CPP Module 06

There is no such thing as a static class in C++. The closest approximation is a class that only contains static data members and static methods.

Static data members in a class are shared by all the class objects as there is only one copy of them in the memory, regardless of the number of objects of the class. Static methods in a class can only access static data members, other static methods or any methods outside the class.

String to Int in C++ – How to Convert a String to an Integer Example

When you're coding in C++, there will often be times when you'll want to convert one data type to a different one. In this article you'll learn how to convert a string to an integer in C++ by seeing two of the most popular ways

https://www.freecodecamp.org/news/string-to-int-in-c-how-to-convert-a-string-to-an-integer-example/

The `stringstream` class is mostly used in earlier versions of C++. It works by performing inputs and outputs on strings.

To use it, you first have to include the `sstream` library at the top of your program by adding the line `#include <sstream>`.

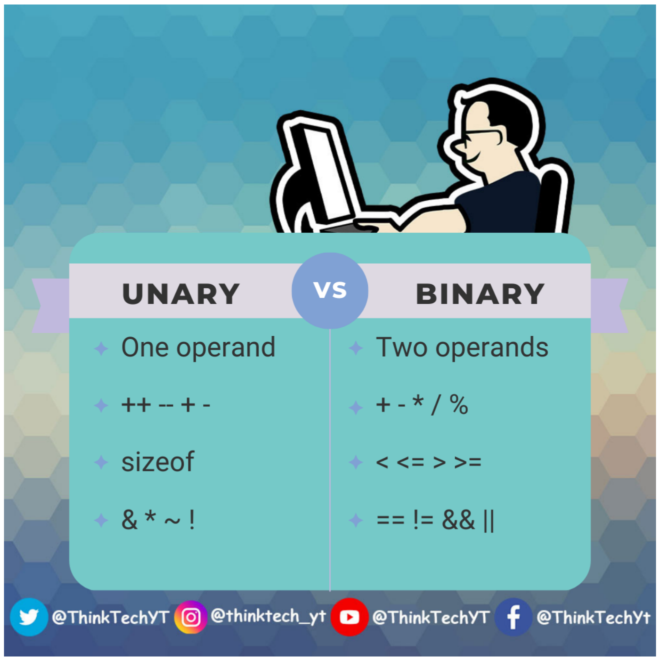
You then add the `stringstream` and create an `stringstream` object, which will hold the value of the string you want to convert to an int and will be used during the process of converting it to an int.

You use the `<<` operator to *extract* the string from the string variable.

Lastly, you use the `>>` operator to *input* the newly converted int value to the int variable.

```
#include <iostream>
#include <string>
#include <sstream> // this will allow you to use stringstream in your program

using namespace std;

int main() {
    //create a stringstream object, to input/output strings
   stringstream ss;

   // a variable named str, that is of string data type
   string str = "7";

   // a variable named num, that is of int data type
   int num;


   //extract the string from the str variable (input the string in the stream)
   ss << str;

   // place the converted value to the int variable
   ss >> num;

   //print to the consloe
   cout << num << endl; // prints the intiger value 7
}
```

⚠️ if(ss >> num) ile çevrim gerçekleşiyor mu diye control et!



UNARY	BINARY
One operand	Two operands
++ -- + -	+ - * / %
sizeof	< <= > >=
& * ~ !	== != && ||

VS
@ThinkTechYT
@thinktech_yt
@ThinkTechYT
@ThinkTechYt

static_cast in C++ | Type Casting operators - GeeksforGeeks

A Computer Science portal for geeks. It contains well written, well thought and well explained computer science and programming articles, quizzes and practice/competitive programming/company

https://www.geeksforgeeks.org/static_cast-in-c-type-casting-operators/

A Cast operator is an **unary operator** which forces one data type to be converted into another data type.C++ supports four types of casting:

1. Static Cast
2. Dynamic Cast
3. Const Cast
4. Reinterpret Cast

Static Cast:
This is the simplest type of cast that can be used. It is a **compile-time cast**. It does things like implicit conversions between types (such as int to float, or pointer to void*), and it can also call explicit conversion functions (or implicit ones).

```
#include <iostream>
using namespace std;
int main()
{
    float f = 3.5;
    int a = f; // this is how you do in C
    int b = static_cast<int>(f);
    cout << b;
}
```

C++ String to float/double and vice-versa

In this tutorial, we will learn how to convert string to floating-point numbers and vice versa with the help of examples.

https://www.programiz.com/cpp-programming/string-float-conversion

---

reinterpret_cast

- It is used to convert a pointer of some data type into a pointer of another data type, even if the data types before and after conversion are different.

