

# Data Structures

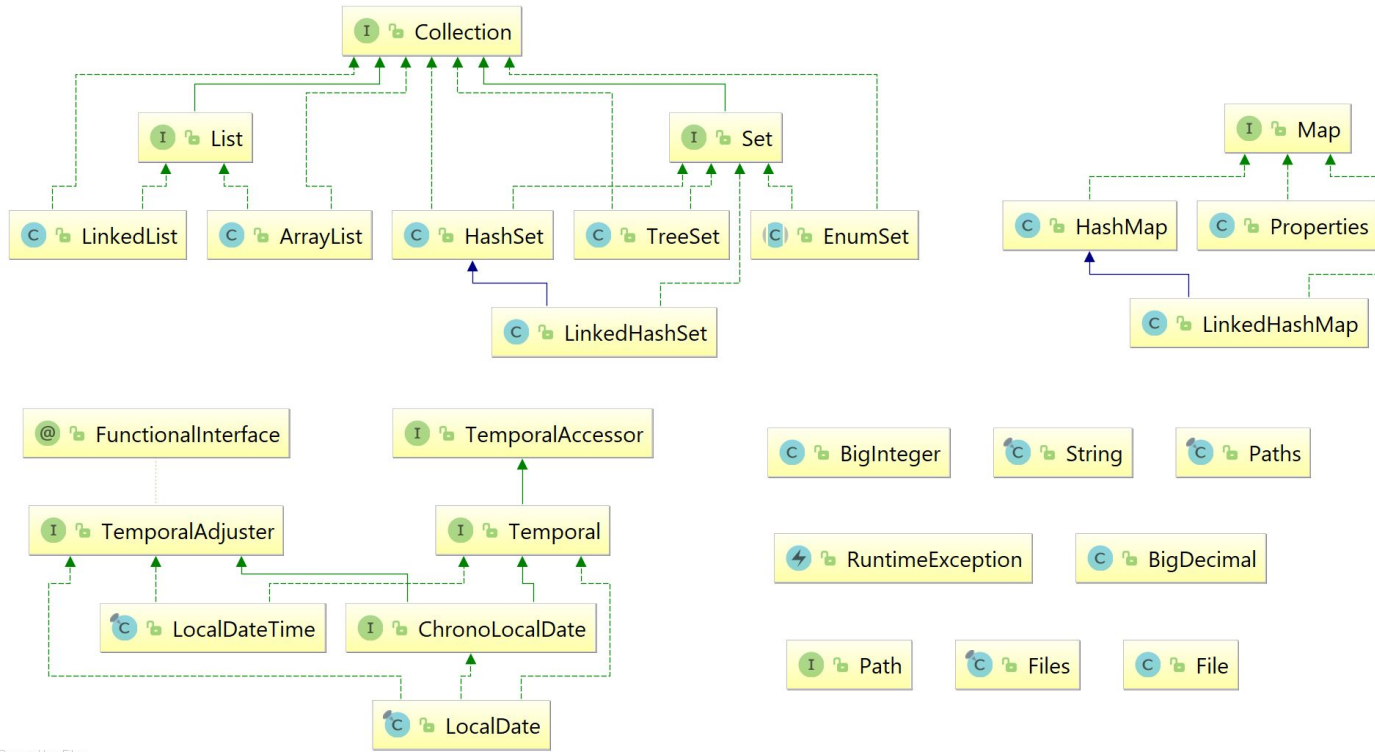


Maciej Koziara

# What is data structure?

*Data structure is designed to organize data to suit a specific purpose so that it can be accessed and worked with in appropriate ways.*

