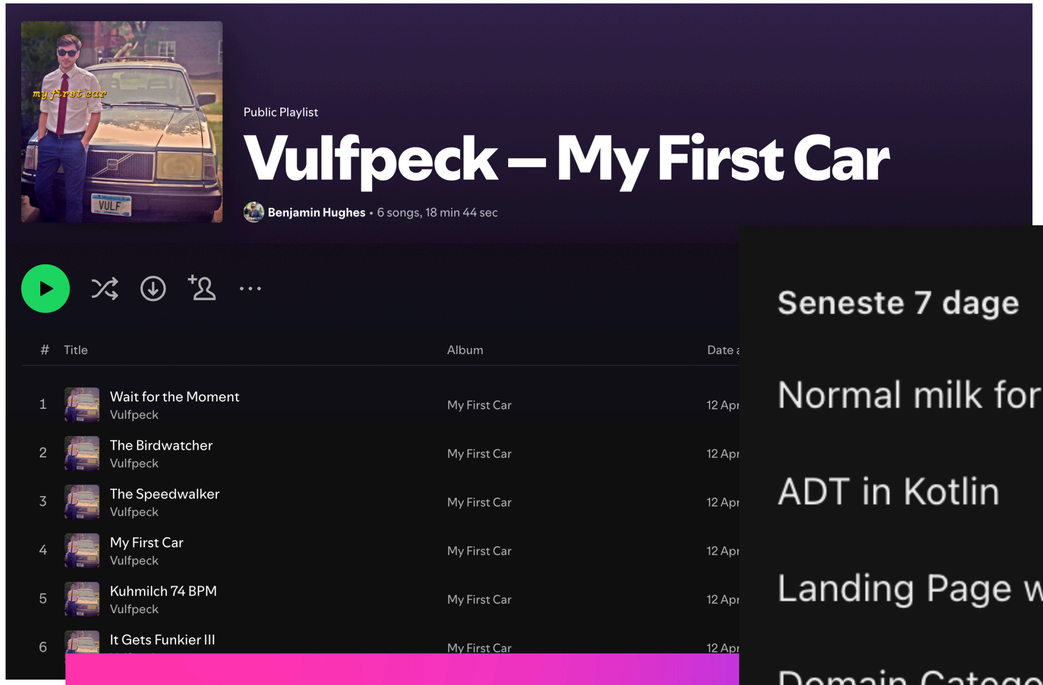
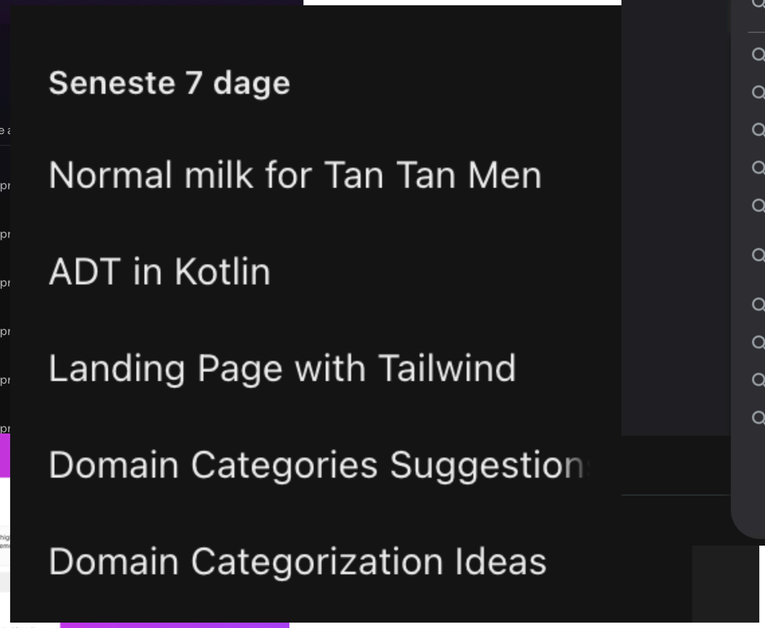
Data structures

