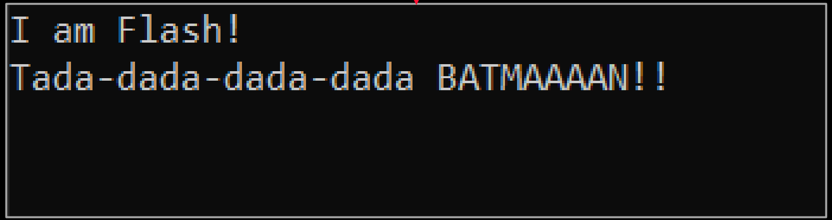
- Declare as virtual in the base class: `public virtual string SaySomething()`
- Override in the inheriting class: `public override string SaySomething()`
- What if you don't override it? → default behavior in base class

INHERITANCE: EXERCISE

- Given Hero class
  - Add the given Hero class to your project
    - Place it in a Model folder!
    - Change the namespace to yours
  - Add a method called SaySomething():
    - Its default behavior is to return "I am *[Hero Name]*!"
    - Make sure it can be overridden
- Create a Batman class
  - Inherit correctly from Hero
  - When SaySomething() is called on this class, it should return "Tada-dada-dada-dada BATMAAAAN!"
- Test using the given test code
  - Your classes should match the code, not vice versa ☺



```
Hero hero = new Hero("Flash");
Batman batman = new Batman();

Console.WriteLine(hero.SaySomething());
Console.WriteLine(batman.SaySomething());

Console.ReadKey();
```



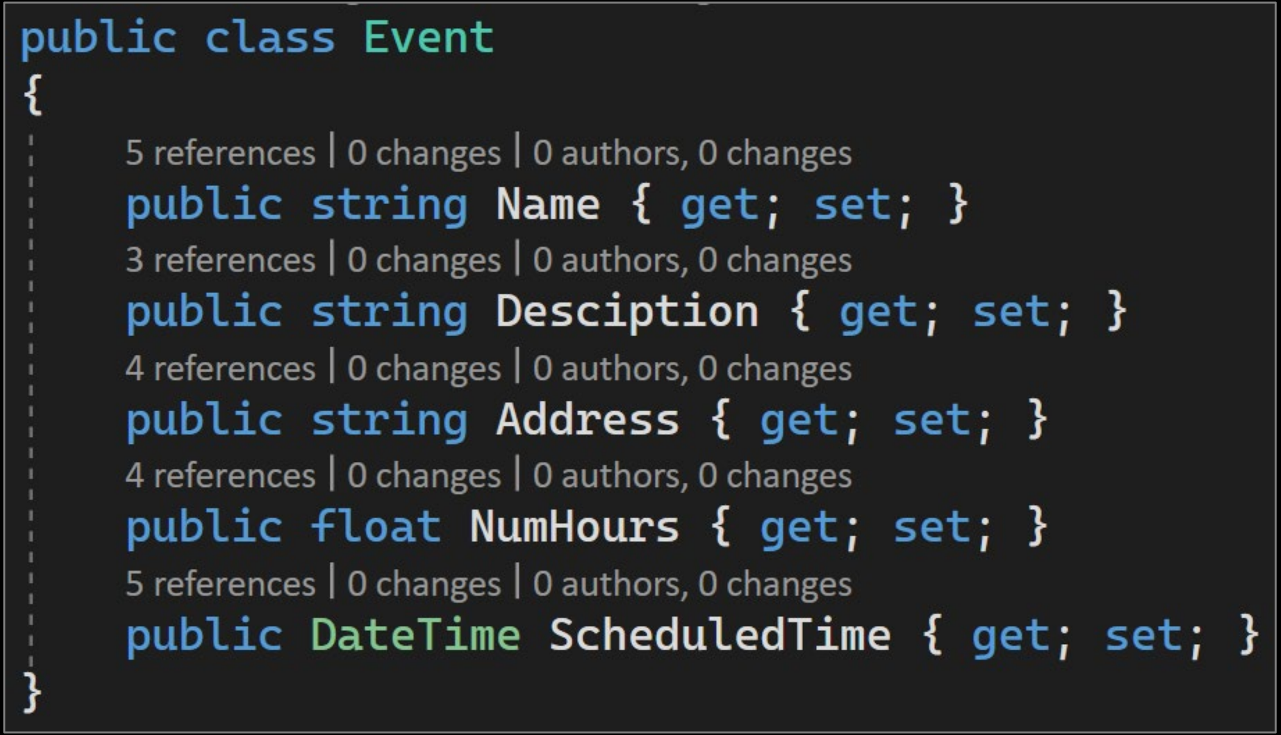
```
I am Flash!
Tada-dada-dada-dada BATMAAAAN!!
```

INTERFACES - EXAMPLE 1

C# SYNTAX

INTERFACES
EXAMPLE 1: CLASS

Situation: we have a class called Event and a list of Event instances:

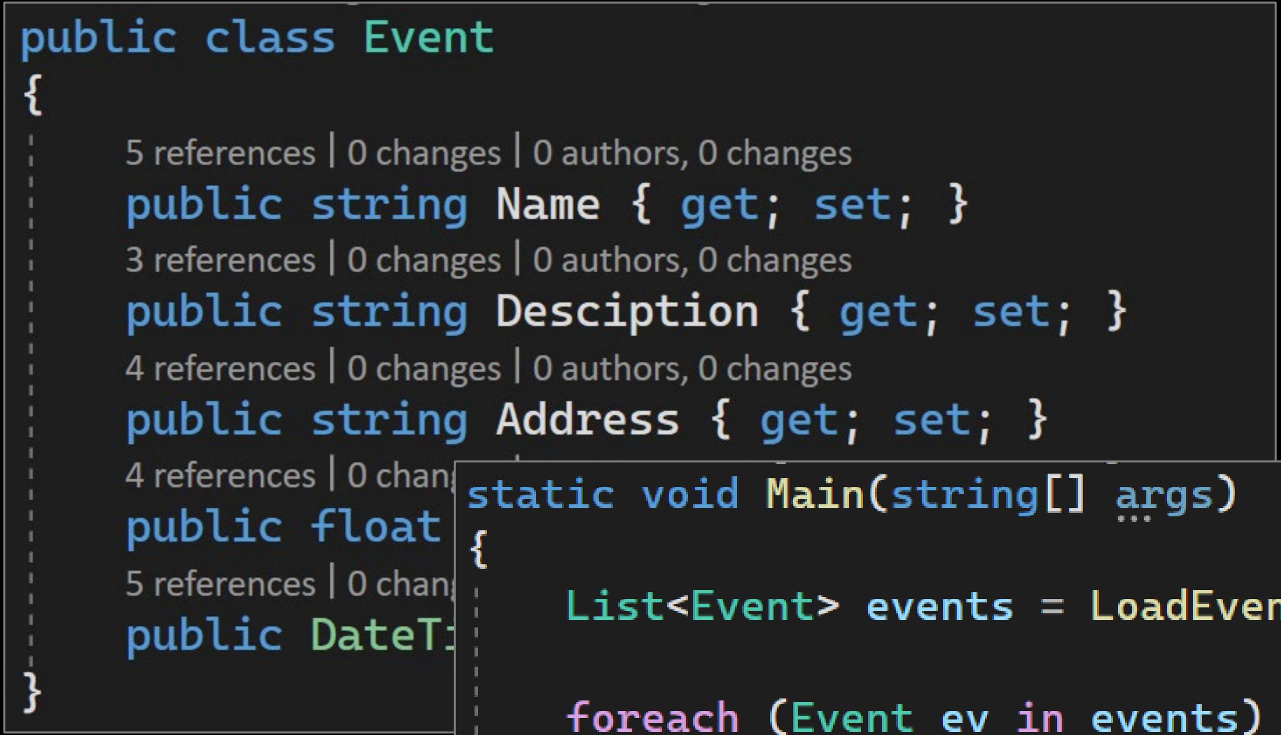


```
public class Event
{
        5 references | 0 changes | 0 authors, 0 changes
        public string Name { get; set; }
        3 references | 0 changes | 0 authors, 0 changes
        public string Desciption { get; set; }
        4 references | 0 changes | 0 authors, 0 changes
        public string Address { get; set; }
        4 references | 0 changes | 0 authors, 0 changes
        public float NumHours { get; set; }
        5 references | 0 changes | 0 authors, 0 changes
        public DateTime ScheduledTime { get; set; }
}
```

