174.Generics

At 11:09 of Lecture 174, (Putting it all together, Final Section Challenge) Tim uses `matches.sort(null)`. Why are we using null? Why aren't we using a variable instead?

Tim uses `null` as the argument for the `sort()` method in the line `matches.sort(null)`. The reason for using `null` is to utilise the default natural order sorting provided by the `Comparable` interface implemented by the elements of the `matches` list. In Java, the `sort()` method can accept a `Comparator` object as an argument to define a custom sorting order. However, when `null` is passed as the argument, the `sort()` method uses the natural ordering of the elements, which is determined by their implementation of the `Comparable` interface. In the lecture, the `Student` class implements the `Comparable` interface and overrides the `compareTo()` method to define the natural ordering based on the student ID. By passing `null` as the argument to `sort()`, Tim is instructing the method to use the natural ordering defined by the `compareTo()` method in the `Student` class. If a custom sorting order using a `Comparator` was desired, a specific `Comparator` object would be passed as an argument to the `sort()` method instead of `null`.

At 05:57, how can we use Comparator.naturalOrder() without implementing the Comparator interface?

`Comparator.naturalOrder()` is used without implementing the `Comparator` interface because it is a static method provided by the `Comparator` interface itself. The `naturalOrder()` method returns a `Comparator` that imposes a natural ordering on objects that implement the `Comparable` interface.

In other words, if the objects in a list or collection implement `Comparable`, we can use `Comparator.naturalOrder()` to sort them in their natural order without explicitly implementing the `Comparator` interface. This is possible because the `naturalOrder()` method is already implemented to compare objects that implement the `Comparable` interface.

In the code from the lecture, the `LPAStudent` and `Student` classes implement the `Comparable` interface to provide natural ordering by their `studentId` field, so `Comparator.naturalOrder()` can be used to sort them by their natural order without implementing the `Comparator` interface.

At 03:01, Tim uses the this:

In Tim's code, he uses the `this` keyword to iterate over the elements of the current instance of the class. In Java, `this` simply refers to the current instance of the class you're working with. So, when Tim writes `for (var item : this)`, he is basically looping through the elements of the current object, which is an instance of a class implementing the `Iterable` interface.

Using `this` here is pretty common in Java, especially when dealing with collections or custom iterable classes. It is just a way to emphasize that the loop is happening within the current object, which can be helpful for clarity, especially in larger or more complex codebases where it might not be immediately obvious what object is being looped over.