Casting and Type Conversion

.NET

C# is statically typed at compile time. After a variable is declared, it cannot be declared again or assigned a value of another type unless that type is implicitly convertible to the variable's type. Converting one type to another implicitely is called ***type conversion****.*

HTTPS://DOCS.MICROSOFT.COM/EN-US/DOTNET/CSHARP/PROGRAMMING-GUIDE/TYPES/CASTING-AND-TYPE-CONVERSIONS

Type Casting vs Type Conversion

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/casting-and-type-conversions

```
public class Person
{
    public Person(string fname, string lname)
    {
        this.Fname = fname;
        this.Lname = lname;
    }

    public string Fname { get; set; }
    public string Lname { get; set; }

    /// <summary>
    /// this method returns a string that identifies the person
    /// </summary>
    /// <returns></returns>
    public virtual string SayMyName()
    {
        return $"The person is {this.Fname} {this.Lname}.";
    }
```

```
public class MoreSpecificPerson : Person
{
    public int Age { get; set; }

    public MoreSpecificPerson(string fname, string lname, int age) : base(fname, lname)
    {
        this.Age = age;
    }


    public override string SayMyName()
    {
        return $"My name is {this.Fname} {this.Lname} and my age is {this.Age}.";
    }
```

Type Casting vs Type Conversion

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/casting-and-type-conversions

There are 2 types of datatype conversions in C#	
Type Conversion (implicit):	Type Casting (explicit):
• This is done by the compiler. • No special syntax is required. • Type safe. • No data loss.	• Requires use of the ***cast operator***, ***()***. • A ***cast*** is required when data might be lost in the conversion, or • When failure could occur.

```
//IMPLICIT Conversion
int myInt2 = 12;
double myDouble2 = myInt2;
Console.WriteLine(myDouble2);
```

```
// EXPLICIT CASTING
double myDouble = 11.11;
Console.WriteLine(myDouble);
int myInt = (int)myDouble;
Console.WriteLine(myInt);
```

```
// UPCASTING
Person p2 = msp;
Console.WriteLine($"P2 i {p2.Fname} {p2.Lname} ...");
//DOWNCASTING
MoreSpecificPerson msp3 = (MoreSpecificPerson)p2;
msp3.Age = 46;
Console.WriteLine($"P2 i {msp3.Fname} {msp3.Lname} who is {msp3.Age} years old");
```

Casting and Type Conversion

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/casting-and-type-conversions

---

There are 2 types of conversions in C#:

- Implicit conversions: No special syntax. Type safe. No data loss.
- Explicit conversions (casts): Explicit conversions require the cast operator, (). A ***cast*** is required when data might be lost in the conversion, or when failure could occur.

Implicit Conversion

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/casting-and-type-conversions#implicit-conversions

Implicit conversion is possible in:

- *numeric* types when the value to be stored can fit into the variable memory without being truncated.
- *integral* types when the range of the source *type* is at least as big as the target *type*.

```
// Implicit conversion. A long can
// hold any value an int can hold, and more!
int num = 2147483647;
long bigNum = num;
```

Implicit conversion is always possible in *reference* types.

- When a class is converted to any one of its direct or indirect *base* classes or *interfaces*.
- No special syntax is necessary.
- Derived classes always contain all the members of the base class.

```
Derived d = new Derived();
Base b = d; // Always OK.
```

Explicit Conversion

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/casting-and-type-conversions#explicit-conversions

---

If there is a risk of losing information, you must perform a ***cast***.

Specify the target ***type*** in ***( )*** in front of the value or variable to be converted.

*This doesn't prevent the loss of data.

An explicit ***cast*** is required if you need to convert from a ***base*** type to a ***derived*** type.

```
class Test
{
    static void Main()
    {
        double x = 1234.7;
        int a;
        // Cast double to int.
        a = (int)x;
        System.Console.WriteLine(a);
    }
}
// Output: 1234
```

```
// Create a new derived type.
Giraffe g = new Giraffe();

// Implicit conversion to base type is safe.
Animal a = g;

// Explicit conversion is required to cast back
// to derived type. Note: This will compile but will
// throw an exception at run time if the right-side
// object is not in fact a Giraffe.
Giraffe g2 = (Giraffe) a;
```

Type conversion run time exceptions

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/casting-and-type-conversions#type-conversion-exceptions-at-run-time

In some ***reference*** type conversions, It is possible for a ***cast*** operation that compiles correctly to fail at run time.

A ***type cast*** that fails at run time will cause an **InvalidCastException** to be thrown.

```
using System;

class Animal
{
    public void Eat() { Console.WriteLine("Eating."); }
    public override string ToString()
    {
        return "I am an animal.";
    }
}
class Reptile : Animal { }
class Mammal : Animal { }

class UnSafeCast
{
    static void Main()
    {
        Test(new Mammal());

        // Keep the console window open in debug mode.
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }

    static void Test(Animal a)
    {
        // Cause InvalidCastException at run time
        // because Mammal is not convertible to Reptile.
        Reptile r = (Reptile)a;
    }
}
```