# Java data structures



# Let's get started!



# ArrayList - summary

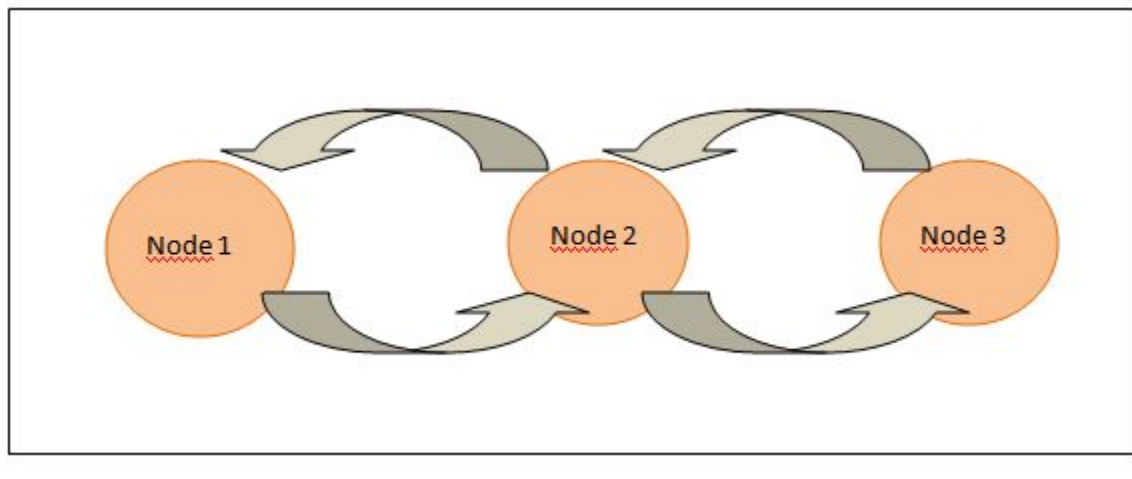
- // **List** implementation
- // Most often used in everyday situations
- // Based on array of objects (**Object[]**)

# Iterable and Iterator

- // **Iterator** allows to traverse through all collection elements
- // Class should implement **Iterable** interface, if we want to use it with **foreach** syntax
- // Every **Collection** is Iterable by default

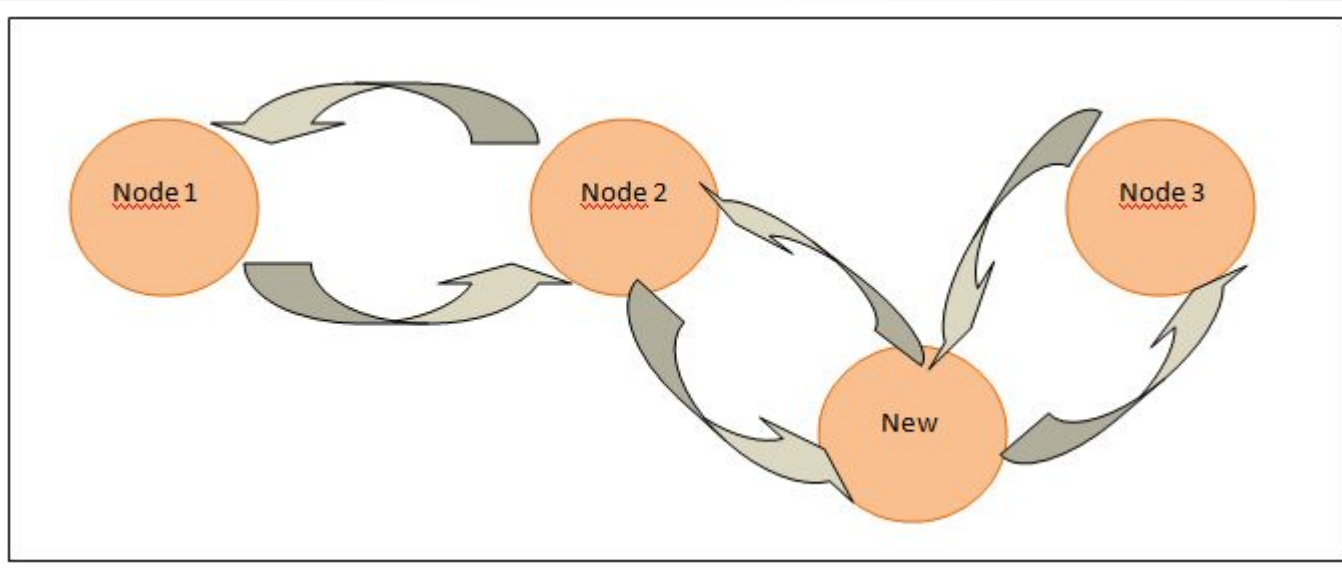
# LinkedList - Theory

- // Consists of **nodes** connected to each other
- // Each **node** knows only about next and previous **node**



