13.5 The new constructor converts an array \( a \) whose elements have type \( T \):

```cpp
template<class T>
class Vector
{
public:
    Vector(T* a) : size(sizeof(a)), data(new T[size])
    { for (int i = 0; i < size; i++) data[i] = a[i]; } // other members
};
```

Here is a test driver for the new constructor:

```cpp
int main()
{ int a[] = { 22, 44, 66, 88 };  // 4
    Vector<int> v(a);
    cout << v.size() << endl;
    for (int i = 0; i < 4; i++)
        cout << v[i] << " ";
}
```

The advantage of this constructor is that we can initialize a vector now without having to assign each component separately.

13.6 The derived template has three member functions: two constructors and a new subscript operator:

```cpp
template <class T, class E>
class Array : public Vector<T>
{
public:
    Array(E last) : Vector<T>(unsigned(last) + 1) { }
    Array(const Array<T,E>& a) : Vector<T>(a) { }
    T& operator[](E index) const
    { return Vector<T>::operator[](unsigned(index)); }
};
```

The first constructor calls the default constructor defined in the parent class `Vector<T>`, passing to it the number of `E` values that are to be used for the index. The new copy constructor and subscript operator also invoke their equivalent in the parent class.

Here is a test driver for the `Array<T,E>` template:

```cpp
class enum Days { SUN, MON, TUE, WED, THU, FRI, SAT };

int main()
{ Array<int,Days> customers(SAT);
    customers[MON] = 27; customers[TUE] = 23;
    customers[WED] = 20; customers[THU] = 23;
    customers[FRI] = 36; customers[SAT] = customers[SUN] = 0;
    for (Days day = SUN; day <= SAT; day++)
        cout << customers[day] << " ";
}
```

The enumeration type `Days` defines seven values for the type. Then the object `customers` is declared to be an array of `ints` indexed by these seven values. The rest of the program applies the subscript operator to initialize and then print the array.