

# What is Type-safe?

Asked 16 years, 1 month ago    Modified 1 year, 11 months ago

Viewed 157k times



What does "type-safe" mean?

352

language-agnostic

programming-languages



computer-science

terminology

type-safety



Share



Improve this question

Follow

edited Jan 26, 2010 at 15:50



Sinan Ünür

118k ● 15 ● 199 ● 343

asked Nov 4, 2008 at 2:27



Razin

[en.wikipedia.org/wiki/Type\\_safety](https://en.wikipedia.org/wiki/Type_safety) – Basil Bourque Aug 1, 2021 at 23:09

13 Answers

Sorted by:

Highest score (default)



Type safety means that the compiler will validate types while compiling, and throw an error if you try to assign the wrong type to a variable.

304



Some simple examples:



```
// Fails, Trying to put an integer in a string  
String one = 1;  
// Also fails.  
int foo = "bar";
```

This also applies to method arguments, since you are passing explicit types to them:

```
int AddTwoNumbers(int a, int b)  
{  
    return a + b;  
}
```

If I tried to call that using:

```
int Sum = AddTwoNumbers(5, "5");
```

The compiler would throw an error, because I am passing a string ("5"), and it is expecting an integer.

In a loosely typed language, such as javascript, I can do the following:

```
function AddTwoNumbers(a, b)  
{  
    return a + b;  
}
```

if I call it like this:

```
Sum = AddTwoNumbers(5, "5");
```

Javascript automatically converts the 5 to a string, and returns "55". This is due to javascript using the + sign for string concatenation. To make it type-aware, you would need to do something like:

```
function AddTwoNumbers(a, b)
{
    return Number(a) + Number(b);
}
```

Or, possibly:

```
function AddOnlyTwoNumbers(a, b)
{
    if (isNaN(a) || isNaN(b))
        return false;
    return Number(a) + Number(b);
}
```

if I call it like this:

```
Sum = AddTwoNumbers(5, " dogs");
```

Javascript automatically converts the 5 to a string, and appends them, to return "5 dogs".

Not all dynamic languages are as forgiving as javascript (In fact a dynamic language does not implicitly imply a loose typed language (see Python)), some of them will actually give you a runtime error on invalid type casting.

While its convenient, it opens you up to a lot of errors that can be easily missed, and only identified by testing the running program. Personally, I prefer to have my compiler tell me if I made that mistake.

Now, back to C#...

C# supports a language feature called [covariance](#), this basically means that you can substitute a base type for a child type and not cause an error, for example:

```
public class Foo : Bar
{
}
```

Here, I created a new class (Foo) that subclasses Bar. I can now create a method:

```
void DoSomething(Bar myBar)
```

And call it using either a Foo, or a Bar as an argument, both will work without causing an error. This works because C# knows that any child class of Bar will implement the interface of Bar.

However, you cannot do the inverse:

```
void DoSomething(Foo myFoo)
```

In this situation, I cannot pass Bar to this method, because the compiler does not know that Bar implements

Foo's interface. This is because a child class can (and usually will) be much different than the parent class.

Of course, now I've gone way off the deep end and beyond the scope of the original question, but its all good stuff to know :)

Share Improve this answer

edited Jul 2, 2012 at 2:50

Follow



Phil.Wheeler

16.8k ● 10 ● 102 ● 156

answered Nov 4, 2008 at 2:33



FlySwat

175k ● 75 ● 248 ● 314

---

**38** I feel that this answer is wrong: type safety is not necessarily enforced at compile time. I understand that Scheme, for instance, is considered type safe, but is dynamically checked (type safety is enforced at runtime). This is mostly paraphrasing the introduction to Types and Programming Languages, by Benjamin C. Pierce. – [Nicolas Rinaudo](#) Feb 2, 2014 at 18:44

---

**17** What you describe is called polymorphism, not covariance. Covariance is used in generics. – [IS4](#) Apr 16, 2015 at 18:12

