Internal working of Vector :

Vector works the same as List. The only different it methods of Vector are synchronised while for List, Synchronization is a responsibility of programmer.
We can use vector when we want thread safety, but nowadays there are much more efficient collections in concurrent package of java 1.5.

Default capacity of Vector is also 10. If you try to add another element after 10, the capacity would increase by 2 times, so it becomes 20, whereas in ArrayList it increases by 50%.
You can also pass the amount by which you want to increase the capacity of the vector.

Internally, Vector just stored objects as type Object. Because all objects inherit from Object, it can store all objects.

Primitive values, such as int and double, however do not inherit from Object, because they are not objects. To cope with them, Java will convert them from primitive values to Objects on the fly, as needed. So the int 5 will be converted to an Integer object holding 5, and the double 4.3 will be converted to a Double object holding 4.3. This is known as auto-boxing.

Java will also do the reverse, convert from Integer to int, for you automatically. This is known as 'un-boxing'.

So that allows it to store any object, but when you code with Vector, you can be more specific. For example Vector or Vector. This restriction on what you can store inside the Vector, is achieved using 'generics', which allows you to pass in a type as a parameter to the class. In those examples that parameter is String and Integer.

Generics however is only enforced at compile time; at runtime, Vector will just continue to use Object internally.

What this means is that when your application is running, there is no difference between Vector and Vector. They are exactly the same, and in fact there is only Vector (no Vector or Vector really exists). The difference between them is entirely at compile time, before the application runs, where with Vector, it will only compile if you only store String objects within it.