- Solution

Let's have a look at the solution review of the last exercise in this lesson.

WE'LL COVER THE FOLLOWING

- Solution Review
  - Explanation

Solution Review

```
//templateClassTemplateMethods2.cpp

#include <algorithm>
#include <iostream>
#include <vector>

template <typename T, int N>
class Array{

public:
  Array()= default;

  template <typename T2, int M> friend class Array;

  template <typename T2>
  Array<T, N>& operator=(const Array<T2, N>& arr){
          static_assert(std::is_convertible<T2, T>::value, "Cannot convert source type to des
    elem.clear();
          elem.insert(elem.begin(), arr.elem.begin(), arr.elem.end());
          return *this;
  }

  int getSize() const;

private:

  std::vector<T> elem;
};

template <typename T, int N>
int Array<T, N>::getSize() const {
  return N;
}

int main(){
  Array<double, 10> doubleArray{};
  Array<int, 10> intArray{};


  doubleArray = intArray;

}
```

Explanation

In the code above, we have created an `Array` class in which we have defined a method `=` which copies the entries of `integer` array to `double` array. The `getSize` function returns the size of the array passed in the argument. Template classes with different template arguments are considered of different types, which is why `Array<int, 10>` and `Array<double, 10>` aren't able to access each other. And that's why the `Array` template class has to be declared a `friend` of itself.

To access the `private elem` of `arr` in line 19, `class` `Array` is a friend of the class `Array` in line 13.

In the next lesson, we'll study dependent names.