vector Variables

All **library collection classes specify the kind of values they contain** by including the **base type** name in angle brackets following the class name. For example:

```
vector<int>     //represents a vector that contains integers
vector<Card>    // a vector of playing cards (a user-defined type)
vector<string>  // a vector containing string objects.
```

Classes with a base-type specification are called **parameterized** classes, and they are implemented, in C++, using a technique called **class templates**. This means that the classes, **`vector<int>`**, **`vector<Card>`**, and **`vector<string>`** are **independent classes** which each share a common general structure.

To use the standard collections, **include the appropriate header** (**`<vector>`**). The **`vector`**, like the **`string`** class, is in the **`std namespace`**.

Creating a **`vector`** variable is similar to creating an **`int`** variable:

```
int n;  // create an integer
vector<int> iVec;     // an empty vector that stores integers
vector<double> dVec;  // stores doubles
vector<string> sVec;  // stores strings
```

The variable, **`iVec`** **is** a **`vector`** of integers. There is **no separate instantiation step** as in Java, where you might expect to write something like this:

```
vector<int> ivec = new vector<int>(); // Illegal
vector<int&< v1; // Illegal
const vector<int>& = ...;  // OK
```

You **cannot** create a **`vector`** of references, but a **`vector`** may, itself, be a reference (usually when used as a parameter).