

ArrayList Hierarchy

Learning Objective

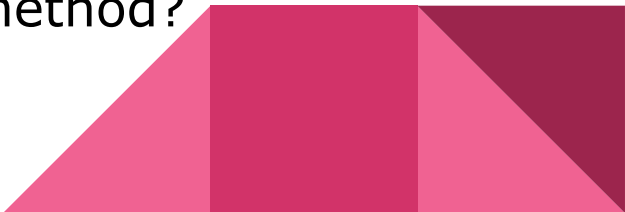
Apply inheritance to ArrayLists to implement subclasses with a specific behavior.

Agenda

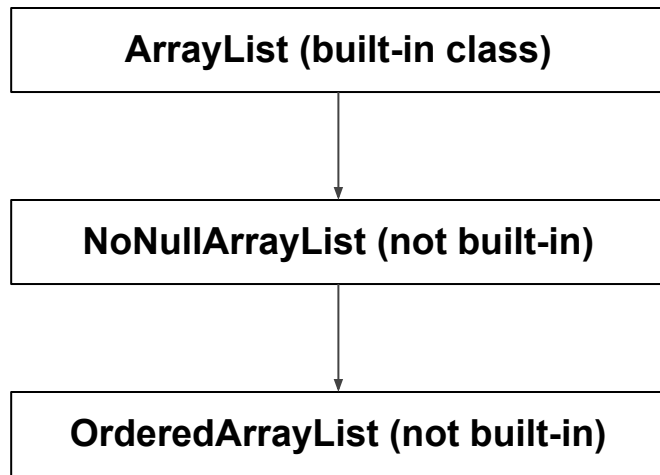
- Do Now activity
 - Mini Lesson:
 - Explain ArrayList Hierarchy
 - What classes do we need?
 - What constructors do we need?
 - What methods should be overridden?
 - Practice:
 - Implement ArrayList Hierarchy in Java (2 days)
-

Do Now

Discuss with a partner:

- About a way to prevent you from adding a null item to an ArrayList.
 - How would you do to keep your ArrayList sorted everytime you add a new item without using any built-in method?
- 

ArrayList Hierarchy



NoNullArrayList Characteristics

Considering that NoNullArrayList **IS-A** ArrayList:

- What constructors should NoNullArrayList have?
- Which basic methods should be overridden to prevent having null elements in the list?
- What strategy would you use to prevent having null elements?



Tips to implement a NoNullArrayList

- NoNullArrayList **IS-A** ArrayList (NoNullArrayList inherits from ArrayList).
- ArrayList uses any type of data. You must get that behavior in your NoNullArrayList by declaring you class like this:

```
public class NoNullArrayList<T> extends ArrayList<T>{  
  
}
```

<T> indicates the generic type that will be used.

- Use different test cases to try your code.



OrderedArrayList Characteristics

OrderedArrayList maintains the elements sorted in a list when they are added.

Considering that OrderedArrayList **IS-A** NoNullArrayList:

- What constructors should OrderedArrayList have?
- Which basic methods should be overridden to have all elements sorted in order?
- How would you override the method set?



Tips to implement an `OrderedArrayList`

- `OrderedArrayList` **IS-A** `NonNullArrayList` (`OrderedArrayList` inherits from `NonNullArrayList`).
- Your code should allow `T` to use the `compareTo()` method which compares an object with another, and returns a numerical result based on the comparison. If the result is negative, this object sorts less than the other; if 0, the two are equal, and if positive, this object sorts greater than the other (Example: `o1.compareTo(o2)`).

```
public class OrderedArrayList<T extends Comparable<T>> extends  
    NonNullArrayList<T>{  
  
}
```

<T extends Comparable<T>> means that the type `T` must implement the `Comparable` interface, which ensures objects of type `T` can be compared with one another.

<https://docs.oracle.com/javase/6/docs/api/java/lang/Comparable.html>



Coding Time!!!