---

@NicolasRinaudo note that the gap between dynamic languages and static is being eroded by dynamic compilation and precompilation for "interpreted" languages, and by reflection in "compiled" languages. Reflection allows runtime duck typing, for example, so a compiled language can say "hey, this has a Quack() method, I'll call that and see what happens". Pascal-like languages also often have (optional) runtime overflow checking, leading to those "compiler" errors happening at runtime "cannot fit integer supplied into 8 bit

destination {core dump}". – [Code Abominator](#) May 24, 2017 at 3:35

---

- 4 Your example references to a concept called "strongly typed" which is not the same as type safety. Type safety is when a language can detect type errors on execution or compile time. Python for example is weakly typed and type safe. This answer should be flagged as it's very misleading.  
– [dantebarba](#) Feb 18, 2019 at 16:26
- 

The first paragraph in the wikipedia article you linked, says that it's called subtyping (not covariance, covariance is something else). – [CClairvoyant](#) Dec 6 at 18:53

---



Type-safety should not be confused with static / dynamic typing or strong / weak typing.

**101**



A type-safe language is one where the only operations that one can execute on data are the ones that are condoned by the data's type. That is, if your data is of type `x` and `x` doesn't support operation `y`, then the language will not allow you to to execute `y(x)`.



This definition doesn't set rules on *when* this is checked. It can be at compile time (static typing) or at runtime (dynamic typing), typically through exceptions. It can be a bit of both: some statically typed languages allow you to cast data from one type to another, and the validity of casts must be checked at runtime (imagine that you're trying to cast an `object` to a `Consumer` - the compiler has no way of knowing whether it's acceptable or not).

Type-safety does not necessarily mean strongly typed, either - some languages are notoriously weakly typed, but still arguably type safe. Take Javascript, for example: its type system is as weak as they come, but still strictly defined. It allows automatic casting of data (say, strings to ints), but within well defined rules. There is to my knowledge no case where a Javascript program will behave in an undefined fashion, and if you're clever enough (I'm not), you should be able to predict what will happen when reading Javascript code.

An example of a type-unsafe programming language is C: reading / writing an array value outside of the array's bounds has an undefined behaviour *by specification*. It's impossible to predict what will happen. C is a language that has a type system, but is not type safe.

Share Improve this answer

answered Aug 6, 2014 at 9:57

Follow



Nicolas Rinaudo

6,148 ● 1 ● 30 ● 44

- 
- 1 what are other examples of type-unsafe languages? What do you mean by "writing an array value outside of the array's bounds has an undefined behaviour by specification. It's impossible to predict what will happen". Like Javascript, it will return undefined right? Or really anything can happen. Can you give example of this? – [ARK](#) Feb 15, 2018 at 20:30
-

- 1 @AkshayrajKore sure. Arrays are memory pointers, so by writing out of bounds, you might be overwriting another program's data - which can do nothing, crash the program, cause it to erase your hard drive - it's undefined and depends on who's reading that bit of memory and how it will react to it.  
– [Nicolas Rinaudo](#) Feb 22, 2018 at 20:34 ✎

---

@Nicolas Rinaudo That is not correct. You should read about virtual memory. Each process has its own virtual address space so a process cannot "overwrite another program's data" in such way. – [ilstam](#) Jul 8, 2018 at 19:27

- 
- 1 @NicolasRinaudo The code segment of the program is mapped read-only in the virtual address space. So if you tried to write to it that would cause a segmentation fault and your program would crash. As well if you tried to write to unmapped memory that would cause a page fault and again crash. However, if you are unlucky you might just overwrite data from the process's stack or heap (like other variables or other stuff). In that case you probably wouldn't crash immediately which is even worse because you won't notice the bug until (hopefully) later! – [ilstam](#) Jul 12, 2018 at 23:05 ✎

- 
- 1 Excellent answer. Just to add, Python is another well known example of type-safe language that is dynamically typed.  
– [dantebarba](#) Feb 18, 2019 at 16:31
- 



50

Type safety is not just a compile time constraint, but a **run time** constraint. I feel even after all this time, we can add further clarity to this.



There are 2 main issues related to type safety. Memory\*\* and data type (with its corresponding operations).







## Memory\*\*

A `char` typically requires 1 byte per character, or 8 bits (depends on language, Java and C# store unicode chars which require 16 bits). An `int` requires 4 bytes, or 32 bits (usually).

