



Week 16 Collection

Teaching Team of
Algorithm and Data Structure
Information Technology Department

Learning Objectives



Students must have a good understanding on basic concept of Collection and the usage



Students must be able to implement Collection provided by Java to solve the problem

Outline

Java Collection Framework

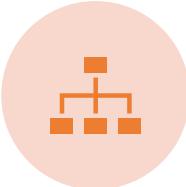
Usage of Java Collection Framework

Sorting dan Searching

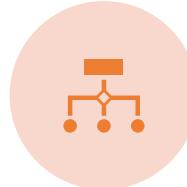
Subinterface Collection

Questions

Introduction



Remember that List, Stack and Queue are linear data structures, while non-linear data structure are tree and graph.



Linear List is used for serial data, while non-linear means that the data is presented in a non-serial form (hierarchical or irregular). example: queue, name of day of week, name of month of year, social media network, network topology, and others.



Please imagine if the example above is made using conventional techniques that only utilize the concepts of classes and objects.



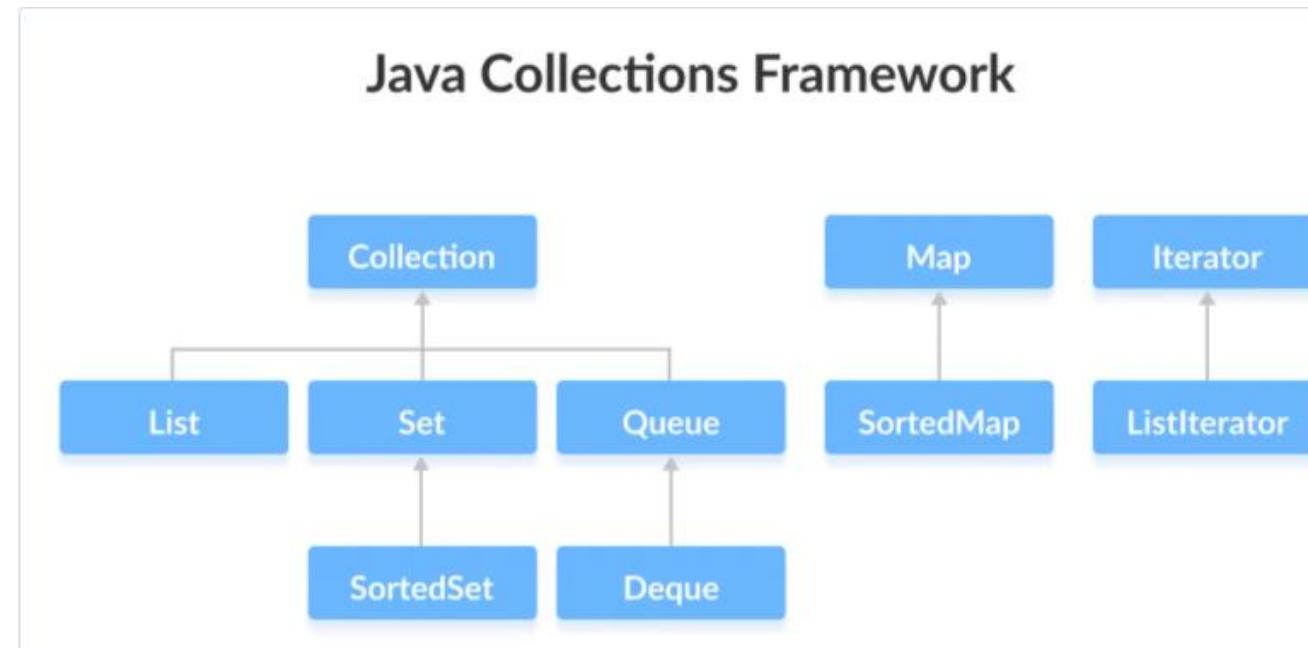
Concepts are very important before learning more advanced and easy techniques.



Structure types both linear and non-linear will be wrapped into the Java Collection Framework.

