```
{
    static std::size_t s_count;     // track number of objects constructed
    T*                 d_data_p;    // pointer to dynamically allocated memory
    std::size_t        d_length;    // current number of elements in the vector
    std::size_t        d_capacity;  // number of elements currently allocated

public:
    // ...

    std::size_t length() const { return d_length; }
        // Return the number of elements.

    // ...
};

// ...          Any partial or full specialization definitions            ...
// ...          of the class template Vector go here.                     ...

template <typename T>
void swap(Vector<T> &lhs, Vector<T> &rhs) { return std::swap(lhs, rhs); }
    // free function that operates on objects of type my::Vector via ADL

// ...            Any [full] specialization definitions                   ...
// ...            of free function swap would go here.                    ...

template <typename T>
const std::size_t vectorSize = sizeof(Vector<T>);  // C++14 variable template
    // This nonmodifiable static variable holds the size of a my::Vector<T>.

// ...           Any [full] specialization definitions                    ...
// ...           of variable vectorSize would go here.                    ...

template <typename T>
std::size_t Vector<T>::s_count = 0;
    // definition of static counter in general template

// ... We might opt to add explicit-instantiation declarations here.
// ...

}  // close namespace my

#endif  // close internal include guard
```

In the `my_vector` **component** in the code snippet above, we have defined the following, in the `my` namespace.

1. A **`class`** template, `Vector`, parameterized on element type