# 10 common C# mistakes and how to avoid them

14 mei 2020 om 11:26 by ParTech Media - 0 Comments

C# is a strongly typed language with many cool features to help us develop software with the greates ease. We can however still make mistakes and many of them are quite common.

Here are ten common C# mistakes that are often made and how to avoid them!

## 1. USAGE OF STRING CONCATENATION

String concatenation functions in a very simple way - every time when you add something to string, then a new address in the memory is automatically allocated. The previous string is copied to the new part with the changed location, it is an ineffective process!

```
List values = new List(){"This ","is ","Sparta ","!"};
string outputValue = string.Empty;

foreach (var value in values)
{
  outputValue += value; // Creates a copy of the string in memory
}
```

Copy

*Source: https://blog.aspnetrya.nl/technology/8-most-common-mistakes-c-developers-make/*

### HOW TO AVOID

The solution is to use StringBuilder object instead of the string concentration which will keep the same position in the memory without any copy task. Due to the strings appending, the process becomes more effective and can smoothly append hundreds of operations.

```
StringBuilder outputValueBuilder = new StringBuilder();

foreach (var value in values)
{
  outputValueBuilder.Append(value);
}
```

Copy

*Source: https://blog.aspnetrya.nl/technology/8-most-common-mistakes-c-developers-make/*

## 2. INCORRECT EVALUATION OF THE DEFAULT VALUE FOR UNINITIALISED VARIABLES

Value types in C# can't be null. Even the uninitialized variables must have some value. This value is called as a default value. When coders check the value of the uninitialized variable, then it results in an unexpected result:

```
class Program {
  static Point point1;
  static Per per1;

  static void Main(string[] args) {
    Console.WriteLine(per1 == null);  // True
    Console.WriteLine(point1 == null); // False (huh?)
  }
}
```

Copy

*Source: https://www.toptal.com/c-sharp/top-10-mistakes-that-c-sharp-programmers-make*

### HOW TO AVOID

Many value types contain an `IsEmpty` property that you have to check and ensure that if it is equal to a default value or not. When you are checking the variable initialized status, then make sure to know what value an uninitialized variable of the type has by default and don't assume it to be null.

```
Console.WriteLine(point1.IsEmpty);
```

## 3. DON'T USE `THROW EX`

When you are planning to plug catch and rethrow an exception, then using throw ex will not preserve the exception call stack for you and that can be a big blunder.

```
catch(SomeException ex)
{
  logger.log(ex);
  throw ex;
}
```

Copy

*Source: https://codeaddiction.net/articles/38/10-common-traps-and-mistakes-in-c*

### HOW TO AVOID

You should use the simple `throw;` syntax to avoid the problem of preserving the exception call stack.

```
catch(SomeException ex)
{
  logger.log(ex);
  throw;
}
```

Copy

*Source: http://codeaddiction.net/articles/38/10-common-traps-and-mistakes-in-c*

## 4. USING `T` CASTING

It is one of the very common mistakes committed by C# developers where they use simple 'T' casting. This casting has a negative impact because casted objects can be easily castable. There's a very small chance that an object might not be castable under some favorable circumstances.

```
var woman = (Woman)person;
```

*Source: https://blog.aspnetrya.nl/technology/8-most-common-mistakes-c-developers-make/*

### HOW TO AVOID

Instead of simple `(T)` casting, `as T` needs to be used as it will protect casted objects from turning into castable under different circumstances.

```
var woman = person as Woman;
```

*Source: https://blog.aspnetrya.nl/technology/8-most-common-mistakes-c-developers-make/*

## 5. NOT KNOWING THE IMPORTANCE OF `USING` FOR OBJECT DISPOSAL

Many C# developers aren't familiar with the concept that `using` keyword isn't only used as a directive to add namespaces, but is highly beneficial for disposing of objects as well.

