Vector in Java Collection Framework

🔵 What is a Vector?

- A **Vector** is a **growable array** — it can increase or decrease its size automatically.
- It is **synchronized**, which means it is **thread-safe** — multiple threads can access it safely.

Synchronized means

⍰ **Synchronized** means **only one thread can access a method of the Vector at a time**.

⍰ **Thread-safe** means **no data will be lost or corrupted even if multiple threads are trying to use it at the same time**.

1.Real-Life Analogy:

Think of a synchronized system like a traffic signal:

- **When one car has a green light, others must stop and wait.**
- **That way, no crash happens.**

A non-synchronized system is like no traffic signal at a busy intersection — everyone goes at the same time → accident.

2.Analogy: A School Backpack

- **Think of a vector like a backpack that can grow in size.**
- **At first, it may only hold a few books.**
- **As you add more books, the backpack magically expands to fit them.**
- **You don't need to buy a new backpack every time you add a book.**
- **If you remove books, it could shrink again.**

📌 **Just like a vector: it automatically resizes as you add or remove items.**

🟢 Key Features of Vector

1. **Dynamic Size** – Automatically grows when needed.

2. **Ordered** – It maintains the order in which elements are added.
3. **Allows Duplicates** – You can store repeated values.
4. **Thread-Safe** – All methods are synchronized by default.
5. ● **Vector Syntax**

```
Vector<Type> vectorName = new Vector<>();
```

● Common Methods in Vector

Method	Use
add()	Add elements
get(index)	Get element at index
remove()	Remove element
size()	Get number of elements
clear()	Remove all elements

✅ Key Differences:

Feature / Method	ArrayList	Vector	Notes
addFirst()	❌	❌	Not available in either
addLast()	❌	❌	Not available in either

🔷 Vector vs ArrayList

Vector	ArrayList
Thread-safe	Not thread-safe
Slower	Faster

⚪ When to Use Vector

- When you need **thread safety**.
- When working with **legacy code**.
- we mostly use **ArrayList** with manual synchronization if needed.

🟤 Conclusion

To conclude:

- Vector is a simple, powerful tool in Java for storing objects.
- It gives you automatic resizing and thread safety.
- However, in modern coding, **ArrayList** is more commonly used.

```
package Week4;
    import java.util.Collections;
    import java.util.Iterator;
    import java.util.Vector;

    public class VectorExample {

        public static void main(String[] args) {
            // Vector of Integers
            Vector<Integer> a = new Vector<>();
            a.add(100);
            a.add(200);
            a.add(300);
            a.add(0, 400);
            // insert at specific position
            System.out.println("Vector a: " + a);

            Vector<Object> mix = new Vector<>();
```

```
mix.add(100);
mix.add("String");
mix.add(0.9);
mix.add(100000);
System.out.println("Mixed Vector: " + mix);

// Type casting
int num = (int) mix.get(0);
System.out.println("First element (int): " + num);
String str1 = (String) mix.get(1);
System.out.println("Second element (String): " + str1);

// Accessing elements using for loop
for (int i = 0; i < a.size(); i++) {
   System.out.println("a[" + i + "] = " + a.get(i));
}

// Using iterator
Iterator<Integer> it = a.iterator();
while (it.hasNext()) {
   Integer s = it.next();
   System.out.println("s = " + s);
}

// Copying and modifying vectors
Vector<Integer> lista = new Vector<>();
lista.add(600);
lista.add(700);
lista.add(800);

System.out.println("lista: " + lista);

Vector<Integer> listb = new Vector<>(lista); // Copy constructor
Collections.copy(listb, lista); // Copy contents
System.out.println("listb (after copy): " + listb);

// Shuffle
System.out.println("Before shuffle: " + lista);
Collections.shuffle(lista);
System.out.println("After shuffle: " + lista);

// Reverse
Collections.reverse(lista);
System.out.println("After reverse: " + lista);

// Swap
Collections.swap(lista, 0, 2);
System.out.println("After swap: " + lista);

// Sort
Collections.sort(lista);
System.out.println("After sort (ascending): " + lista);

// Vector of Strings
Vector<String> listString = new Vector<>();
listString.add("Red");
```

```
        listString.add("Blue");
        listString.add("Yellow");
        listString.add("Black");
        listString.add("White");
        listString.add("Green");

        Collections.sort(listString);
        System.out.println("Sorted list (A-Z): " + listString);

        Collections.sort(listString, Collections.reverseOrder());
        System.out.println("Sorted list (Z-A): " + listString);

        // Merge two Vectors
        Vector<String> listString2 = new Vector<>();
        listString2.add("Red");
        listString2.add("Blue");
        listString2.add("Yellow");
        listString2.add("Black");
        listString2.add("White");
        listString2.add("Green");

        Vector<String> listString3 = new Vector<>();
        listString3.addAll(listString);
        listString3.addAll(listString2);

        System.out.println("Merged list: " + listString3);
    }
}
```