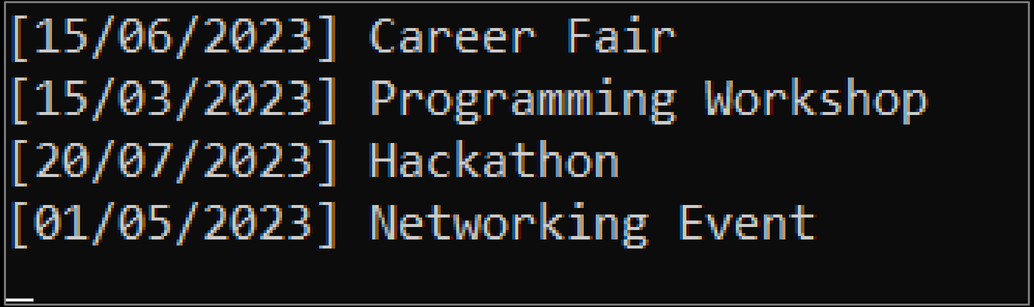
INTERFACES

EXAMPLE 1: LIST OF EVENTS

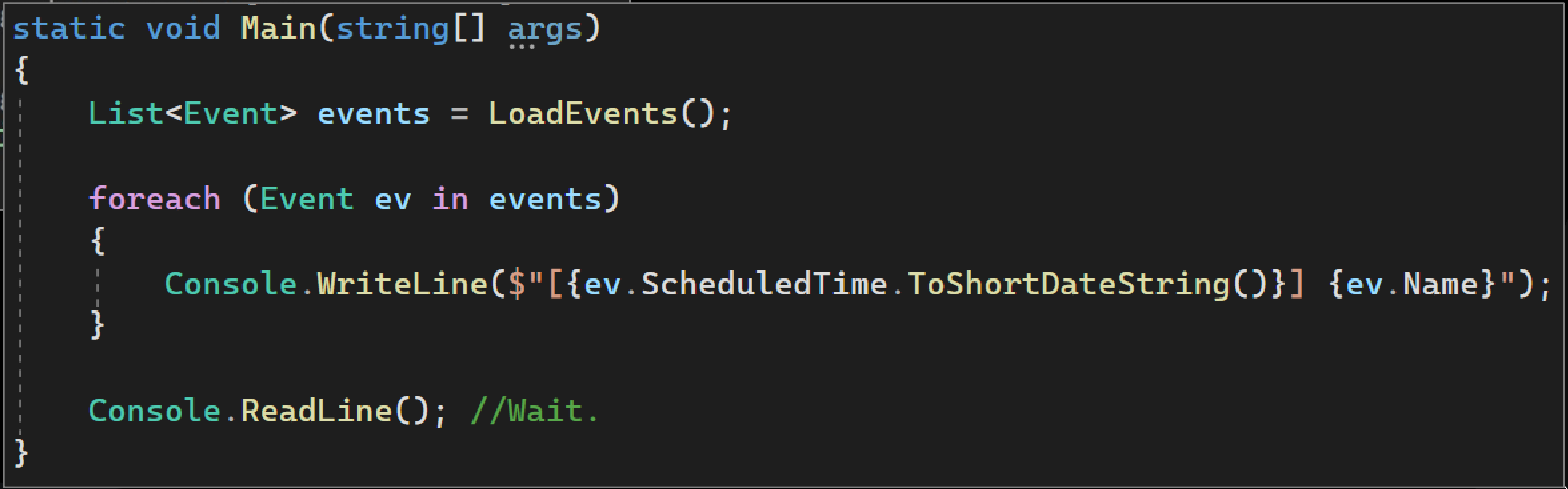
Situation: we have a class called Event and a list of Event instances:



```
public class Event
{
        5 references | 0 changes | 0 authors, 0 changes
        public string Name { get; set; }
        3 references | 0 changes | 0 authors, 0 changes
        public string Desciption { get; set; }
        4 references | 0 changes | 0 authors, 0 changes
        public string Address { get; set; }
        4 references | 0 chan
        public float
        5 references | 0 chan
        public DateTi
}
```



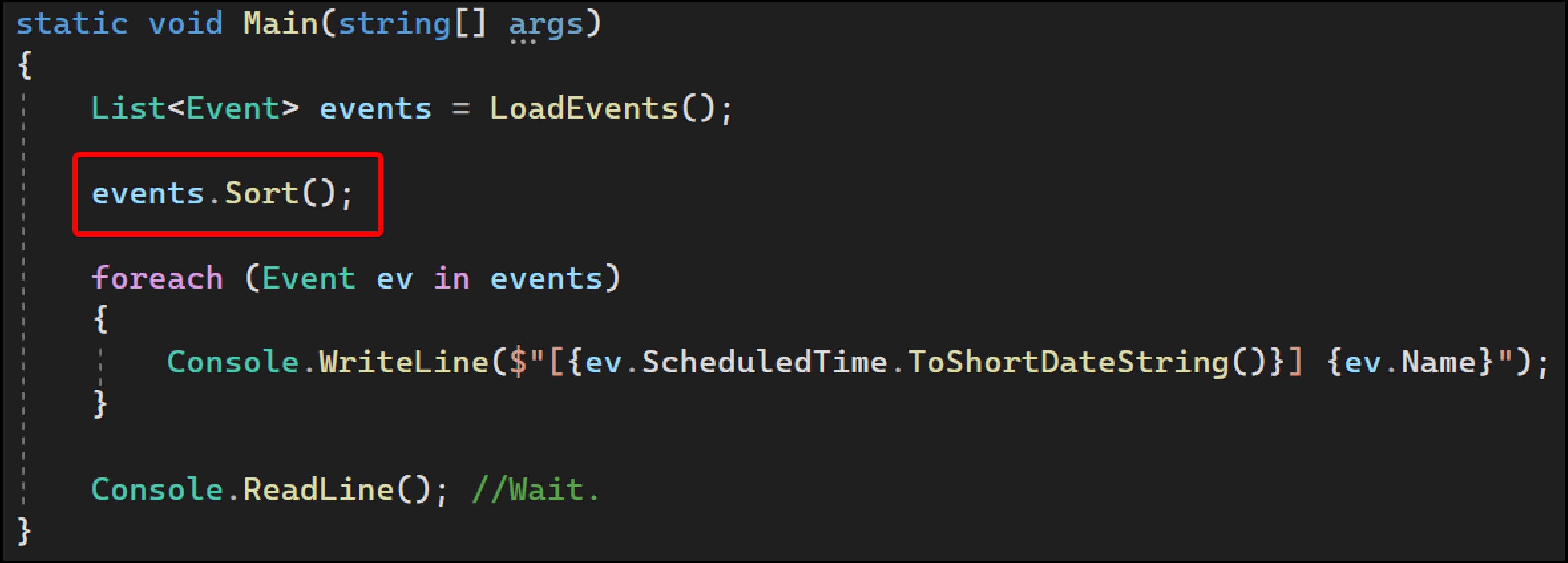
```
[15/06/2023] Career Fair
[15/03/2023] Programming Workshop
[20/07/2023] Hackathon
[01/05/2023] Networking Event
_
```



```
static void Main(string[] args)
{
    List<Event> events = LoadEvents();

    foreach (Event ev in events)
    {
        Console.WriteLine($"[{ev.ScheduledTime.ToShortDateString()}] {ev.Name}");
    }

    Console.ReadLine(); //Wait.
}
```

EXAMPLE 1: SORT LIST OF EVENTS

UPDATE: We want to **sort** our events:



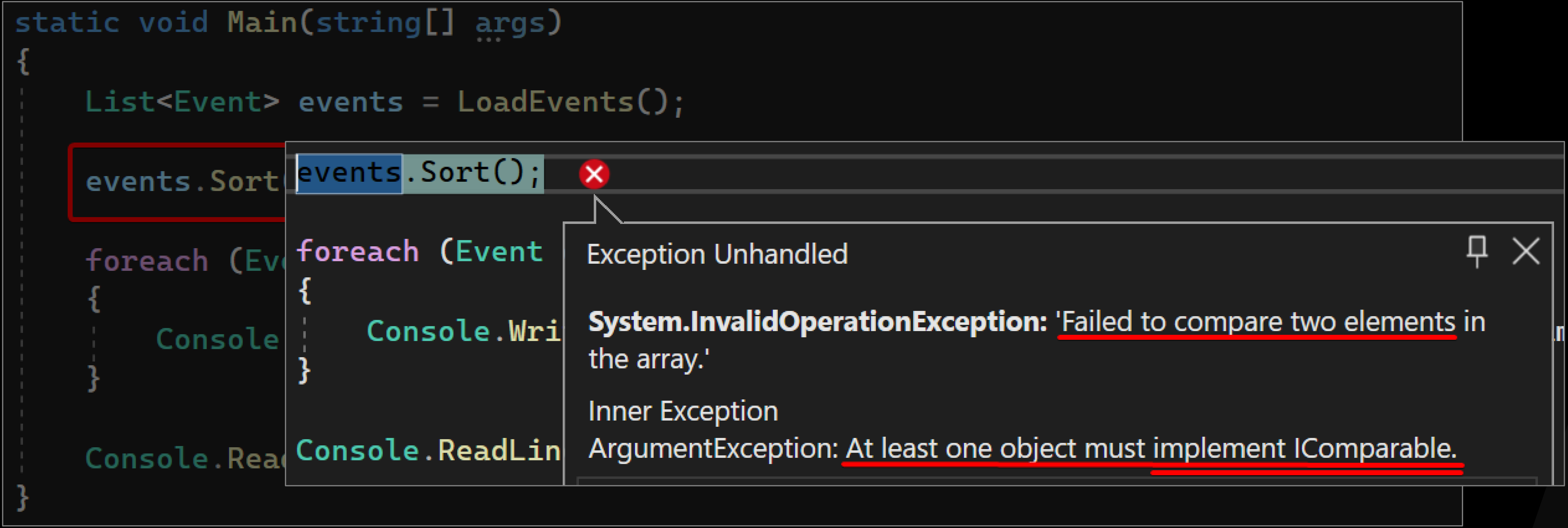
```
static void Main(string[] args)
{
    List<Event> events = LoadEvents();

    events.Sort();

    foreach (Event ev in events)
    {
        Console.WriteLine($"[{ev.ScheduledTime.ToShortDateString()}] {ev.Name}");
    }

    Console.ReadLine(); //Wait.
}
```

- ➢ So... Based on what will these events be sorted? Name? Date? ...??

INTERFACES

EXAMPLE 1: SORT LIST OF EVENTS

ANSWER: It will **crash!**



```
static void Main(string[] args)
{
    List<Event> events = LoadEvents();

    events.Sort();

    foreach (Event
    {
        Console.Wri
    }

    Console.ReadLin
}
```

Exception Unhandled

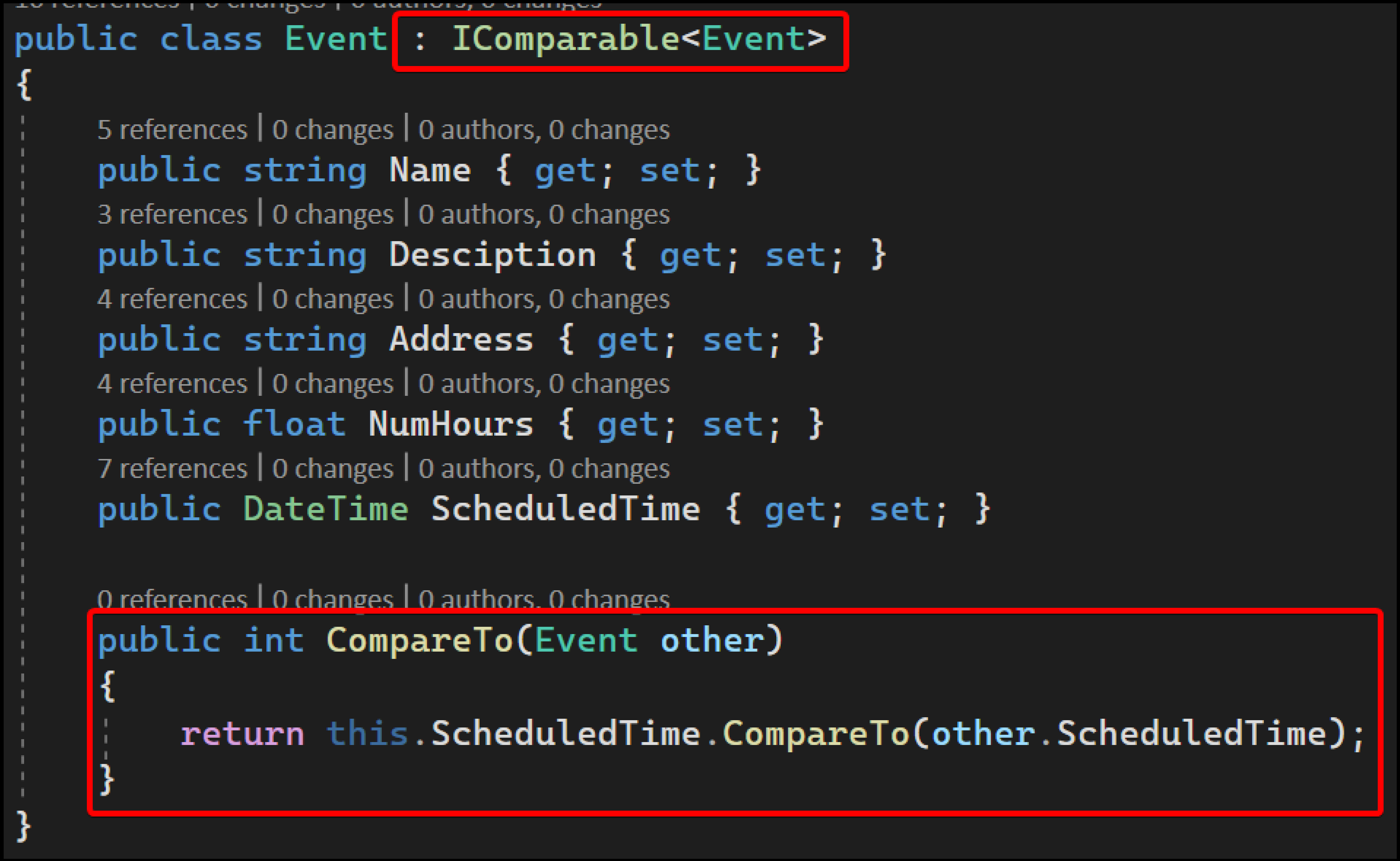
System.InvalidOperationException: 'Failed to compare two elements in the array.'