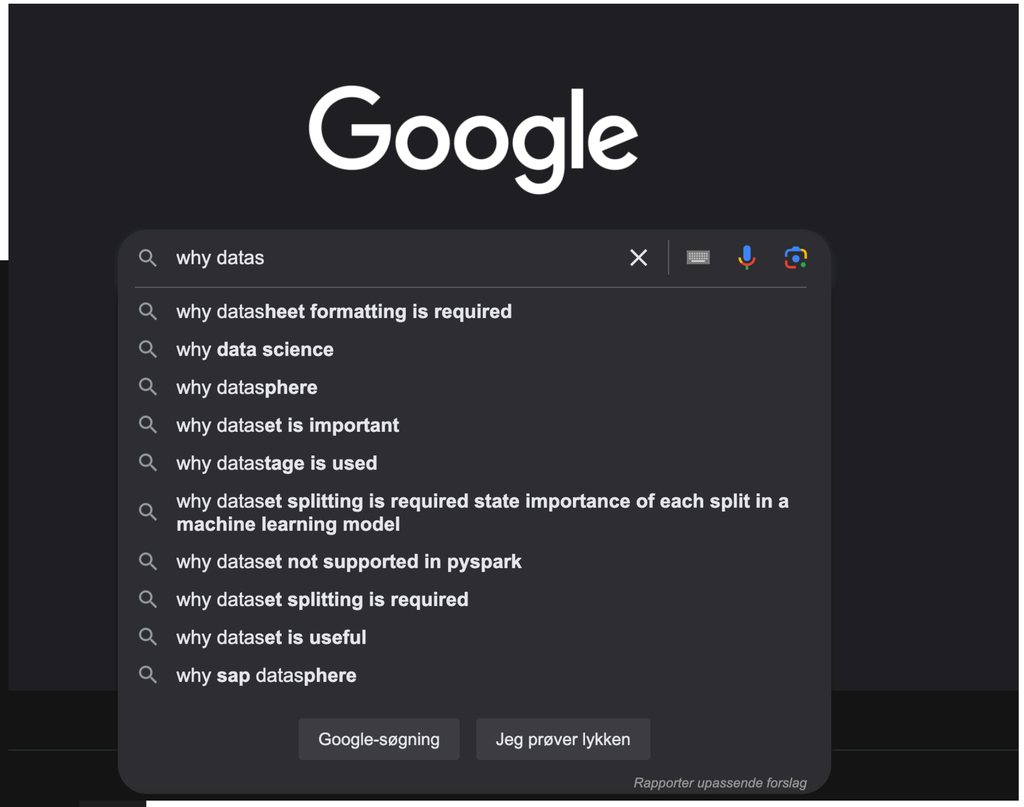
Applikationsudvikling: CS101

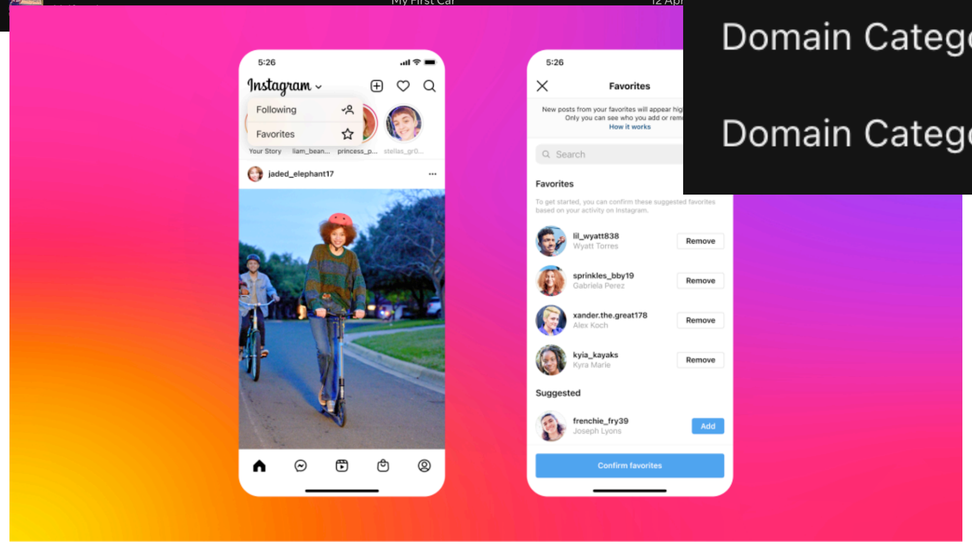
NIFR - https://github.com/nicklasdean/ita-2023-2sem-code-examples