# LinkedList - Theory

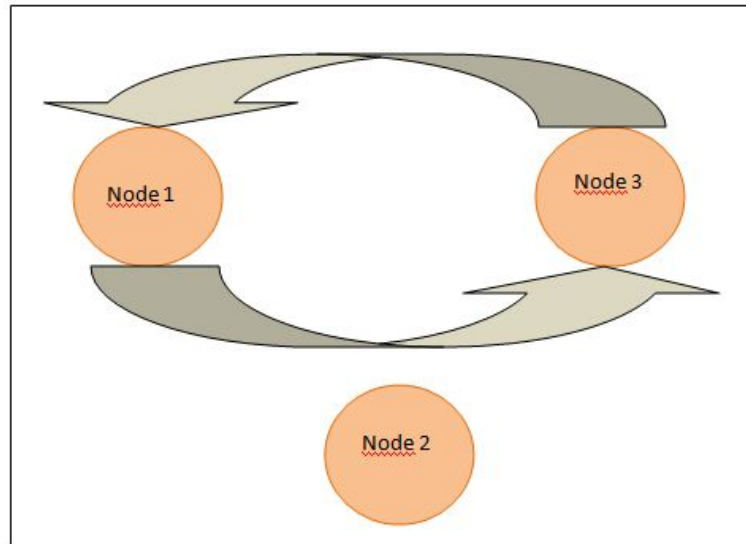
**Add new node:**





# LinkedList - Theory

**Remove node:**



# List - summary

	ArrayList	LinkedList
<i>get()</i>	Very fast	Needs to traverse whole list to find element
<i>remove()</i>	Slower but still fast	Very fast
<i>add()</i>	Slower but still fast	Very fast
When should I use?	Always	Only in specific cases

# Hashing

- // Process of converting an object into integer form
- // Should always return same result when invoked on same object
- // When two objects are equal their hashCode methods should produce the same result
- // Use Objects.hash() method to determine hash value instead of writing it by yourself

# Let's code - again!



## Map<K, V> - summary

- // Used for creating simple relations between objects or group values together
- // One of the uses might be summarizing different kind of data
- // **HashMap** - most common implementation. Requires valid *equals()* and *hashCode()* method implementations
- // The only **Collection** that does not implement **Collection** interface

# Map<K, V> - internals

- // Based on entries (key-value pairs) and buckets
- // Uses hashCode to decide to which bucket entry should be placed
- // If hashCodes of entries are the same uses equals to make sure given Keys are really the same
- // Based on entries and buckets

# Sorting elements

// **Comparator<T>** - class that implements this interface may be used for sorting values of different class

// **Comparable<T>** - lets us define a natural ordering of a given class

// External **Comparator** should be used when we want to sort values in a different way than natural ordering

// Most Java classes implements **Comparator** interface

## Set<T> - summary

// Does not contain duplicates

// Only allows us to iterate over its elements, no positional access

Most common usage scenario is when we want to be sure that values we work with are unique.



# Set<T> - implementations

// **HashSet** - do not guarantee any iteration order. Fastest and used in most situation. For proper work required *equals()* and *hashCode()* implementations. Internally based on HashMap.

// **TreeSet** - Slowest implementation. Guarantees natural ordering iteration order. Requires class to implement **Comparator** interface

// **TreeSet** - Speed between **HashSet** and **TreeSet**. Guarantees inserting order while iterating over it.