Inner Exception
ArgumentException: At least one object must implement IComparable.

- Runtime error. It does not know how to compare the items.
- The answer to the problem is literally there: IComparable **interface**

INTERFACES

EXAMPLE 1: SOLUTION

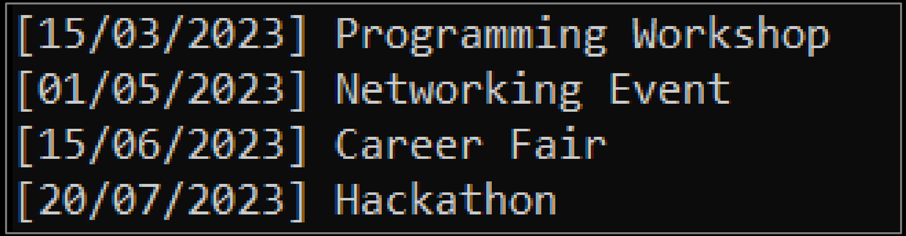


```
public class Event : IComparable<Event>
{
        5 references | 0 changes | 0 authors, 0 changes
        public string Name { get; set; }
        3 references | 0 changes | 0 authors, 0 changes
        public string Desciption { get; set; }
        4 references | 0 changes | 0 authors, 0 changes
        public string Address { get; set; }
        4 references | 0 changes | 0 authors, 0 changes
        public float NumHours { get; set; }
        7 references | 0 changes | 0 authors, 0 changes
        public DateTime ScheduledTime { get; set; }

        0 references | 0 changes | 0 authors, 0 changes
        public int CompareTo(Event other)
        {
            return this.ScheduledTime.CompareTo(other.ScheduledTime);
        }
}
```

- When **implementing** the interface we are forced to implement the **CompareTo** function!
- It allows us to choose how events are ordered (here: scheduled time)

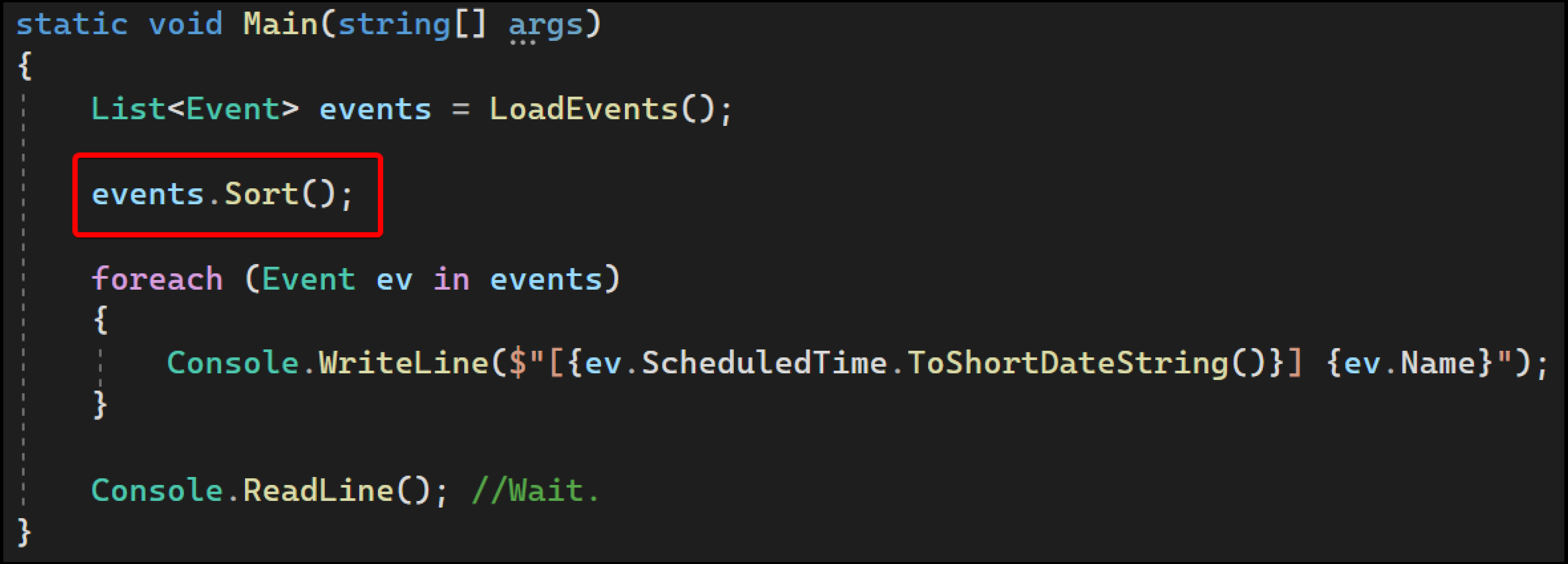
INTERFACES

EXAMPLE 1: RESULT



```
[15/03/2023] Programming Workshop
[01/05/2023] Networking Event
[15/06/2023] Career Fair
[20/07/2023] Hackathon
```

UPDATE: We want to **sort** our events:

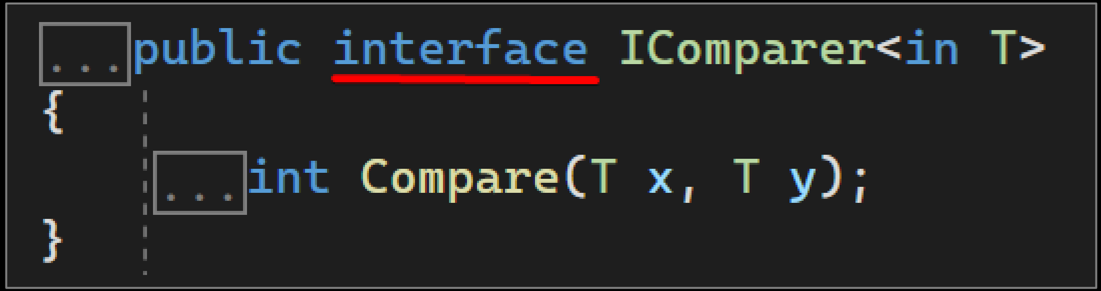


```
static void Main(string[] args)
{
    List<Event> events = LoadEvents();

    events.Sort();

    foreach (Event ev in events)
    {
        Console.WriteLine($"[{ev.ScheduledTime.ToShortDateString()}] {ev.Name}");
    }

    Console.ReadLine(); //Wait.
}
```

- Events are **sorted** based on their Scheduled Time

EXAMPLE 1: HOW DOES IT WORK?

1. ICompare interface



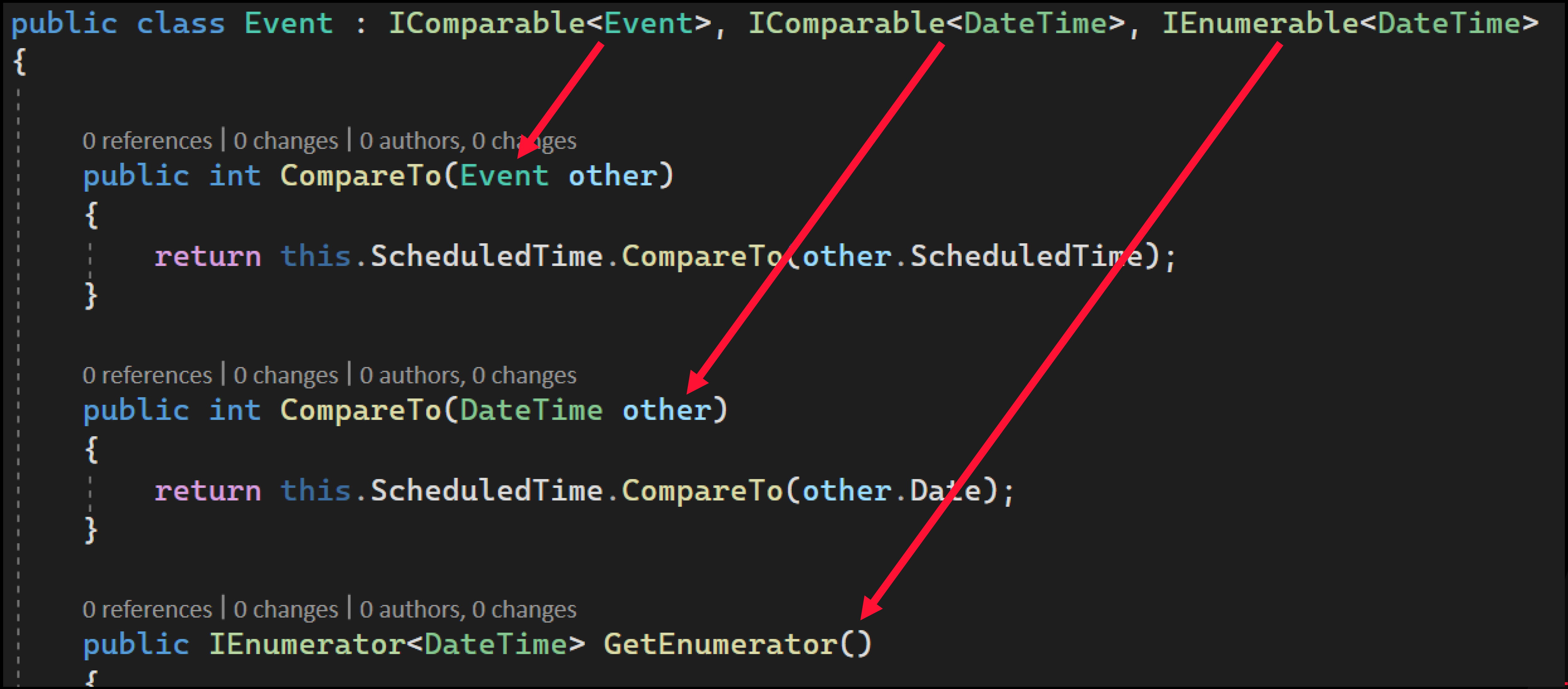
```
...public interface IComparer<in T>
{
    ...int Compare(T x, T y);
}
```

- ➢ The interface has a Compare function “header”; not implemented!
- ➢ Every class that implements it **must** implement this function! (it is like signing a contract)

2. When sorting, the List class will **call** the Compare function on the object. This is only possible **if** the object implements ICompare!
   - ➢ Therefore, it crashes if it does not.