Visually:

```
char: |-|-|-|-|-|-|-|-|-|
```

```
int : |-|-|-|-|-|-|-|-|-| |-|-|-|-|-|-|-|-|-| |-|-|-|-|-|-|-|-|-|
|-|-|-|-|-|-|-|-|-|
```

A type safe language does not allow an `int` to be inserted into a `char` at *run-time* (this should throw some kind of class cast or out of memory exception). However, in a type unsafe language, you would overwrite existing data in 3 more adjacent bytes of memory.

```
int >> char:
```

```
|-|-|-|-|-|-|-|-|-| |?|?|?|?|?|?|?|?| |?|?|?|?|?|?|?|?|
| |?|?|?|?|?|?|?|?|
```

In the above case, the 3 bytes to the right are overwritten, so any pointers to that memory (say 3 consecutive chars) which expect to get a predictable char value will now have garbage. This causes `undefined` behavior in your program (or worse, possibly in other programs depending on how the OS allocates memory - very unlikely these days).



...'course, banking programs use much larger data types.  
;) LOL!

As others have already pointed out, the next issue is computational operations on types. That has already been sufficiently covered.

## Speed vs Safety

Most programmers today never need to worry about such things unless they are using something like C or C++. Both of these languages allow programmers to easily violate type safety at run time (direct memory referencing) despite the compilers' best efforts to minimize the risk. HOWEVER, this is not all bad.

One reason these languages are so computationally fast is they are not burdened by verifying type compatibility during run time operations like, for example, Java. They assume the developer is a good rational being who won't add a string and an int together and for that, the developer is rewarded with speed/efficiency.

[Share](#) [Improve this answer](#)

[edited Jan 9, 2018 at 21:00](#)

[Follow](#)

answered Jan 9, 2018 at 20:28



[Gr3go](#)

714 ● 5 ● 9

- 
- 1 It is true that ensuring Type Safety puts constraints on Speed. But it is really important that Type Safety is ensured given that C/C++ code is more susceptible to BufferOverflow attacks and other related attacks. Threats of such attacks are reduced by ensuring Type Safety. – [Prateek93a](#) Mar 11, 2021 at 6:15
- 
- 1 Wow.. Very nicely explained. :-) .. Thanks Gr3go – [Mohit Tomar](#) Oct 12, 2022 at 5:02
- 



30



Many answers here conflate type-safety with static-typing and dynamic-typing. A dynamically typed language (like smalltalk) can be type-safe as well.

A short answer: a language is considered type-safe if no operation leads to undefined behavior. Many consider the requirement of explicit type conversions necessary for a language to be *strictly* typed, as automatic conversions can sometimes leads to well defined but unexpected/unintuitive behaviors.

Share Improve this answer

edited Nov 4, 2008 at 6:36

Follow


answered Nov 4, 2008 at 5:26



[ididak](#)

5,858 ● 1 ● 22 ● 21

- 
- 1 Wait, your definition of type-safety does not have a single word "type" :D if no operation leads to undefined behavior . – [VasiliNovikov](#) Sep 2, 2014 at 12:09
-

- 1 Also, I would disagree to such a definition. I think type-safety means exactly 1. the existence of types 2. the knowledge of them to the compiler, and appropriate checks of course.
- [VasiliNovikov](#) Sep 2, 2014 at 12:12 
- 



A programming language that is 'type-safe' means following things:

17



1. You can't read from uninitialized variables
2. You can't index arrays beyond their bounds
3. You can't perform unchecked type casts



Share Improve this answer

edited Jul 26, 2017 at 0:22

Follow



[Frank T](#)

8,986 ● 9 ● 54 ● 67

answered Jul 31, 2015 at 5:20



[Khursheed](#)

373 ● 4 ● 14



An explanation from a liberal arts major, not a comp sci major:

7



When people say that a language or language feature is type safe, they mean that the language will help prevent you from, for example, passing something that isn't an integer to some logic that expects an integer.



For example, in C#, I define a function as:

```
void foo(int arg)
```

The compiler will then stop me from doing this:

```
// call foo  
foo("hello world")
```

In other languages, the compiler would not stop me (or there is no compiler...), so the string would be passed to the logic and then probably something bad will happen.

Type safe languages try to catch more at "compile time".

On the down side, with type safe languages, when you have a string like "123" and you want to operate on it like an int, you have to write more code to convert the string to an int, or when you have an int like 123 and want to use it in a message like, "The answer is 123", you have to write more code to convert/cast it to a string.