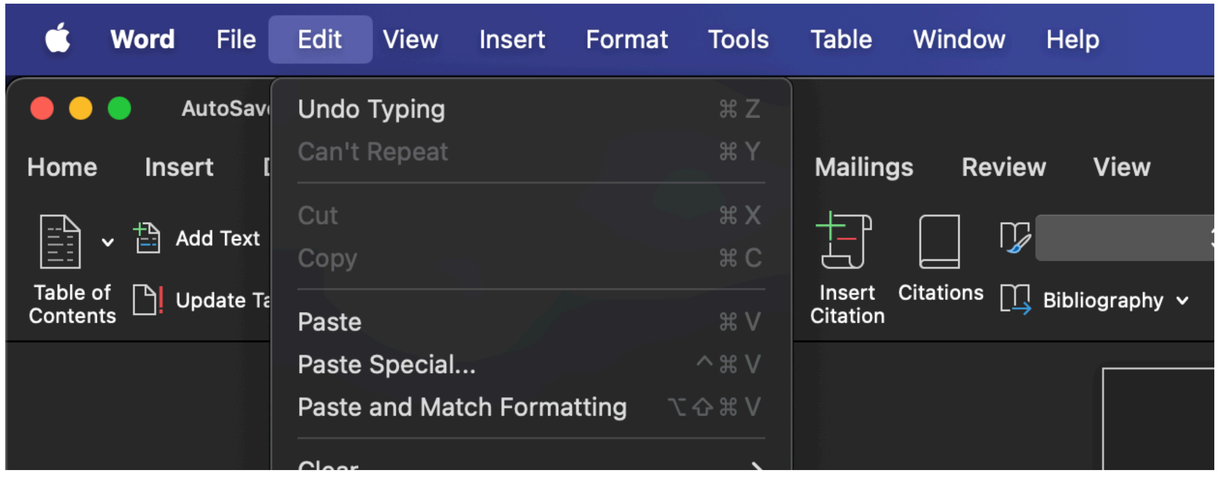
Hvorfor overhovedet snakke om datastrukturer?


Public Playlist
Vulfpeck – My First Car
Benjamin Hughes • 6 songs, 18 min 44 sec
Title Album Date
1 Wait for the Moment Vulfpeck My First Car 12 Apr
2 The Birdwatcher Vulfpeck My First Car 12 Apr
3 The Speedwalker Vulfpeck My First Car 12 Apr
4 My First Car Vulfpeck My First Car 12 Apr
5 Kuhmilch 74 BPM Vulfpeck My First Car 12 Apr
6 It Gets Funkier III My First Car 12 Apr


Seneste 7 dage
Normal milk for Tan Tan Men
ADT in Kotlin
Landing Page with Tailwind
Domain Categories Suggestion
Domain Categorization Ideas


Google
why datas
why datasheet formatting is required
why data science
why datasphere
why dataset is important
why datastage is used
why dataset splitting is required state importance of each split in a machine learning model
why dataset not supported in pyspark
why dataset splitting is required
why dataset is useful
why sap datasphere
Google-søgning
Jeg prøver lykken
Rapporter upassende forslag


Instagram
Following
Favorites
jaded_elephant17
Favorites
Search
lil_wyatt838
sprinkles_bby19
xander.the.great178
kyia_kayaks
Remove
Suggested
frenchie_fry39
Add
Confirm favorites


Word File Edit View Insert Format Tools Table Window Help
Undo Typing ⌘ Z
Can't Repeat ⌘ Y
Cut ⌘ X
Copy ⌘ C
Paste ⌘ V
Paste Special... ^ ⌘ V
Paste and Match Formatting ⌥ ⇧ ⌘ V
Home Insert
Add Text
Table of Contents
Mailings Review View
Insert Citation
Citations
Bibliography

Abstract Data Types (ADT)

Data Structures

- List
  - ArrayList
  - LinkedList
- Map
  - HashMap
- Set
  - HashSet
  - LinkedSet

- Show the ADT
- Implementation

An **Abstract Data Type** (ADT) is the specification of a group of operations that make sense for a given data type. They define an interface for working with variables holding data of a given type—hiding all details of how data is stored and operated in memory.