INTERFACES

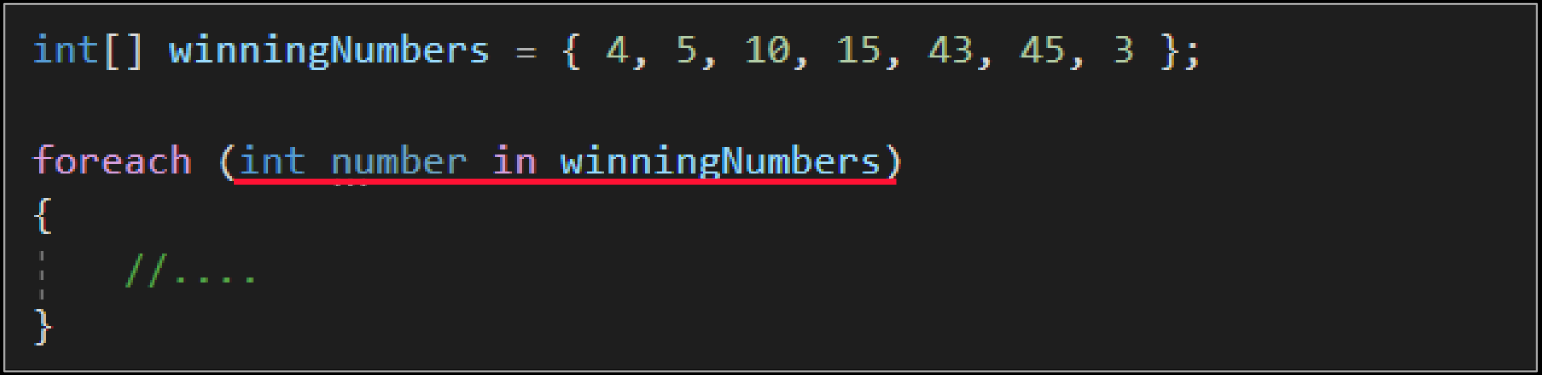
EXAMPLE 1: IMPLEMENTING MULTIPLE INTERFACES



```
public class Event : IComparable<Event>, IComparable<DateTime>, IEnumerable<DateTime>
{

    0 references | 0 changes | 0 authors, 0 changes
    public int CompareTo(Event other)
    {
        return this.ScheduledTime.CompareTo(other.ScheduledTime);
    }

    0 references | 0 changes | 0 authors, 0 changes
    public int CompareTo(DateTime other)
    {
        return this.ScheduledTime.CompareTo(other.Date);
    }

    0 references | 0 changes | 0 authors, 0 changes
    public IEnumerator<DateTime> GetEnumerator()
    {
```

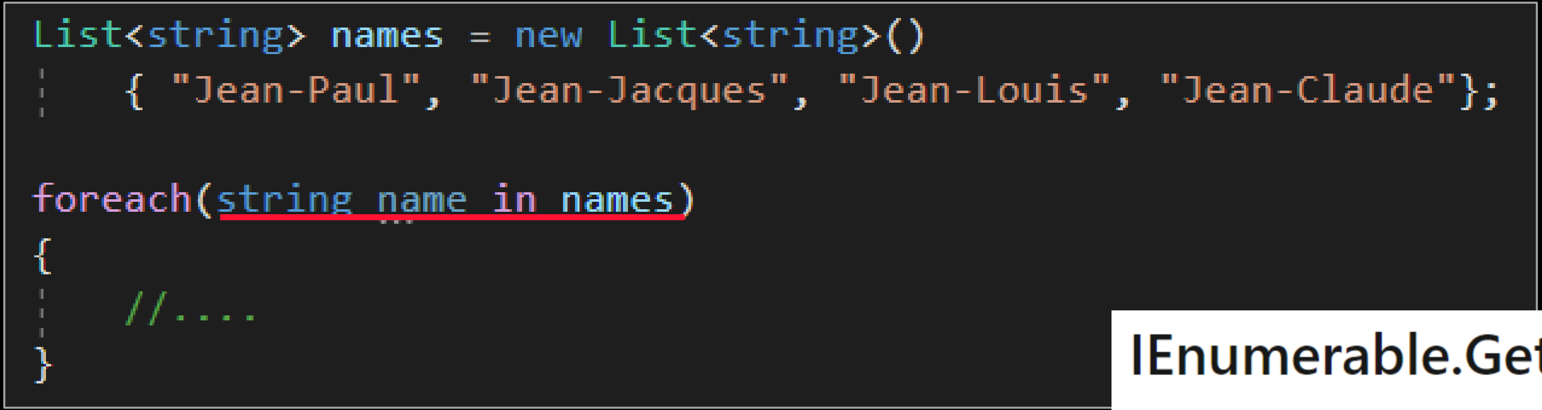
c# syntax: interfaces

EXISTING INTERFACE EXAMPLE C#



```
int[] winningNumbers = { 4, 5, 10, 15, 43, 45, 3 };

foreach (int number in winningNumbers)
{
    //....
}
```

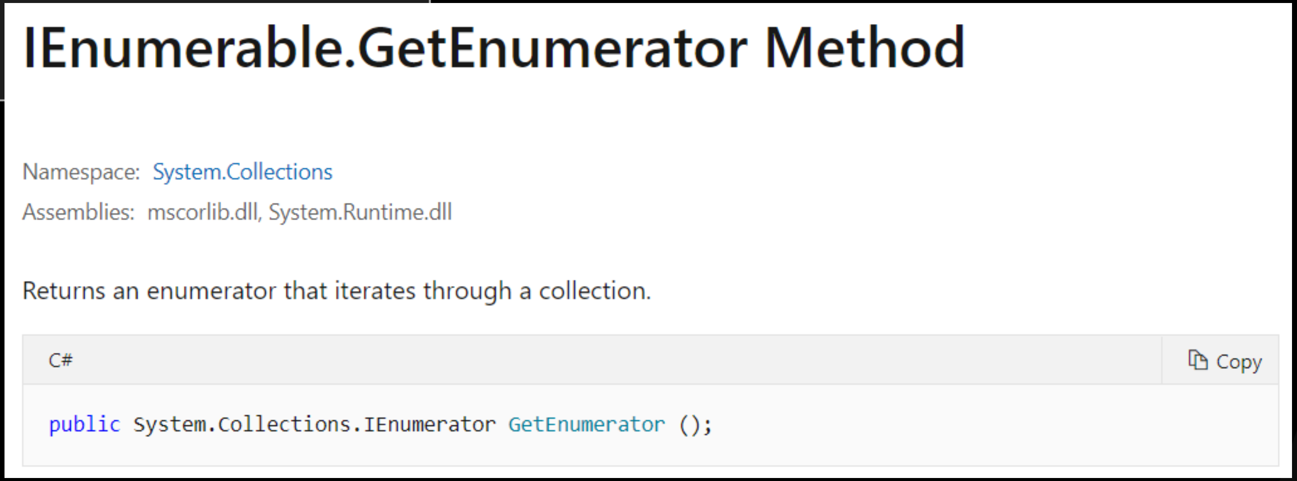
➢ int[] (implicitely) implements IEnumberable<int>



```
List<string> names = new List<string>()
    { "Jean-Paul", "Jean-Jacques", "Jean-Louis", "Jean-Claude"};

foreach(string name in names)
{
    //....
}
```

➢ List<string> implements IEnumberable<string>

➢ IEnumberable<T> interface
- allows to loop over objects



IEnumerable.GetEnumerator Method

Namespace: System.Collections

Assemblies: mscorlib.dll, System.Runtime.dll

Returns an enumerator that iterates through a collection.

```
public System.Collections.IEnumerator GetEnumerator ();
```

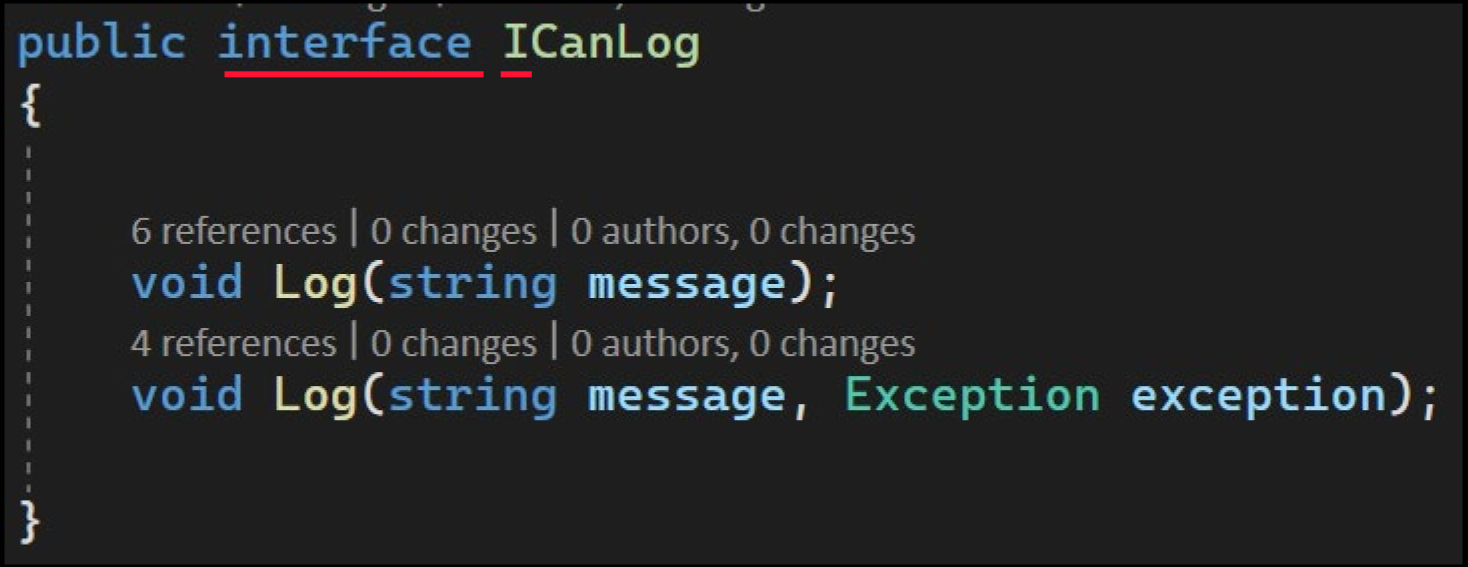
INTERFACES – EXAMPLE 2

CREATING A CUSTOM INTERFACE

c# syntax: interfaces

INTERFACES – WHAT

- "Class" that only contains signatures of
  - Methods
  - Properties
  - Events
  - Indexers