- It does not check if the pointer type and data pointed by the pointer are the same or not.

reinterpret_cast in C++ | Type Casting operators - GeeksforGeeks

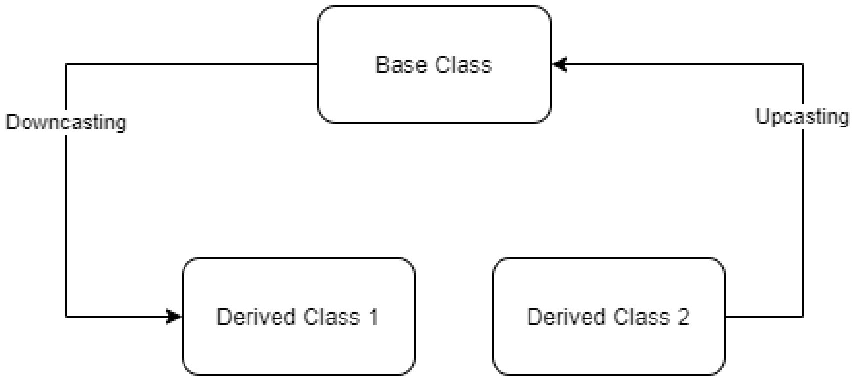
A Computer Science portal for geeks. It contains well written, well thought and well explained computer science and programming articles, quizzes and practice/competitive programming/company interview Questions.

https://www.geeksforgeeks.org/reinterpret_cast-in-c-type-casting-operators/?ref=rp

---

Dynamic _Cast in C++

Dynamic Cast: A cast is an operator that converts data from one type to another type. In C++, dynamic casting is mainly used for safe downcasting at run time. To work on **dynamic_cast** there must be one virtual function in the base class.
A **dynamic_cast** works only with polymorphic base class because it uses this information to decide safe downcasting.



cast doğruysa geçerli bir pointer döndürür. değilse null döndürür.

⚠️ If you want to write really fast code you probably want to avoid that!

RTTI → araştır derinlemesine

```
#include <iostream>

class Animal {
public:
    virtual ~Animal() {}
};

class Dog : public Animal {
public:
    void bark() { std::cout << "Woof!" << std::endl; }
};

int main() {
    Dog dog;

    // Cast the dog object to an Animal reference using dynamic_cast
    Animal& animal = dynamic_cast<Animal&>(dog);

    // Check if the cast was successful
    if (&animal == &dog) {
        std::cout << "Dynamic cast succeeded!" << std::endl;

        // Call a Dog-specific method using the Animal reference
        Dog& dogRef = dynamic_cast<Dog&>(animal);
        dogRef.bark();
    } else {
        std::cout << "Dynamic cast failed!" << std::endl;
    }

    return 0;
}
```

In this example, we have two classes `Animal` and `Dog`, where `Dog` is derived from `Animal`. We create a `Dog` object and then cast it to an `Animal` reference using `dynamic_cast`. We check if the cast was successful and then cast the `Animal` reference back to a `Dog` reference using `dynamic_cast`. Finally, we call a `Dog`-specific method (`bark`) using the `Dog` reference.

Note that `dynamic_cast` can throw a `std::bad_cast` exception if the cast fails, so it's a good idea to use it within a `try`-`catch` block to handle any exceptions that may be thrown.

```
#include <iostream>

class Base {
public:
    virtual ~Base() {}
};

class A : public Base {
public:
    void foo() { std::cout << "A::foo()" << std::endl; }
};

class B : public Base {
```

```
public:
    void bar() { std::cout << "B::bar()" << std::endl; }
};

class C : public Base {
public:
    void baz() { std::cout << "C::baz()" << std::endl; }
};

int main() {
    // Create an A object and cast it to a Base reference
    A a;
    Base& baseRef = a;

    // Attempt to cast the Base reference to an A reference using dynamic_cast
    try {
        A& aRef = dynamic_cast<A&>(baseRef);
        aRef.foo();
    } catch (std::bad_cast& ex) {
        std::cerr << "Failed to cast to A reference: " << ex.what() << std::endl;
    }

    // Attempt to cast the Base reference to a B reference using dynamic_cast
    try {
        B& bRef = dynamic_cast<B&>(baseRef);
        bRef.bar();
    } catch (std::bad_cast& ex) {
        std::cerr << "Failed to cast to B reference: " << ex.what() << std::endl;
    }

    // Attempt to cast the Base reference to a C reference using dynamic_cast
    try {
        C& cRef = dynamic_cast<C&>(baseRef);
        cRef.baz();
    } catch (std::bad_cast& ex) {
        std::cerr << "Failed to cast to C reference: " << ex.what() << std::endl;
    }

    return 0;
}
```

In this example, we have a `Base` class with a virtual destructor and three derived classes `A`, `B`, and `C`. We create an `A` object and cast it to a `Base` reference. We then attempt to cast the `Base` reference to `A`, `B`, and `C` references using `dynamic_cast` and call a method on each derived object if the cast is successful. If the cast fails, we catch the `std::bad_cast` exception and print an error message.

Note that if the `Base` object was not originally created as an instance of the derived class (i.e., it was constructed as a `Base` object), then casting it to a derived reference using `dynamic_cast` will result in undefined behaviour.