The List

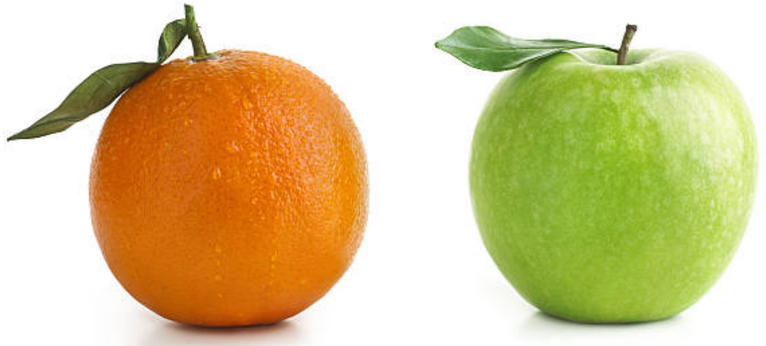
When storing a bunch of items, you sometimes need more flexibility. For instance, you could want to freely reorder the items; or to access, insert and remove items at any position. In these cases, the **List** is handy. Commonly defined operations in a List ADT include:

- **`insert(n, e)`**: insert the item `e` at position `n`,
- **`remove(n)`**: remove the item at position `n`,
- **`get(n)`**: get the item at position `n`,
- **`sort()`**: sort the items in the list,
- **`slice(start, end)`**: return a sub-list slice starting at the position `start` up until the position `end`,
- **`reverse()`**: reverse the order of the list.

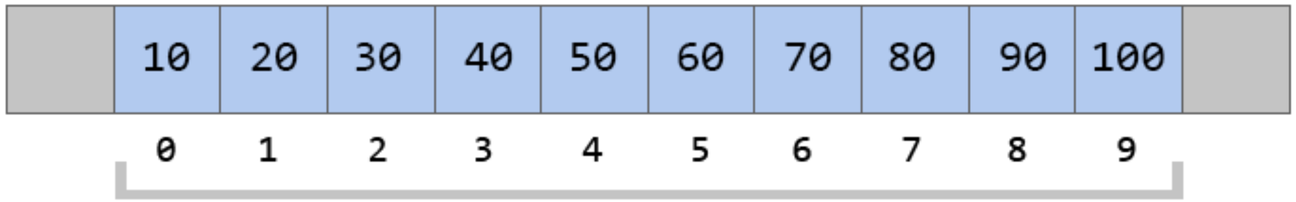
List (ArrayList / LinkedList)

Kotlin list collections

- List is the interface
- ArrayList / LinkedList is the implementation
- Differ in implementation but adheres to the same interface
- Can be instantiated as mutable / unmutable