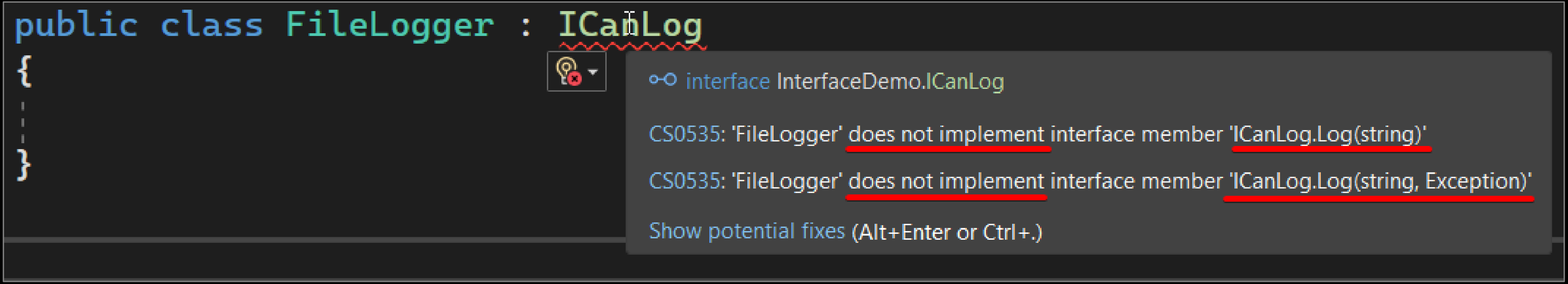
```
public interface ICanLog
{

    6 references | 0 changes | 0 authors, 0 changes
    void Log(string message);
    4 references | 0 changes | 0 authors, 0 changes
    void Log(string message, Exception exception);

}
```

- An interface has **no** implementations
  - *(except when a method is declared static, then it must be implemented)*
  - *(from C# 8.0 on, default implementation for data members is allowed)*

- Can be compared to a pure virtual class in C++

c# syntax: interfaces

INTERFACES – WHAT & WHY

- C# does not allow multiple inheritance,
  but you can implement multiple interfaces!
- Implementing an interface is like entering a **contract**:
  - Every member/method/.. **must** be implemented!
  - 

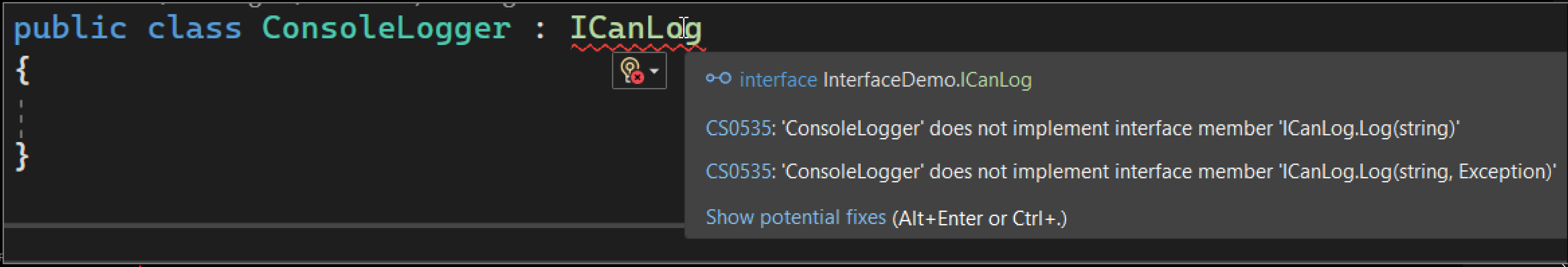
```
public class FileLogger : ICanLog
{

}
```

interface InterfaceDemo.ICanLog

CS0535: 'FileLogger' does not implement interface member 'ICanLog.Log(string)'

CS0535: 'FileLogger' does not implement interface member 'ICanLog.Log(string, Exception)'

Show potential fixes (Alt+Enter or Ctrl+.)



```
public class ConsoleLogger : ICanLog
{

}
```

interface InterfaceDemo.ICanLog

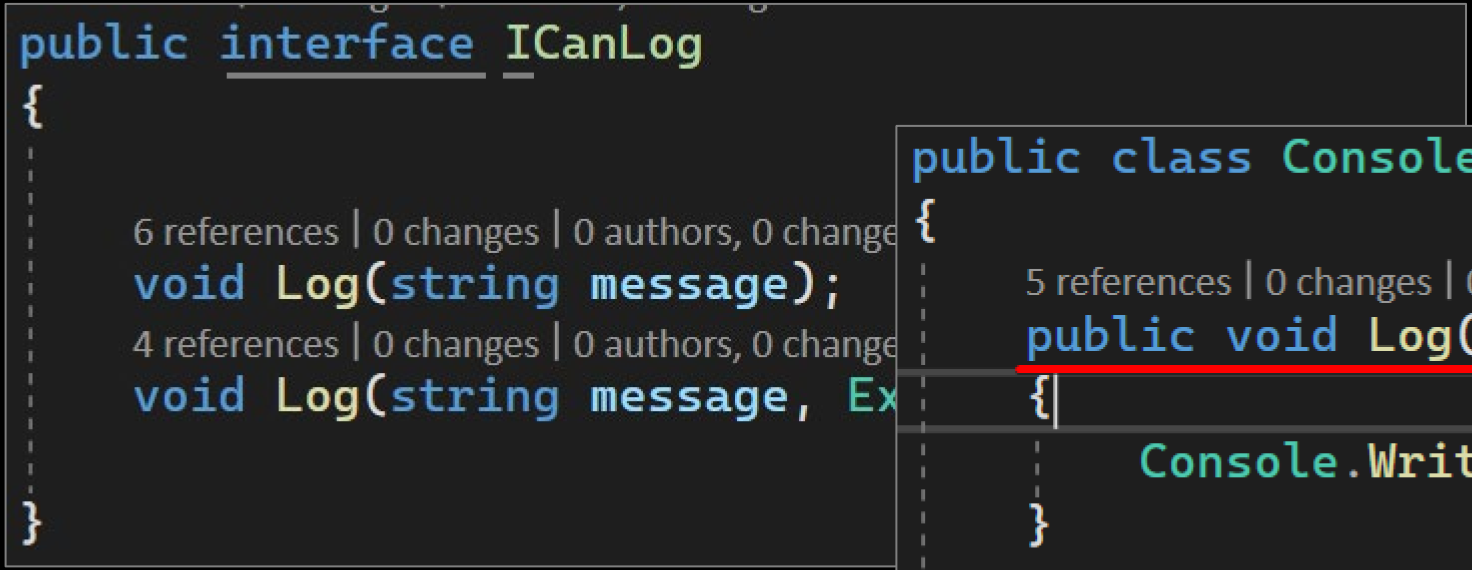
CS0535: 'ConsoleLogger' does not implement interface member 'ICanLog.Log(string)'

CS0535: 'ConsoleLogger' does not implement interface member 'ICanLog.Log(string, Exception)'

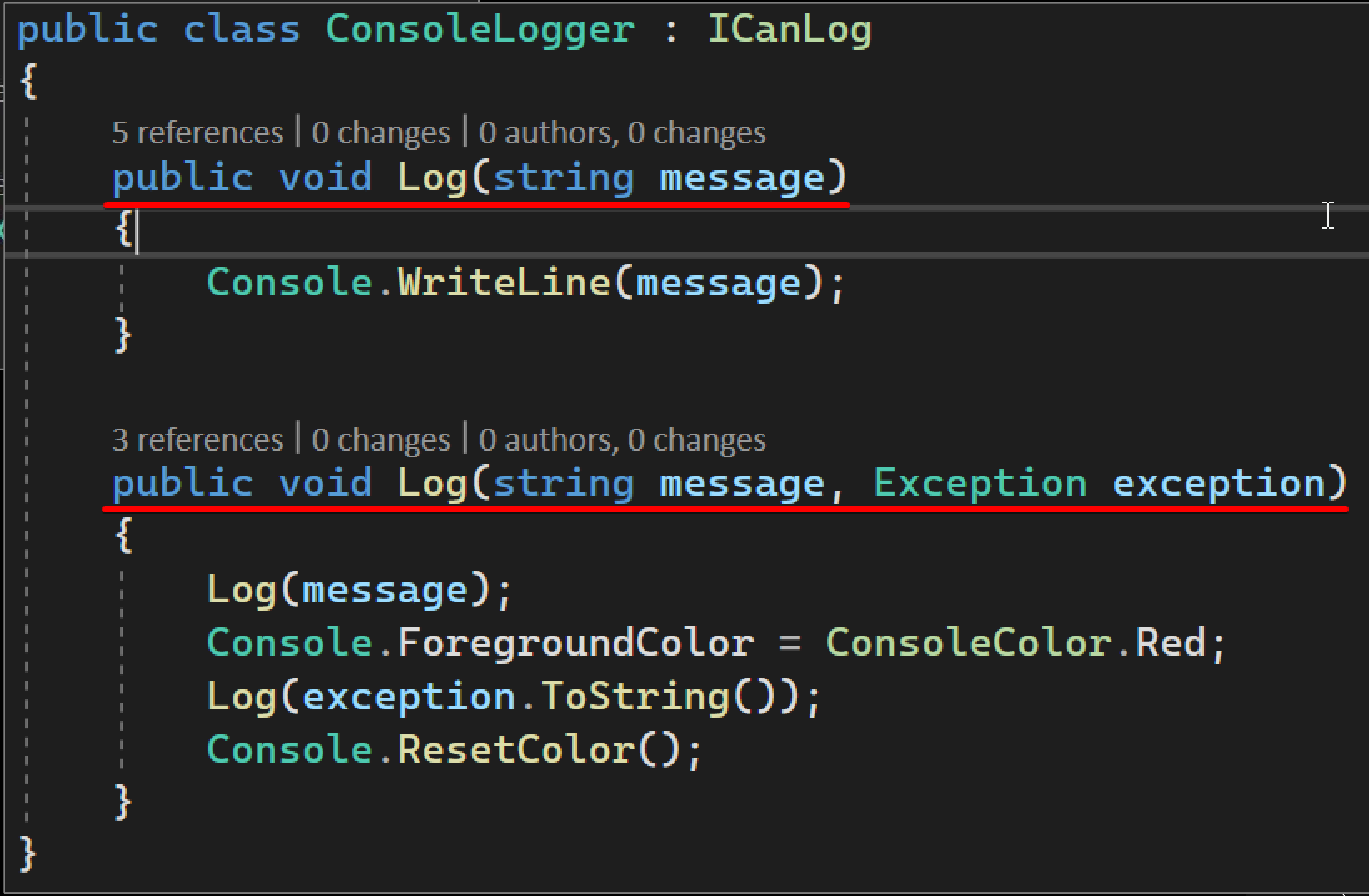
Show potential fixes (Alt+Enter or Ctrl+.)

