- Exercise

In this lesson, we'll solve an exercise related to thread-safe initialization.

WE'LL COVER THE FOLLOWING

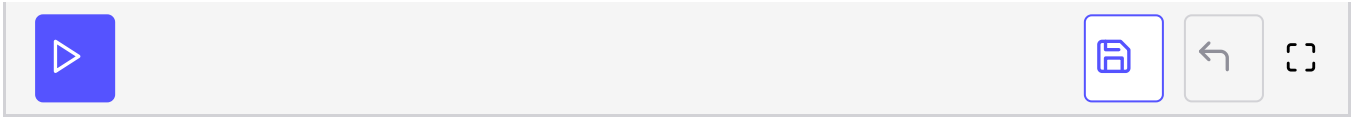
Problem statement

The classical implementation of the singleton pattern in the given code is not thread-safe.

- Use the function `std::call_once` in combination with the `std::once_flag` to make `MySingleton` class thread-safe.

```
#include <iostream>

class MySingleton{

  private:
    static MySingleton* instance;
    MySingleton()= default;
    ~MySingleton()= default;

  public:
    MySingleton(const MySingleton&)= delete;
    MySingleton& operator=(const MySingleton&)= delete;

    // use call_once method to make sure thread executes once only

    static MySingleton* getInstance(){
      if ( !instance ){
        instance= new MySingleton();
      }
      return instance;
    }
};

MySingleton* MySingleton::instance= nullptr;


int main(){
```

```
  std::cout << std::endl;


  std::cout << "MySingleton::getInstance(): "<< MySingleton::getInstance() << std::endl;
  std::cout << "MySingleton::getInstance(): "<< MySingleton::getInstance() << std::endl;

  std::cout << std::endl;

}
```



In the next lesson, we'll look at the solution to this exercise.