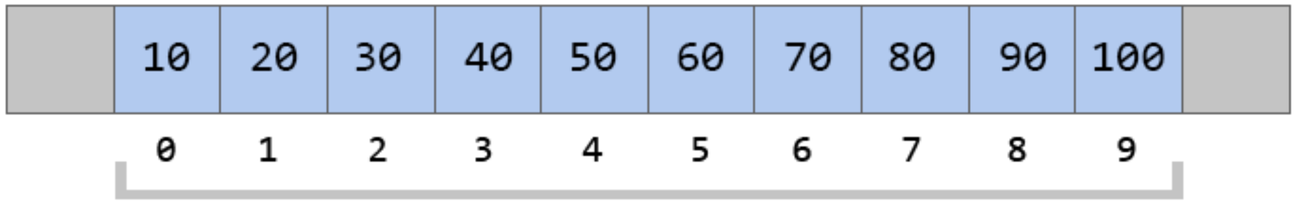


Array



Fixed size

ArrayList

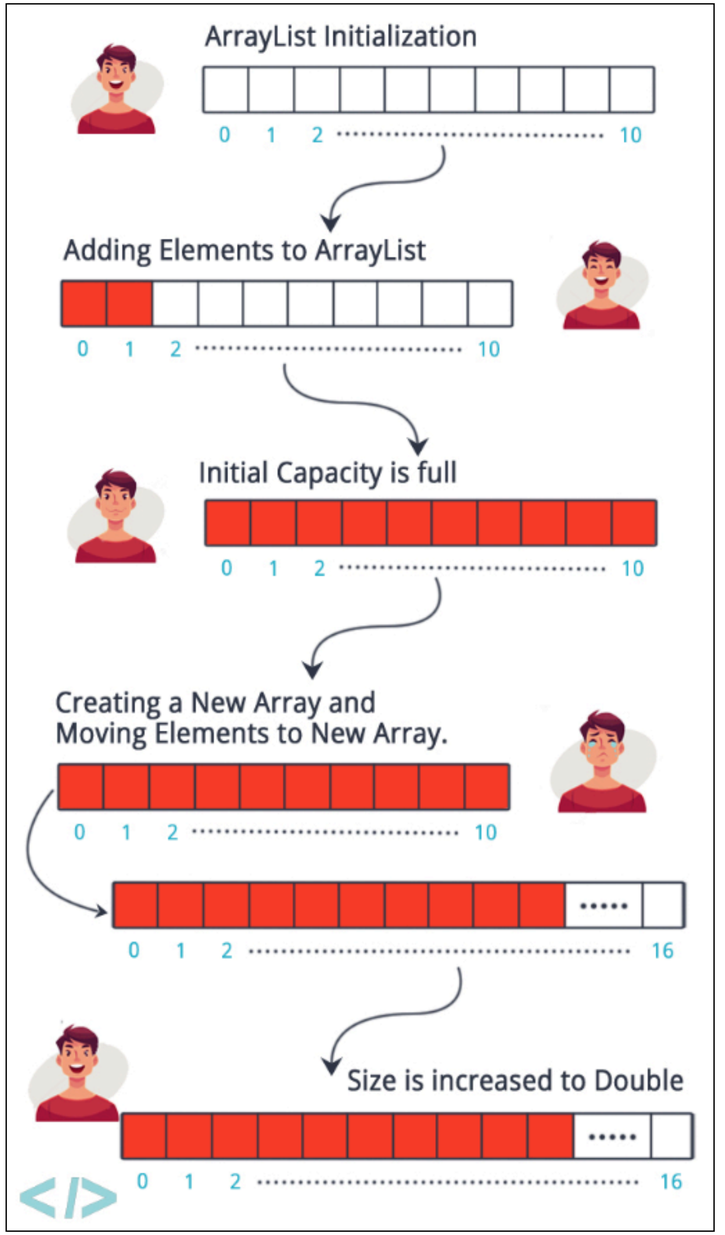


Dynamic size

- Difference between ArrayList and MutableList?

ArrayList

ImplementationADT

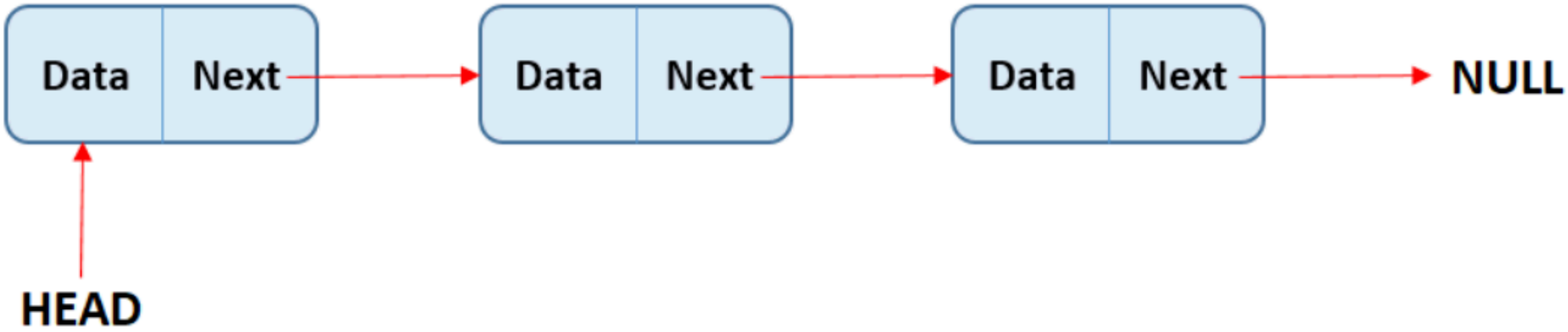


ArrayList

- What is big O of
  - Inserting an element?
  - Finding the size?
  - Finding an element in the list