c# syntax: interfaces

INTERFACES – WHAT & WHY: LOG TO CONSOLE



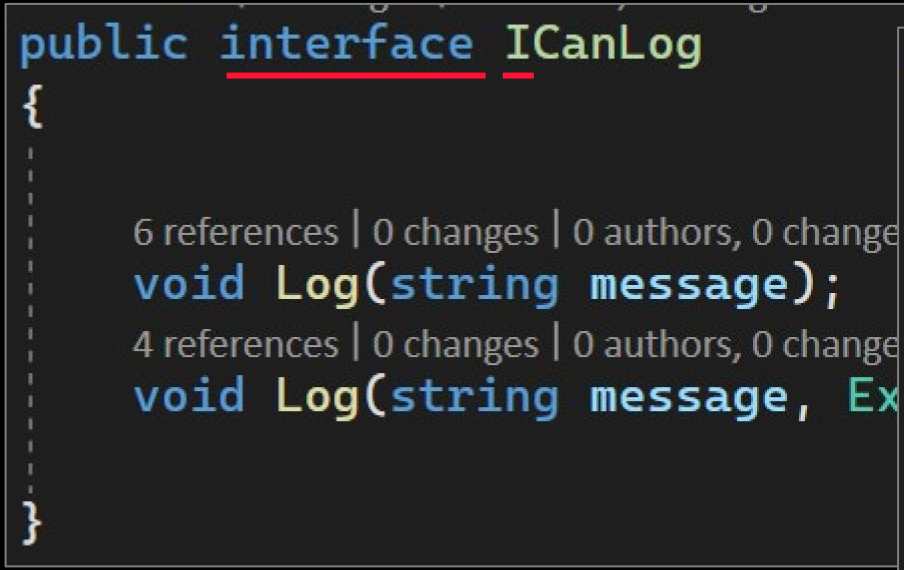
```
public interface ICanLog
{

    6 references | 0 changes | 0 authors, 0 change
    void Log(string message);
    4 references | 0 changes | 0 authors, 0 change
    void Log(string message, Ex

}
```



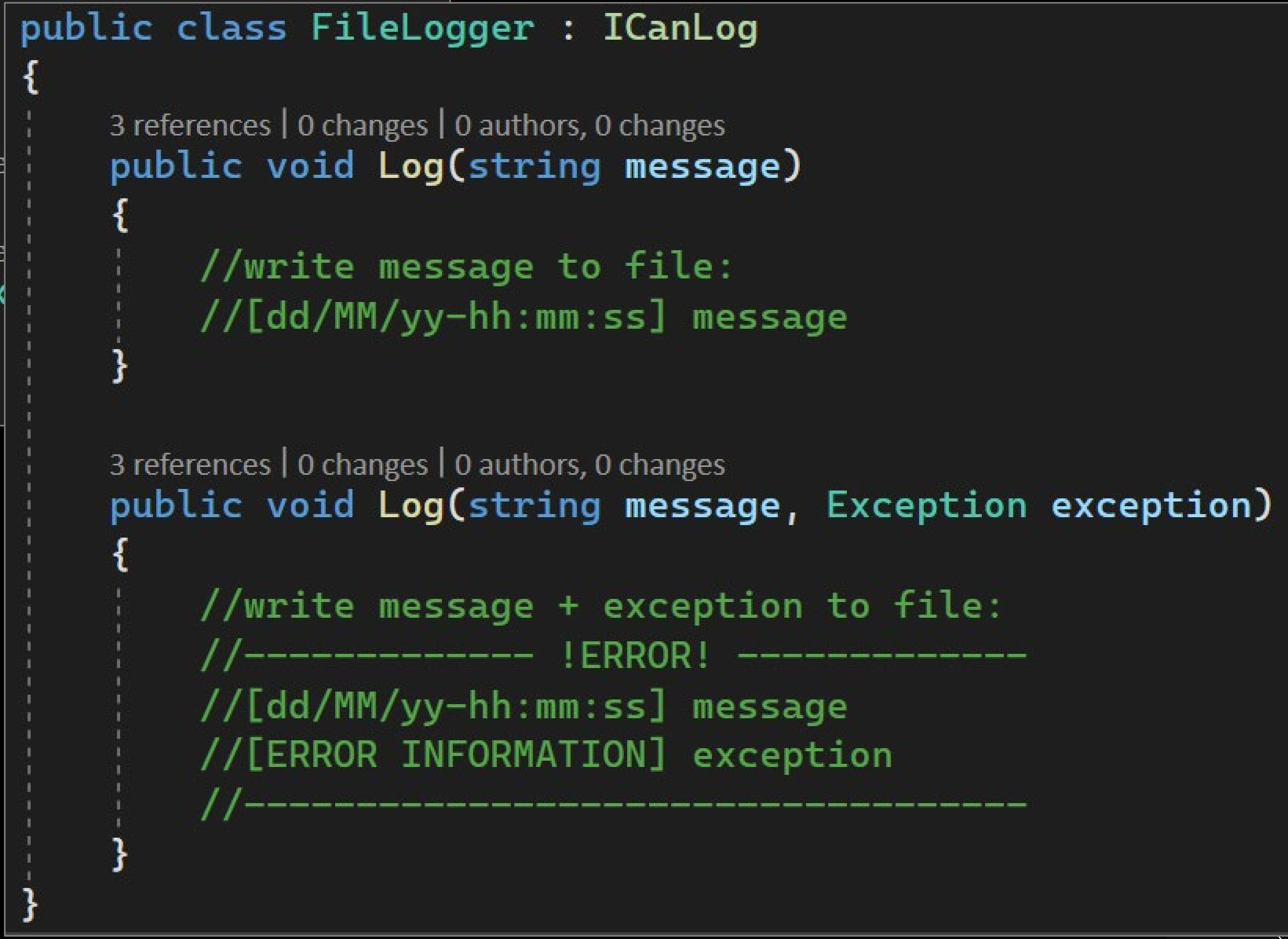
```
public class ConsoleLogger : ICanLog
{
    5 references | 0 changes | 0 authors, 0 changes
    public void Log(string message)
    {
        Console.WriteLine(message);
    }

    3 references | 0 changes | 0 authors, 0 changes
    public void Log(string message, Exception exception)
    {
        Log(message);
        Console.ForegroundColor = ConsoleColor.Red;
        Log(exception.ToString());
        Console.ResetColor();
    }
}
```

c# syntax: interfaces

INTERFACES – WHAT & WHY: LOG TO FILE



```
public interface ICanLog
{

    6 references | 0 changes | 0 authors, 0 change
    void Log(string message);
    4 references | 0 changes | 0 authors, 0 change
    void Log(string message, Ex

}
```

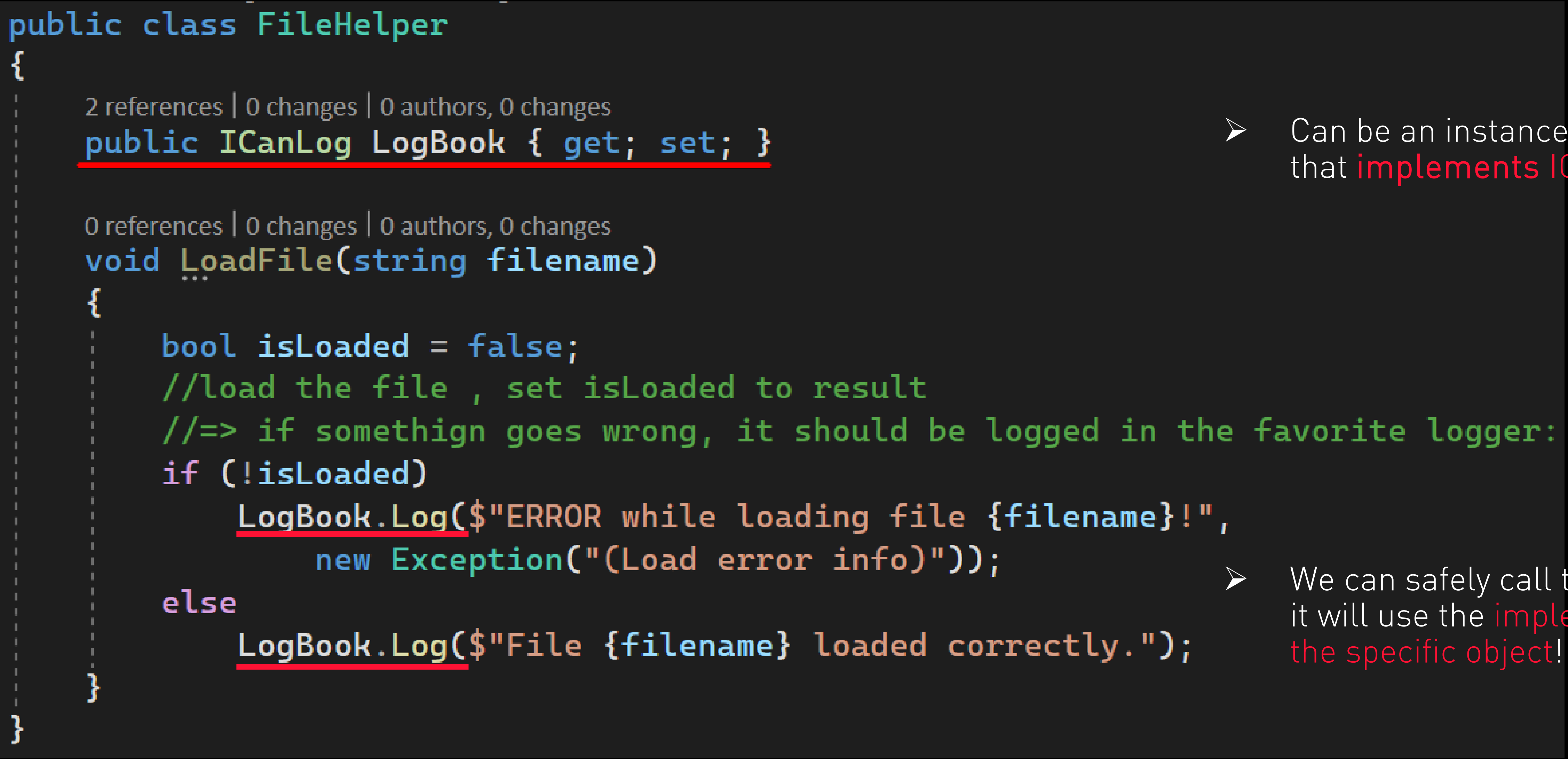


```
public class FileLogger : ICanLog
{
    3 references | 0 changes | 0 authors, 0 changes
    public void Log(string message)
    {
        //write message to file:
        //[dd/MM/yy-hh:mm:ss] message
    }

    3 references | 0 changes | 0 authors, 0 changes
    public void Log(string message, Exception exception)
    {
        //write message + exception to file:
        //------------- !ERROR! -------------
        //[dd/MM/yy-hh:mm:ss] message
        //[ERROR INFORMATION] exception
        //---------------------------------
    }
}
```

c# syntax: interfaces

INTERFACES – WHAT & WHY (USAGE)

