- Solution

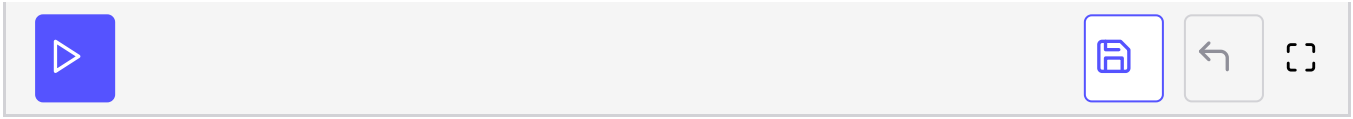
Let's review the solution of the exercise in this lesson.

WE'LL COVER THE FOLLOWING

Solution Review

```
// templateClassTemplateMethods3.cpp

#include <initializer_list>
#include <iostream>
#include <list>
#include <vector>

template <typename T, template <typename, typename> class Cont = std::vector>
class Matrix{
public:
  explicit Matrix(std::initializer_list<T> inList): data(inList){
    for (auto d: data) std::cout << d << " ";
  }
  int getSize() const{
    return data.size();
  }

private:
  Cont<T, std::allocator<T>> data;

};

int main(){

  std::cout << std::endl;

  Matrix<int> myIntVec{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  std::cout << std::endl;
  std::cout << "myIntVec.getSize(): " << myIntVec.getSize() << std::endl;

  std::cout << std::endl;

  Matrix<double> myDoubleVec{1.1, 2.2, 3.3, 4.4, 5.5};
  std::cout << std::endl;
  std::cout << "myDoubleVec.getSize(): "  << myDoubleVec.getSize() << std::endl;
```

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


  Matrix<std::string,std::list> myStringList{"one", "two", "three", "four"};
  std::cout << std::endl;
  std::cout << "myStringList.getSize(): " << myStringList.getSize() << std::endl;

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



Explanation

The class template `Matrix` uses, by default, an `std::vector` for holding its elements (line 8). Thanks to the default container, a `Matrix` can be instantiated (lines 27 and 33) without explicitly specifying a container.

Let's move on to specialization in the next lesson.