LinkedList

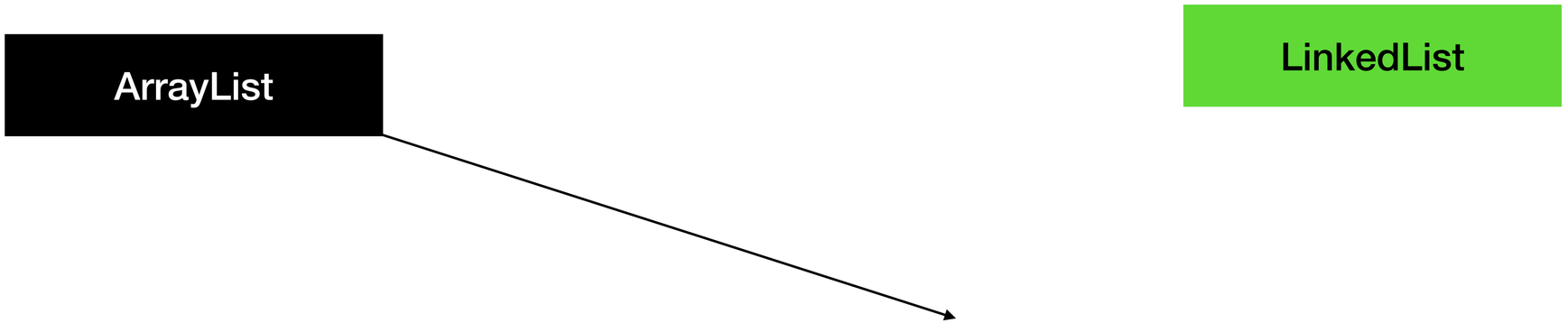
ImplementationADT



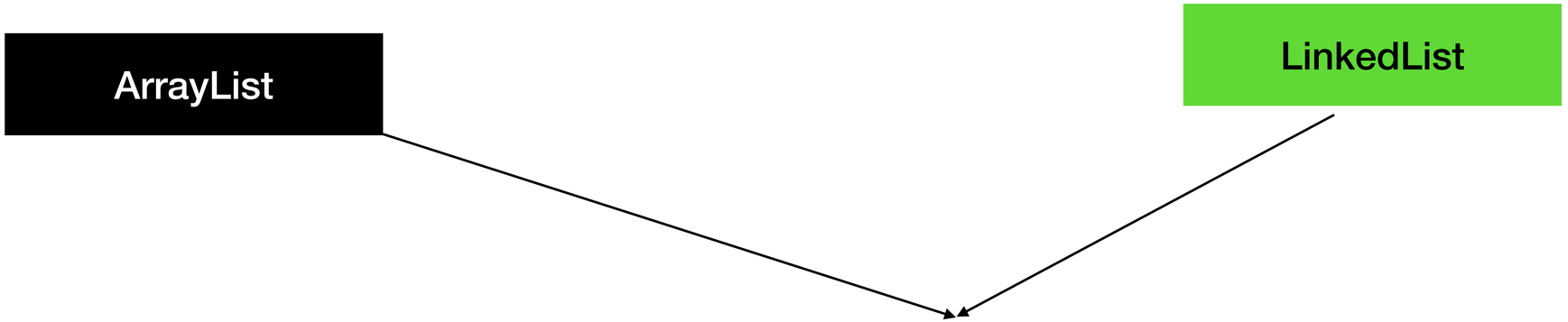
LinkedList

ImplementationADT

- What is big O of
  - Inserting an element?
  - Finding the size?
  - Finding an element in the list




```
fun doSomethingWithLists(list: ArrayList<String>){
    //List operations
}
```

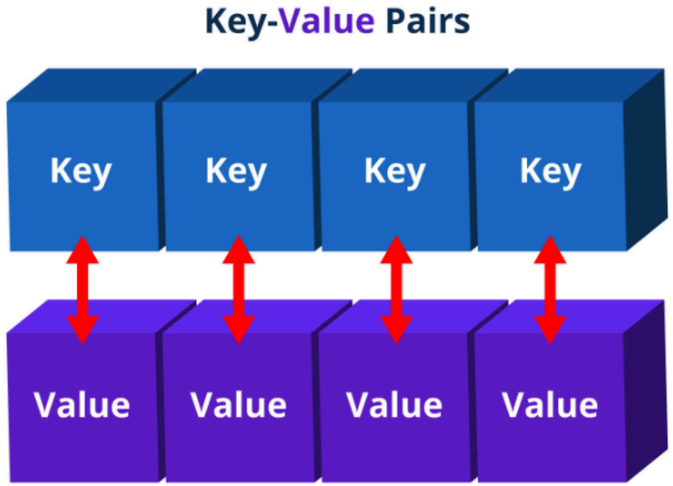



```
fun doSomethingWithLists(list: List<String>){
    //List operations
}
```