- Creating / using an object that implements the Interface



```
public class FileHelper
{
    2 references | 0 changes | 0 authors, 0 changes
    public ICanLog LogBook { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    void LoadFile(string filename)
    {
        bool isLoaded = false;
        //load the file , set isLoaded to result
        //=> if somethign goes wrong, it should be logged in the favorite logger:
        if (!isLoaded)
            LogBook.Log($"ERROR while loading file {filename}!",
                new Exception("(Load error info)"));
        else
            LogBook.Log($"File {filename} loaded correctly.");
    }
}
```

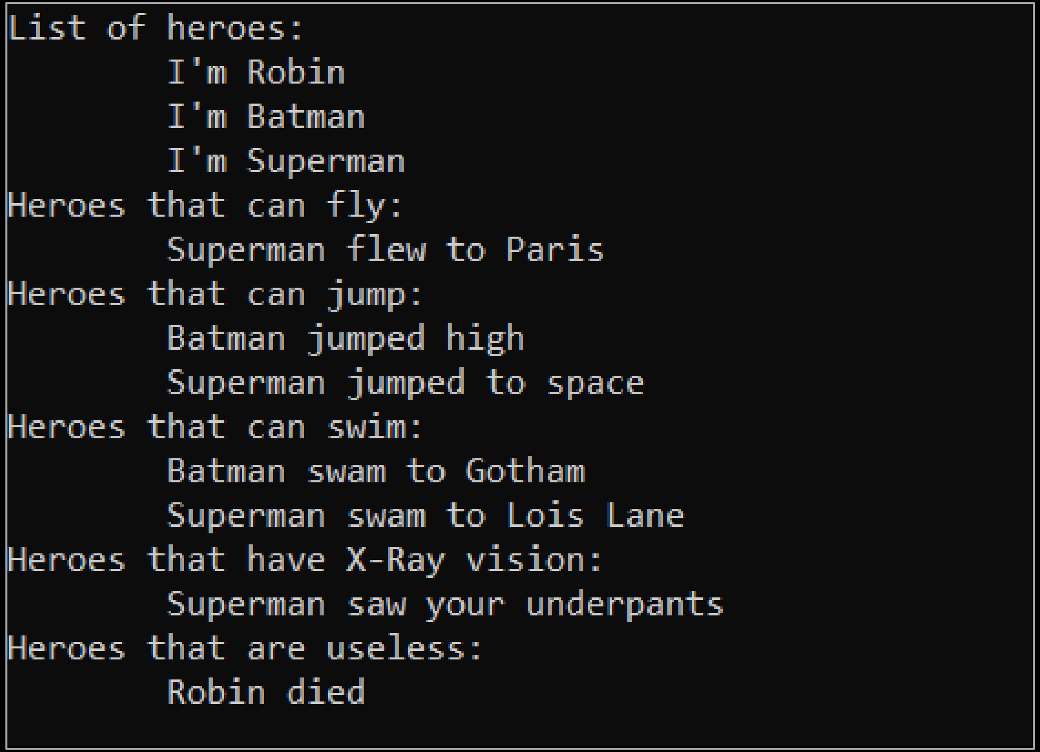
- Can be an instance of any class that implements ICanLog!

- We can safely call this method; it will use the implementation of the specific object!

c# syntax: interfaces

EXERCISE

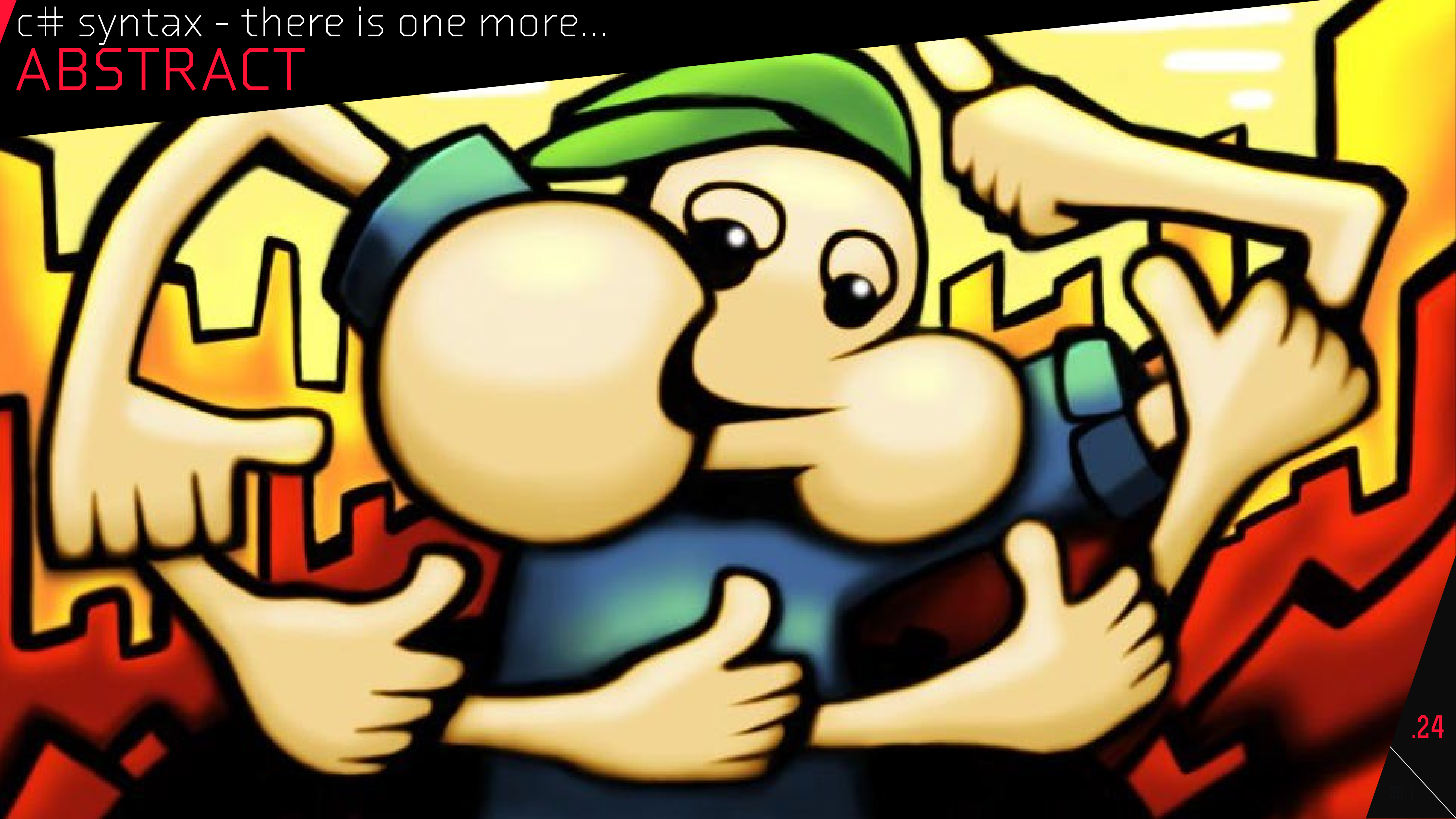
- Given is a class called BaseHero, and Batman/Robin/Superman that inherit this
  - Currently you get an error on the subclasses. Why?
  - Give them a constructor without parameters and pass their literal name to the basehero class.
- Create the following interfaces (the methods write to the console):
  - ICanFly → method Fly();
  - ICanJump → method Jump();
  - ICanSwim → method Swim();
  - IHasXRayVision → method SeeThroughStuff();
  - IUseless → method Die();
- In the subclasses (Batman/Robin/..), implement interfaces of your choice (example screenshot)
- Create a test class
  - Add Batman, Robin and Superman to a list
  - Check their abilities
  - Hint : if (object is Interface)…



```
List of heroes:
        I'm Robin
        I'm Batman
        I'm Superman
Heroes that can fly:
        Superman flew to Paris
Heroes that can jump:
        Batman jumped high
        Superman jumped to space
Heroes that can swim:
        Batman swam to Gotham
        Superman swam to Lois Lane
Heroes that have X-Ray vision:
        Superman saw your underpants
Heroes that are useless:
        Robin died
```

c# syntax - there is one more...

