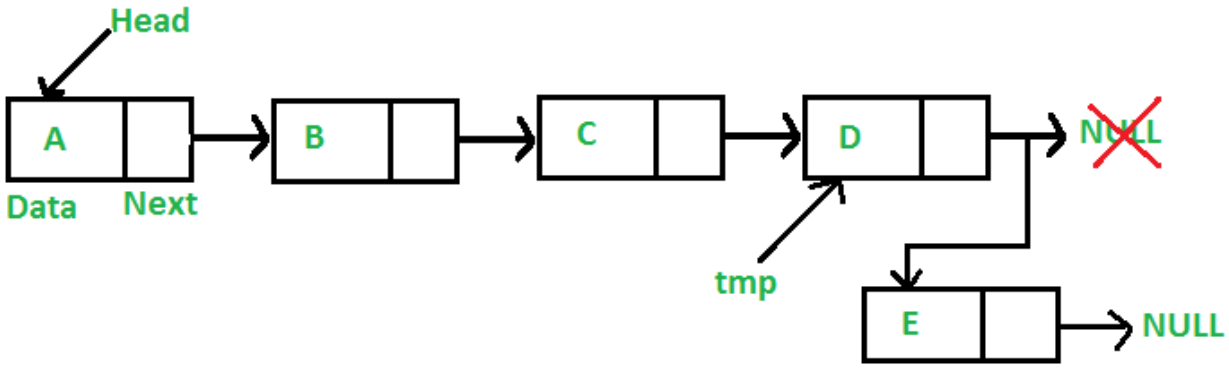
Lists

Lists - (Generic type) This is an interface that describes a sequence of collections, these are linear data structures meaning that the data is in order.

Types of lists:

LinkedList

This is an implementation of list, this is slower than an array because you lose random access, meaning to find the 5 element you need to know where the fourth element is, this works recursively starting from 0. This is because the previous array has a reference to the next array. But this allows the array size to change dynamically. The last item points to null and the first item is called the head. This is not a continues piece of memory. For reading it has a time complexity of O(n) and for inserting it has a time complexity of O(1).




```
public class LetsLinkedList{
  List <Integer> list = new LinkedList<>();

  list.add(1);

  list.get(0);

  list.add(0 , 5) // Index 0 store the value 5
}
```

ArrayList

This is also an implementation of List, but uses array as it's underlying data structure. This works by using an array with a size of 10 at the beginning, after the capacity is full it automatically resizes it. Meaning it makes a new array with the size of 1.5 times the size of the old array and copies the data from the old array to the new array. This is slower than an array when inserting because it might have to resize but it provides a dynamic array. If it it inserting at the end it has a time complexity of O(1) but if it is inserting at the beginning it has a time complexity of O(n), reading would have a time complexity of O(1).

```
public class LetsArrayList{
    List <String> list = new ArrayList<>();

    ist.add("Hello");

    list.get(0);

    list.add(0 , "World");
}
```

LinkedList vs ArrayList

ArrayLists are quicker than LinkedLists when getting data because you still have random access, but LinkedList is quicker when adding/ removing items. Also linkedList are more storage efficient, as you will only have the size of items in the array.

Vectors

This is similar to an arraylist but it is synchronized so only one threads can use it at a time and can't have nulls. It is a part of the Java Collections Framework and is synchronized, which means it is thread-safe and can be safely accessed by multiple threads concurrently. Ensuring data integrity in a multi-threaded environment.

```
import java.util.Vector;

public class VectorExample {
  public static void main(String[] args) {
      Vector<String> vector = new Vector<>();

      vector.add("Apple");
      vector.add("Banana");
      vector.add("Orange");

      System.out.println(vector);  // Output: [Apple, Banana, Orange]

      String secondElement = vector.get(1);
      System.out.println("Second element: " + secondElement);  // Output:
Second element: Banana

      vector.remove(0);
      System.out.println(vector);  // Output: [Banana, Orange]

      boolean containsOrange = vector.contains("Orange");
      System.out.println("Contains Orange? " + containsOrange);  //
Output: Contains Orange? true
  }
}
```