Share Improve this answer

edited Nov 6, 2008 at 2:00

Follow

answered Nov 4, 2008 at 2:38



Corey Trager

23.1k ● 20 ● 87 ● 121

- 
- 5 Liberal arts major would say *an* explanation :) You're also conflating static typing and dynamic typing. – [ididak](#) Nov 4, 2008 at 5:01
-



6

To get a better understanding do watch the below video which demonstrates code in type safe language (C#) and NOT type safe language ( javascript).



[http://www.youtube.com/watch?v=Rlw\\_njQhkxw](http://www.youtube.com/watch?v=Rlw_njQhkxw)



Now for the long text.



Type safety means preventing type errors. Type error occurs when data type of one type is assigned to other type UNKNOWINGLY and we get undesirable results.

For instance JavaScript is a NOT a type safe language. In the below code “num” is a numeric variable and “str” is string. Javascript allows me to do “num + str” , now GUESS will it do arithmetic or concatenation .

Now for the below code the results are “55” but the important point is the confusion created what kind of operation it will do.

This is happening because javascript is not a type safe language. Its allowing to set one type of data to the other type without restrictions.

```
<script>
var num = 5; // numeric
var str = "5"; // string
var z = num + str; // arithmetic or concat ????
```

```
alert(z); // displays "55"  
</script>
```

C# is a type safe language. It does not allow one data type to be assigned to other data type. The below code does not allow "+" operator on different data types.

```
int num = 5;  
string str = "5";  
int total = num + str;
```

(local variable) string str

Error:

Cannot implicitly convert type 'string' to 'int'

Share Improve this answer

answered Jan 15, 2014 at 17:22

Follow



[Shivprasad Koirala](#)

28.5k ● 7 ● 87 ● 75



5



### Concept:

To be very simple Type Safe like the meanings, it makes sure that type of the variable should be safe like

1. no wrong data type e.g. can't save or initialized a variable of string type with integer
2. Out of bound indexes are not accessible
3. Allow only the specific memory location

so it is all about the safety of the types of your storage in terms of variables.



Share Improve this answer

answered Nov 28, 2018 at 8:03

Follow



azizsagi

278 ● 3 ● 12



4



Type-safe means that programmatically, the type of data for a variable, return value, or argument must fit within a certain criteria.

In practice, this means that 7 (an integer type) is different from "7" (a quoted character of string type).



PHP, Javascript and other dynamic scripting languages are usually weakly-typed, in that they will convert a (string) "7" to an (integer) 7 if you try to add "7" + 3, although sometimes you have to do this explicitly (and Javascript uses the "+" character for concatenation).

C/C++/Java will not understand that, or will concatenate the result into "73" instead. Type-safety prevents these types of bugs in code by making the type requirement explicit.

Type-safety is very useful. The solution to the above "7" + 3 would be to type cast (int) "7" + 3 (equals 10).

Share Improve this answer

answered Nov 4, 2008 at 2:50

Follow



Jared Farrish

49.2k ● 17 ● 99 ● 105

Try this explanation on...



3



TypeSafe means that variables are statically checked for appropriate assignment at compile time. For example, consider a string or an integer. These two different data types cannot be cross-assigned (ie, you can't assign an integer to a string nor can you assign a string to an integer).

For non-typesafe behavior, consider this:

```
object x = 89;  
int y;
```

if you attempt to do this:

```
y = x;
```

the compiler throws an error that says it can't convert a System.Object to an Integer. You need to do that explicitly. One way would be:

```
y = Convert.ToInt32( x );
```

The assignment above is not typesafe. A typesafe assignment is where the types can directly be assigned to each other.

Non typesafe collections abound in ASP.NET (eg, the application, session, and viewstate collections). The good news about these collections is that (minimizing multiple server state management considerations) you can put pretty much any data type in any of the three collections.

The bad news: because these collections aren't typesafe, you'll need to cast the values appropriately when you fetch them back out.

For example:

```
Session[ "x" ] = 34;
```

works fine. But to assign the integer value back, you'll need to:

```
int i = Convert.ToInt32( Session[ "x" ] );
```

Read about generics for ways that facility helps you easily implement typesafe collections.

C# is a typesafe language but watch for articles about C# 4.0; interesting dynamic possibilities loom (is it a good thing that C# is essentially getting Option Strict: Off... we'll see).

Share Improve this answer

edited Nov 4, 2008 at 2:47

Follow

answered Nov 4, 2008 at 2:41



rp.