ABSTRACT



OVERVIEW + ABSTRACT KEYWORD

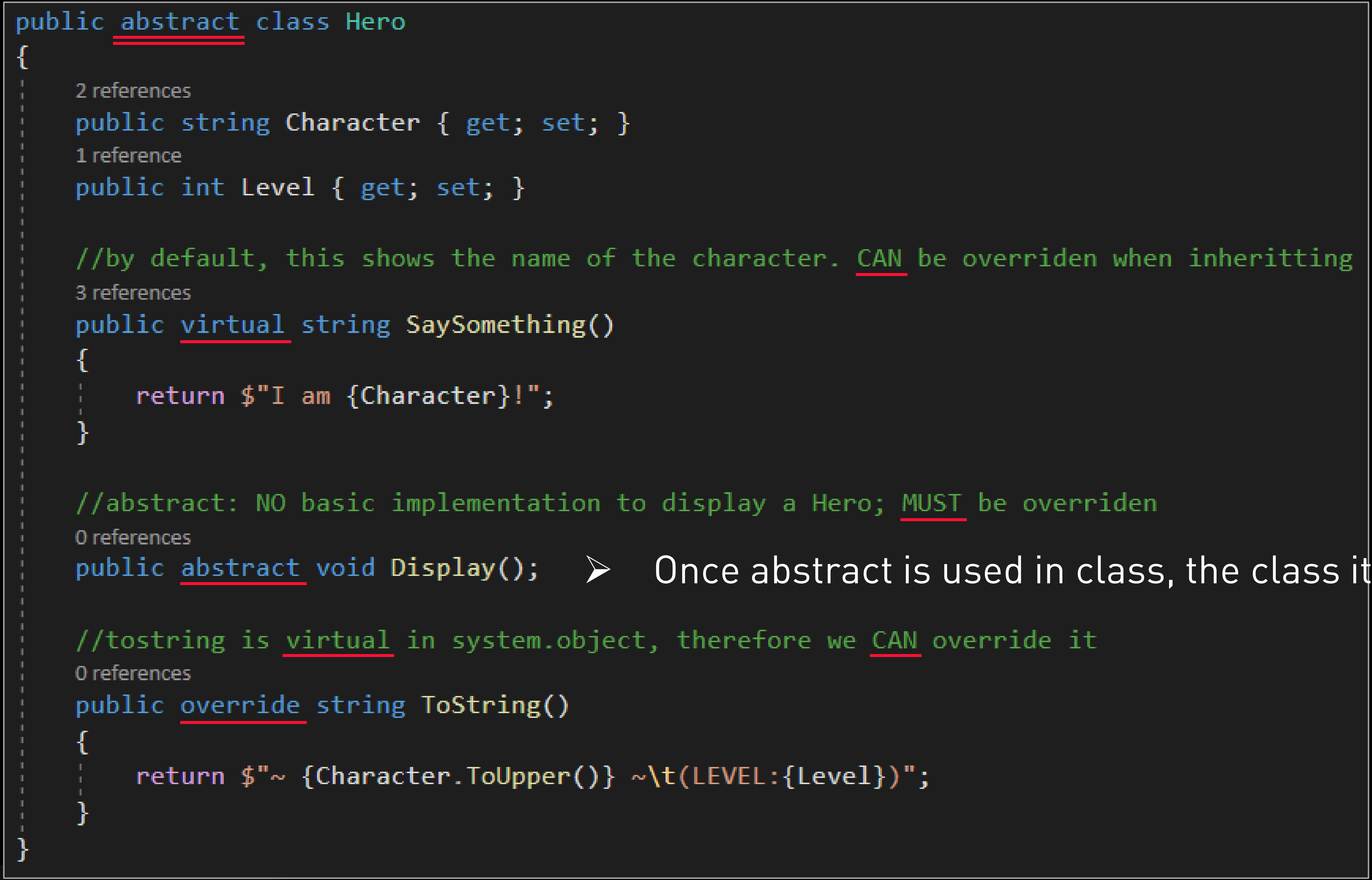
- We have a base class:
  - ✓ we want some default implementation,
  - ✓ but we want to allow inheriting classes to override this behavior,
  - ✓ only if they want to though!
  - ➔ use the **virtual** keyword

- We have a contract:
  - we have **no** default implementations,
  - and we want to force the user to implement everything in the contract
  - ➔ use an **interface** instead of a class

- So what if we want to implement part of a class, but not everything??
  - ➔ abstract

c# syntax: inheritance & interfaces

ABSTRACT & VIRTUAL KEYWORDS

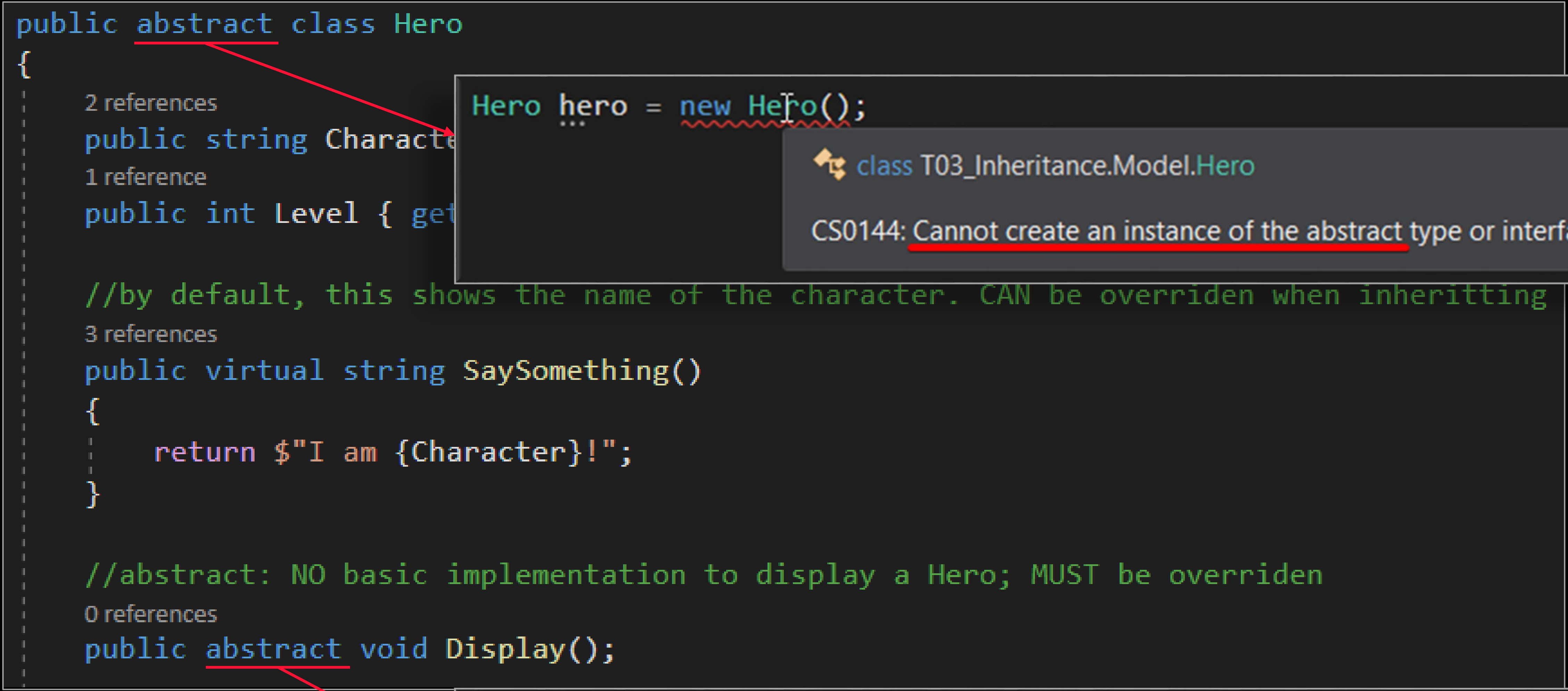


```
public abstract class Hero
{
    2 references
    public string Character { get; set; }
    1 reference
    public int Level { get; set; }

    //by default, this shows the name of the character. CAN be overriden when inheritting
    3 references
    public virtual string SaySomething()
    {
        return $"I am {Character}!";
    }

    //abstract: NO basic implementation to display a Hero; MUST be overriden
    0 references
    public abstract void Display();

    //tostring is virtual in system.object, therefore we CAN override it
    0 references
    public override string ToString()
    {
        return $"~ {Character.ToUpper()} ~\t(LEVEL:{Level})";
    }
}
```

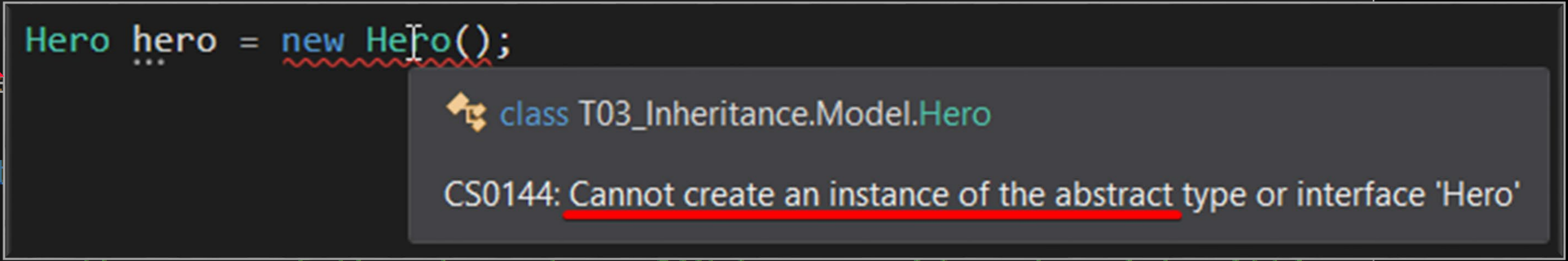
- Once abstract is used in class, the class itself becomes abstract!

c# syntax: inheritance & interfaces

ABSTRACT & VIRTUAL KEYWORDS



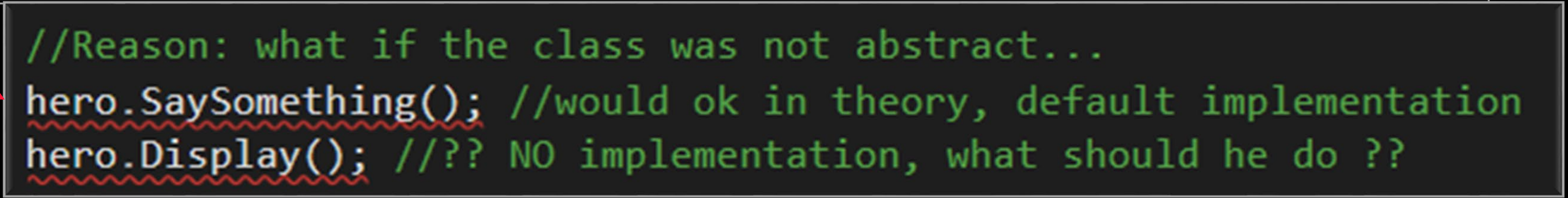
```
public abstract class Hero
{
    2 references
    public string Characte
    1 reference
    public int Level { get

    //by default, this shows the name of the character. CAN be overriden when inheritting
    3 references
    public virtual string SaySomething()
    {
        return $"I am {Character}!";
    }

    //abstract: NO basic implementation to display a Hero; MUST be overriden
    0 references
    public abstract void Display();
```



```
Hero hero = new Hero();
```

class T03_Inheritance.Model.Hero

CS0144: Cannot create an instance of the abstract type or interface 'Hero'



```
//Reason: what if the class was not abstract...
hero.SaySomething(); //would ok in theory, default implementation
hero.Display(); //?? NO implementation, what should he do ??
```