Java Collection Framework

A Java Collection Framework provides a collection of interfaces or classes that implement data structure algorithms



<https://www.programiz.com/java-programming/collections>

Usage of Java Collection Framework

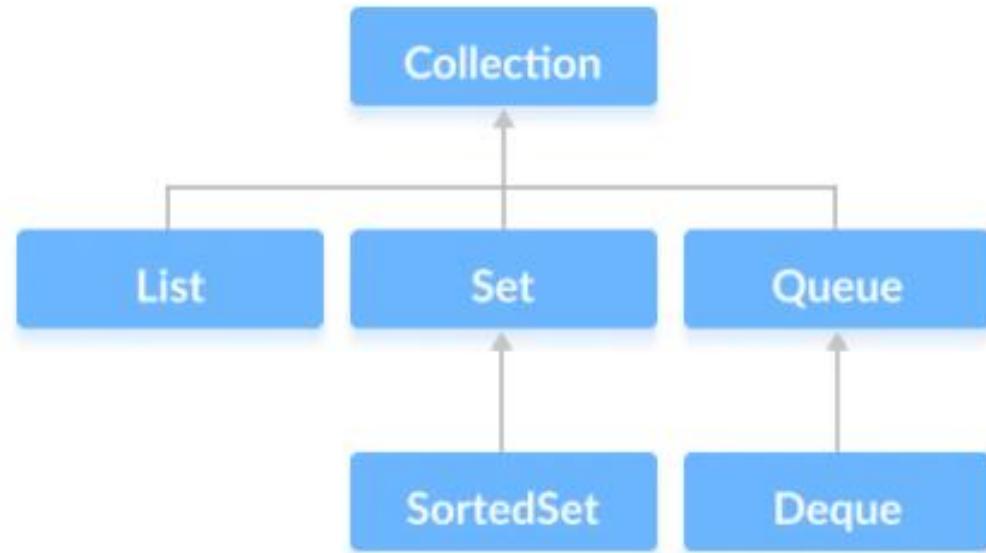
The Java Collection Framework provides data structures and algorithms that we can use directly. The advantages/benefits are as follows

- We will not manually code when implementing data structure algorithm
- The code we create, will be much more efficient
- We can also use certain java collections according to the data structure. For example, when we want to hold unique data, use the collection Set.

Sorting dan Searching Java Collection Framework

- The Java Collection Framework provides standard operations in data structures, such as **sorting** and **searching**.
- Both functions are contained in the **java.util.Collections** package.
- To sort data using static functions, namely **sort()**, **Collections.sort()**.
- The **sort()** function in Collections implements the **merge sort algorithm**.
- To search data can use **Collections.binarySearch()**

Java Collection Interface



<https://www.programiz.com/java-programming/collection-interface>

- The **Collections** interface is the root interface of the Java collections framework
- To implement it we have to create sub-interfaces, such as **List**, **Set**, and **Queue**. Suppose the **ArrayList** class implements the **List** which is a sub-interface of **Collection**

Subinterface Collection

The Collection interface contains subinterfaces that can be implemented by various classes in Java

- Interface List

A collection or data structure that allows adding/removing elements just like an array.

- Interface Set

With Interface Set we can store elements in a collection without any duplication of data

- Interface Queue

Used to store and access elements in a First In, First Out (FIFO) rule.

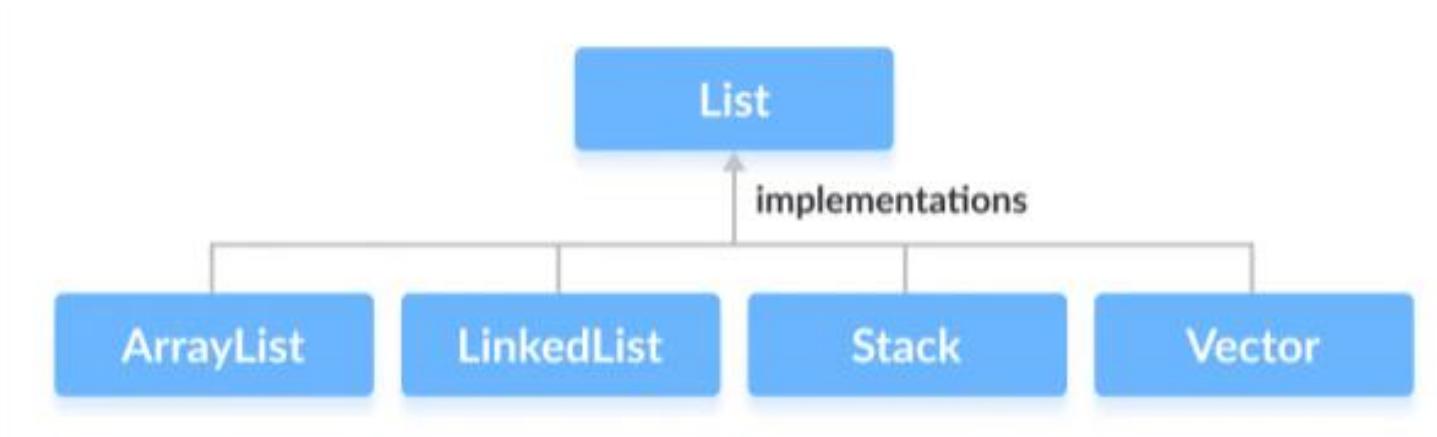
Metode/Fungsi Collection

Interface Collection has functions/methods that can be used to simplify data structures, these methods also apply to subinterfaces.