17.6k ● 14 ● 65 ● 79

---

Personally, I hate the Convert.To notation, why don't you just use safe cast? Its only less function call on the callstack as well. – [FlySwat](#) Nov 4, 2008 at 2:43

---



2

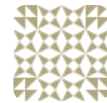


Type-Safe is code that accesses only the memory locations it is authorized to access, and only in well-defined, allowable ways. Type-safe code cannot perform an operation on an object that is invalid for that object. The C# and VB.NET language compilers always produce type-safe code, which is verified to be type-safe during JIT compilation.

Share Improve this answer

answered Oct 27, 2011 at 14:10

Follow



[Jonuz](#)

139 ● 8

---

Do you mean memory safety? – [golopot](#) Nov 20, 2017 at 10:19

---



2



## Type Safety

In modern C++, type safety is very important. Type safety means that you use the types correctly and, therefore, avoid unsafe casts and unions. Every object in C++ is used according to its type and an object needs to be initialized before its use.

**Safe Initialization: {}**

The compiler protects from information loss during type conversion. For example,

`int a{7};` The initialization is OK

`int b{7.5}` Compiler shows ERROR because of information loss.\

## Unsafe Initialization: = or ()

The compiler doesn't protect from information loss during type conversion.

`int a = 7` The initialization is OK

`int a = 7.5` The initialization is OK, but information loss occurs. The actual value of a will become 7.0

`int c(7)` The initialization is OK

`int c(7.5)` The initialization is OK, but information loss occurs. The actual value of a will become 7.0

Share Improve this answer

answered Dec 24, 2022 at 10:53

Follow



**Forhad Hossain**

428 ● 5 ● 18



1



Type-safe means that the set of values that may be assigned to a program variable must fit well-defined and testable criteria. Type-safe variables lead to more robust programs because the algorithms that manipulate the variables can trust that the variable will only take one of a well-defined set of values. Keeping this trust ensures the integrity and quality of the data and the program.

For many variables, the set of values that may be assigned to a variable is defined at the time the program is written. For example, a variable called "colour" may be allowed to take on the values "red", "green", or "blue" and never any other values. For other variables those criteria may change at run-time. For example, a variable called "colour" may only be allowed to take on values in the "name" column of a "Colours" table in a relational database, where "red", "green", and "blue", are three values for "name" in the "Colours" table, but some other part of the computer program may be able to add to that list while the program is running, and the variable can take on the new values after they are added to the Colours table.

Many type-safe languages give the illusion of "type-safety" by insisting on strictly defining types for variables and only allowing a variable to be assigned values of the same "type". There are a couple of problems with this approach. For example, a program may have a variable "yearOfBirth" which is the year a person was born, and it is tempting to type-cast it as a short integer. However, it is not a short integer. This year, it is a number that is less than 2009 and greater than -10000. However, this set grows by 1 every year as the program runs. Making this a "short int" is not adequate. What is needed to make this variable type-safe is a run-time validation function that ensures that the number is always greater than -10000 and less than the next calendar year. There is no compiler that can enforce such criteria because these

criteria are always unique characteristics of the problem domain.

Languages that use dynamic typing (or duck-typing, or manifest typing) such as Perl, Python, Ruby, SQLite, and Lua don't have the notion of typed variables. This forces the programmer to write a run-time validation routine for every variable to ensure that it is correct, or endure the consequences of unexplained run-time exceptions. In my experience, programmers in statically typed languages such as C, C++, Java, and C# are often lulled into thinking that statically defined types is all they need to do to get the benefits of type-safety. This is simply not true for many useful computer programs, and it is hard to predict if it is true for any particular computer program.

The long & the short.... Do you want type-safety? If so, then write run-time functions to ensure that when a variable is assigned a value, it conforms to well-defined criteria. The down-side is that it makes domain analysis really difficult for most computer programs because you have to explicitly define the criteria for each program variable.

[Share](#) [Improve this answer](#)

answered Nov 4, 2008 at 3:25

[Follow](#)



[Jay Godse](#)

15.5k ● 16 ● 92 ● 133

- 
- 3 Python variables are typed (*strongly* typed, in fact). Try doing this, for example: "str" + 1. You'll get an error. However, the types are checked at runtime, rather than compile time.
- [mipadi](#) Nov 4, 2008 at 6:19
-