List vs Set Interfaces

wtf the list and the set is called as interface

In Java, both **List** and **Set** are **interfaces** because they define the structure and behavior that classes implementing these interfaces must follow, but they do not provide implementations themselves.

- **List**: This interface defines an ordered collection where elements can be accessed by their index. It allows duplicates. Common implementations include `` `ArrayList` ``, `` `LinkedList` ``, etc.
- **Set**: This interface defines a collection that does not allow duplicate elements. It also doesn't maintain any specific order. Common implementations include `` `HashSet` ``, `` `TreeSet` ``, etc.

By defining them as interfaces, Java allows different classes to implement these behaviors in their own way, giving flexibility in how the underlying collection operates.

i dont understand

Let's break it down step by step:

1. **What is an interface?** An interface in Java is like a blueprint. It defines **what** methods a class must have but not **how** those methods should work. Think of it as a contract that says, "If you're going to be a List or Set, you must follow these rules."
2. **Why are List and Set interfaces?**
    - **List**: It's an interface that defines a collection where elements are stored in a specific order, and you can have duplicates. It doesn’t explain **how** this should be done; it just says, "You need to allow duplicates, and elements should be accessible by index."
    - **Set**: It's an interface that defines a collection where you can't have duplicate elements, but it doesn’t explain how to manage that rule. It just sets the rule.
3. **How does it work?** Classes like `` `ArrayList` `` (for List) or `` `HashSet` `` (for Set) actually **implement** these interfaces. They provide the **how**—the actual code that makes the collection work.

In simple terms:

- **List** and **Set** tell classes what they must do.
- Classes like `` `ArrayList` `` and `` `HashSet` `` actually do the work.

Does this help clarify?

what methods a class must have but not how those methods should work you said this for the definition of interface... in what way way the list and set this is interface.. and where the fuck it has only methods but not the implementation

Let me explain the idea behind **List** and **Set** being interfaces more clearly.

What is an Interface?

- An **interface** defines **what** should be done, but not **how** to do it. In other words, it declares methods without giving the actual code (implementation) for those methods.

Example of an Interface

In Java, an interface looks like this:

```java
interface Animal {
    void sound();  // Just declares that any class implementing this should have a sound() method
}
```

But it doesn't provide any code for `sound()`. Any class that **implements** this interface must provide its own version of `sound()`.

How Does This Apply to `List` and `Set`?

- **List** and **Set** are also **interfaces**, which means they **declare methods** but don't provide the actual code (implementation) for those methods.

Where are `List` and `Set` interfaces in Java?

When you look at `List` and `Set` in Java, they have **method declarations** but not actual implementations. These interfaces define the structure and rules for how classes like `ArrayList` or `HashSet` should work.
Here's an example of `List` interface:

```java
public interface List<E> {
    void add(E element);    // Declares method to add an element
    E get(int index);       // Declares method to get an element by index
}
```

As you can see, there is **no code** for how to add an element or get an element. It just says, "Hey, any class that implements `List` must have these methods."

Where is the Implementation?

The actual **implementation** of these methods is done in classes like `ArrayList` or `LinkedList`. Here's an example of how `ArrayList` implements the `add()` method:

```java
public class ArrayList<E> implements List<E> {
    public void add(E element) {
        // Actual code to add an element in the list
    }
}
```

So:

- **List** and **Set** (interfaces) **declare methods** like `add()` and `remove()` but don't provide any actual code.
- Classes like `ArrayList` and `HashSet` **implement** these methods and provide the code for how these methods should work.

Why Use Interfaces Like List and Set?

Java uses **interfaces** so that many different classes (like `ArrayList`, `LinkedList`, `HashSet`, etc.) can all follow the same "contract" (i.e., the methods defined in `List` or `Set`) but implement them in their own way.

In short:

- **List** and **Set** are interfaces, which means they declare methods (like `add()` or `remove()`) but don't provide the code.
- Classes like `ArrayList` and `HashSet` implement these interfaces and provide the actual code for these methods.

Does that clear things up?

what these fucking things
```
public class ArrayList<E> implements List<E> {
    public void add(E element) {
        // Actual code to add an element in the list
    }
}
```
i have never seen this code

No worries, I'll explain step-by-step what that code means. This type of code shows **how Java works under the hood** when you're using things like `ArrayList` and `List`.

The Code You Mentioned

```java
public class ArrayList<E> implements List<E> {
    public void add(E element) {
        // Actual code to add an element in the list
    }
}
```

This is a simplified version of how the **ArrayList** class is implemented in Java. You may not have seen this exact code before because usually, when you use `` `ArrayList` ``, you just do something like:

```java
List<String> list = new ArrayList<>();
list.add("Hello");
```

But let's break down what's going on in the code you shared.

Code Breakdown

1. **`` `public class ArrayList<E> implements List<E>` ``:**
    - This line declares a class called `` `ArrayList` ``.
    - `` `<E>` `` is called a **generic type**. It means that `` `ArrayList` `` can store elements of any type. For example, it can store `` `Strings` ``, `` `Integers` ``, etc.
        - If you say `` `ArrayList<String>` ``, then `` `E` `` becomes `` `String` ``. If you say `` `ArrayList<Integer>` ``, then `` `E` `` becomes `` `Integer` ``.
    - `` `implements List<E>` `` means that `` `ArrayList` `` is implementing the `` `List` `` interface. In other words, `` `ArrayList` `` is providing the actual code (implementation) for the methods that the `` `List` `` interface declares.
2. **`` `public void add(E element) { }` ``:**
    - This is a method inside the `` `ArrayList` `` class. It's part of the implementation of the `` `List` `` interface.
    - The `` `add()` `` method is used to add elements to the list. But in this example, the method doesn't show any actual code yet (just a comment). In real code, this method would have logic to add an element to the internal data structure.

Real-life Use Case: How You Usually Use ArrayList

When you use `` `ArrayList` `` in your program, you don't need to see the actual code of the `` `ArrayList` `` class. Here's what you usually write:

```java
List<String> myList = new ArrayList<>();
myList.add("Hello");
myList.add("World");
```

- When you call `` `add()` ``, the `` `ArrayList` `` class behind the scenes has the **actual code** that takes care of adding the element `` `"Hello"` `` or `` `"World"` `` into the list.

Why You Don’t See This Code

- You don’t usually see the code of `ArrayList` because **Java hides it from you**. You just use the class to **add**, **remove**, or **get** elements from the list. Java already did the hard work of implementing all those methods.
- The Java Development Kit (JDK) contains this code. If you want to see it, you can look at the source code for `ArrayList` in the JDK, but that’s not necessary for using it.

What’s the Point of the Code?

This code example was to show you **how Java’s `ArrayList` implements** the `List` interface. The interface declares what methods should exist (`add()`, `remove()`, etc.), and the class like `ArrayList` provides the real code for how to do that.

In short:

- You usually don’t see this code because Java takes care of it.
- When you use `ArrayList`, Java internally runs the code that adds, removes, or gets elements from the list, but you don’t need to worry about how it works.

Would you like to see how it works step-by-step with a real code example that you can run yourself?

ChatGPT can make mistakes. Check important info.