- `add()` – add element
- `size()` – returns the size of the collection
- `remove()` – remove element
- `iterator()` – returns an iterator to access the elements (iterate over the elements)
- `addAll()` – add all elements(collection) in collection
- `removeAll()` – removes all elements from the specified collection
- `clear()` – removes all elements

Interface List

The List interface is an ordered collection that allows you to store and access elements sequentially. Some classes that implement the List interface are ArrayList, LinkedList, Vector, and Stack.



<https://www.programiz.com/java-programming/list>

Implement List

Because List is an interface, so we cannot directly create an instance/object from the interface.

```
List list = new ArrayList(); (1)  
List<String> list = new Stack<>(); (2)
```

- To use List you have to **import java.util.List**
- The list object cannot be instantiated from a **List** but from the implementation class i.e. **ArrayList**
- The first form is an **ArrayList** which can handle all data types
- The second form is a stack which can only handle String.
- The <> symbol is a generic or read of type String.

Class ArrayList

ArrayList is a class of the Java Collection Framework that provides array-like functions, but it is dynamic in size.



<https://www.programiz.com/java-programming/arraylist>

How to Implement ArrayList

- First thing to do is import `java.util.ArrayList`

```
ArrayList<Type> arrayList= new ArrayList<>(); (1)
ArrayList<Integer> arrayList = new ArrayList<>(); (2)
ArrayList<String> arrayList = new ArrayList<>(); (3)
```

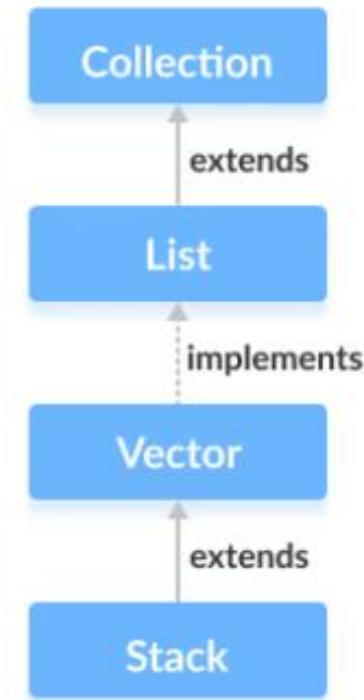
- The general syntax is shown in the first line, Type is a wrapper class so that a specific `arrayList` object handles objects of a certain class
- The second line means that the `arrayList` object can only store integer type data
- The third line means that the `arrayList` object can only store string type data
- If it is not specified or there is no Type then the `arrayList` object can hold any data. 

Methods of ArrayList

- `size()` – returns the length of the arraylist
- `sort()` – sorts arraylist elements
- `clone()` – creates a new arraylist with the same elements, size, and capacity from an arraylist.
- `contains()` – looks for an element in an arraylist and returns a boolean value
- `ensureCapacity()` – specifies the arraylist elements that can be accommodated
- `isEmpty()` – check if an arraylist is empty
- `indexOf()` – returns the index of the element in an arraylist

Class Stack

- Stack is a class of the Java Collection Framework that provides stack functions on data structures.
- On a stack, elements can be added or accessed using the Last In First Out (LIFO) concept.
- The element is added(push) to the top stack and taken(pop) from the top stack.



<https://www.programiz.com/java-programming/stack>

How to Use Stack

- First thing to do is **import java.util.Stack**

```
Stack<Type> stacks = new Stack<>(); (1)
Stack<Integer> stacks = new Stack<>(); (2)
Stack<Mahasiswa> stacks = new Stack<>(); (3)
```

- The general syntax is shown on the first line.
- The second line means that the stacks object can only store integer type data
- The third line means that the stacks object can only store objects instantiated from the Student class.