### HOW TO AVOID

If you are sure that certain object needs to be disposed of off after performing the operations, then you can always use the 'using' statement to ensure that object has been completely disposed of.

```
using(SomeDisposableClass someDisposableObject = new SomeDisposableClass())
{
  someDisposableObject.DoTheJob();
}

// Does the same as:
SomeDisposableClass someDisposableObject = new SomeDisposableClass();
try
{
  someDisposableObject.DoTheJob();
}
finally
{
  someDisposableObject.Dispose();
}
```

Copy

*Source: https://blog.aspnetrya.nl/technology/8-most-common-mistakes-c-developers-make/*

## 6. NOT USING `YIELD RETURN`

When you have to perform enumerating over objects for some another caller, then you should utilize the yield return feature. This alone feature will help you a lot.

### HOW TO AVOID

You should not try to create a return collection. By using `yield return`, you have multiple benefits:

- You won't have to store the whole collection in memory which will be a lot.
- This return will immediately return control to the caller after each iteration.
- You will only process the results that are actually useful because there's no purpose of iterating the whole collection.

## 7. SELECTING THE WRONG TYPE OF COLLECTION

C# offers an array of collection objects such as `HashTable`, `HybridDictionary`, `ListCh`, `NameValueCollection`, `OrderedDictionary`, `Queue`, and many more. Using different objects a programmer can develop really functional projects. But, sometimes too many options can lead to confusion. A large number of options are useless unless the programmer doesn't know how to use the right objects as per the demand of the situation.

### HOW TO AVOID

If you are working with C# then you have to learn how to take leverage from collections. Take time to research and select the optimal collection type for your solution.

## 8. MAKING TOO MANY DATABASE CALLS

Another common error, especially junior programmers commit this mistake a lot when they are using ORMs like Entity Framework or NHibernate. Every database call consumes some time, thus it is vital to decrease the number of datavase calls as much as you can to save time.

### HOW TO AVOID

The number of database calls can be reduced in different ways:

- Combine multiple queries into one.
- Enclose database operations in transactions .
- When you have to deal with complex logics, move it to a stored procedure.

## 9. PARSING AMBIGUOUS DATES**

When you have to parse ambiguous dates in your program, then you have to specify a format provider.

### HOW TO AVOID

You have to use the proper date format of the month, day and year format to parse ambiguous dates. To do that, you can use `DateTime.ParseExact`.

## 10. REPLACING HASH CODE**

The programmer always needs to remember that dictionaries depend upon the keys' return value of `Object.GetHashCode()`. Thus, if you change the hash code of key after it is added to the dictionary, then that would be a mistake.

### HOW TO AVOID

Don't change your hash code after inserting it into a dictionary.

## CONCLUSION

C# is a great programming language with lots of measures in place to avoid us to make mistakes. Even so, there are still traps that you may easily fall into. Experience is the only to avoid mistakes, so keep up the work and never stop learning!

Deel deze publicatie:

## GERELATEERD


Top .NET Libraries You Should Know
vrijdag om 15:00


Overview of ASP .NET Core Authentication
12 maart 2021 om 10:00


Aan de slag met NuGet 5.8
30 november 2020 om 15:30


.NET Conference 2020
30 november 2020 om 15:30


C# 9.0 is nu uit!
30 november 2020 om 15:30

## NIEUWSTE


Kestrel vs IIS Web Servers
vandaag om 10:00


What is Protobuf?
maandag om 10:00


Top .NET Libraries You Should Know
vrijdag om 15:00


Differences between MVC,MVP, and MVVM
een week geleden om 10:00


Introduction to ONNX
94 oktober 2021 om 10:00



**OVER PARTECH**

ParTech is dé technische partner voor je online platform. Wij zijn gespecialiseerd in ontwikkeling, consultancy, support & beheer en trainingen op de Microsoft stack.

Meer over ParTech

**CONTACTGEGEVENS**

Rompertdreef 9
5233 ED 's-Hertogenbosch

+31 (0)85 - 0020 678
info@partech.nl