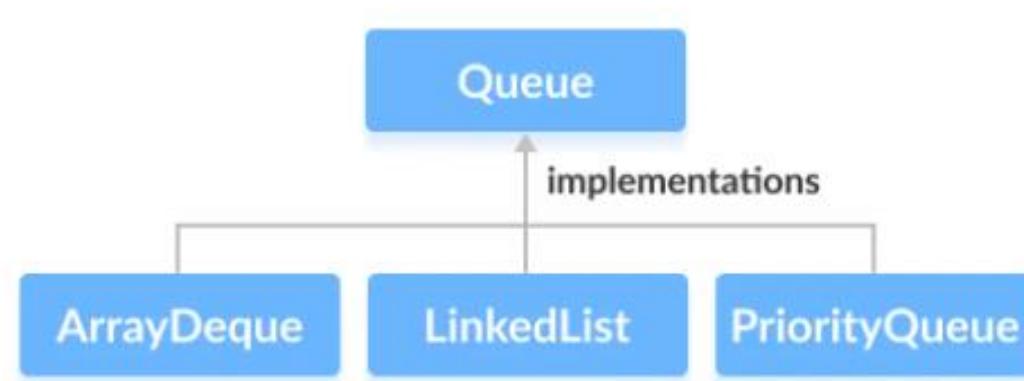
Methods of Stack

All functions/methods owned by the Vector class are owned by the Stack class, the difference is as follows

- `push()` – add element to stack
- `pop()` – removes the element on the stack
- `peek()` – returns the element at the top of the stack
- `search()` – returns the position of the element on a stack
- `empty()` – checks if a stack is empty

Interface Queue

- Queue is an interface of the Java Collection Framework that provides queuing functions in data structures.
- Queue is an interface so it cannot be directly implemented, instantiated
- In queues, elements can be added from the back and accessed/removed from the front, First In First Out(FIFO)



How to Use Queue

- First thing to do is **import java.util.Queue**

```
Queue<Type> queues = new LinkedList<>(); (1)
Queue<Integer> queues = new ArrayDeque<>(); (2)
Queue<Mahasiswa> queues = new PriorityQueue<>(); (3)
```

- The general syntax is shown on the first line.
- `ArrayDeque` is a class that implements the `Queue` and `Deque` interfaces. `Deque` itself is a subinterface of `Queue`, the difference is that it can add/remove elements not only from the front but also from the back
- `PriorityQueue` is an implementation of `Interface Queue`, but the element that has the smallest value is always placed at the front, even if it is not added the first time.

Methods of Queue

All functions/methods that are owned by the Collection interface are also owned by the Queue, the difference is as follows

- `add()` – add a specified element to the queue
- `offer()` – add a specified element to the queue
- `element()` – returns the element that is at the head of the queue
- `peek()` – returns the element that is in the head of the queue
- `remove()` – returns and removes head from queue
- `poll()` - returns and removes heads on queue

Class TreeSet

- A class from the Java collection framework that provides the functionality of a tree data structure.
- This type of data structure is not allowed to hold the same data
- Actually this class is not as specific and detailed as the tree concept that has been studied previously, but can be used for simple tree data structures.



[https://www.programiz.com
/java-programming/treeset](https://www.programiz.com/java-programming/treeset)

Instantiation of TreeSet

- First thing to do is import java.util.TreeSet

```
TreeSet<Integer> trees = new TreeSet<>();
```

- When the instance is not given an argument, the data is automatically sorted in ascending order.

Methods of TreeSet

All functions/methods owned by the Collection interface and its derivatives will automatically be owned by the TreeSet class.

- `add()` – adds a specified element to the tree
- `addAll()` – adds a specified element to the queue
- `element()` – returns the element that is at the head of the queue
- `peek()` – returns the element that is in the head of the queue
- `remove()` – returns and removes head from queue
- `poll()` - returns and removes heads on queue

Framework Graph



Java does not have a specific class that implements the operations of graph, it is recommended to create your own class to implement graph



We have studied the classes created in the previous lesson to implement graph.



Several libraries that can be used to support graph operations

JGraphT, <https://jgrapht.org/>
Google Guava, <https://github.com/google/guava/wiki/GraphsExplained>
Apache Commons, <https://commons.apache.org/sandbox/commons-graph/>

Questions



Explain the differences and similarities between List and Set. Then in what cases will they be utilized?



What is the difference between the clear() and removeAll() functions in the List interface and explain why?



What is the difference between add() and offer() function in Queue?



Why do you think Java doesn't have a specific class that handles Graph operations?