Abstract Data Types

Map Type

Javascript object-like



HashMap

ImplementationADT

- **`set(key, value)`**: add a key-value mapping,
- **`delete(key)`**: remove **`key`** and its associated value,
- **`get(key)`**: retrieve the value that was associated to **`key`**.

```
val hashMap: HashMap<String, Int> = HashMap<String, Int>()

// Adding key-value pairs
hashMap["John"] = 25
hashMap["Alice"] = 30
hashMap["Bob"] = 35

//Returns 35
val bobAge: Int = hashMap.get("Bob");
```

LinkedHashMap

- Like Hashmap but with sorted keys

Key	Value
Nicklas	20436262
Karsten	20202020
Evander	29392291
…	…
N	N

The Set

The **Set** represents unordered groups of *unique* items, like mathematical sets described in Appendix III. They're used when the order of items you need to store is meaningless, or if you must ensure no items in the group occurs more than once. The common Set operations are:

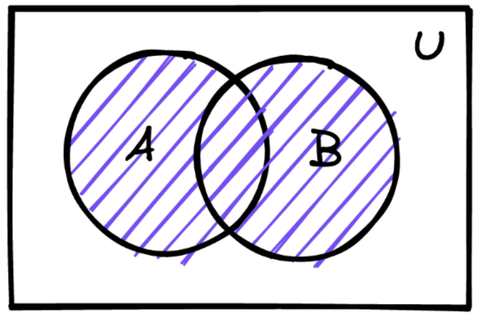
- **`add(e)`**: add an item to the set or produce an error if the item is already in the set,
- **`list()`**: list the items in the set,
- **`delete(e)`**: remove an item from the set.

A set only has unique values

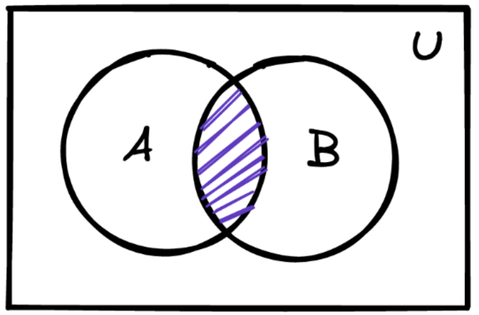
Set Operations in Kotlin

Implementation

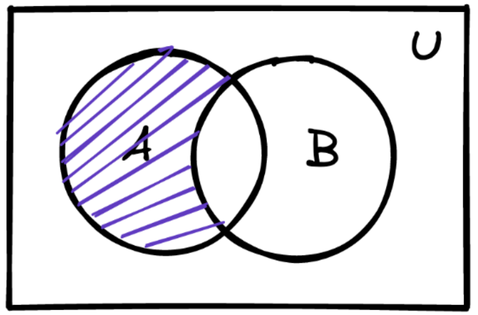
- Set operations will always return a set
- Sets only contains unique values
- Set operations are very fast
- Sets are unordered
- Sets cannot be sorted
- Sets are not very practical for storage or organisation - they are fast and practical for set operations



union



intersection



difference

So which to choose?

- Mutable vs immutable
  - Comes down to if you need to change the elements or values in your data structure
- List vs MutableList
- Set vs MutableSet
- Map vs MutableMap

So which to choose?

List - multiple elements

- ArrayList
  - Pros: Quick access of elements. Efficient appending.
  - Cons: Costly insertion and deletion. Resizing overhead. Allocates fixed memory
- LinkedList
  - Pros: Fast insertion and deletions at beginning or end. Dynamic size. If you have reference for node insertion in middle is efficient aswell
  - Slow for random access

So which to choose?

Map - Key value

- HashMap
  - Pros: Fast access for put, get and remove
  - Cons: No order guaranteed of the keys.
- LinkedHashMap
  - Pros: Maintains order of the keys. Fast access
  - Cons: memory overhead for maintaining linked list

So which to choose?

Set - Unique values

- HashSet
  - Pros: Fast access for add, remove and contains
  - Cons: No order guaranteed of elements
- LinkedHashSet
  - Pros: Maintains order of elements. Fast access
  - Cons: memory overhead for maintaining linked list

BUT WHY??

- Why not just make everything an ArrayList?

Comparable Interface efter pausen

Comparable interface

Imposes natural order - Ordinal data

`public interface` **`Comparable<T>`**

This interface imposes a total ordering on the objects of each class that implements it. This ordering is referred to as the class's *natural ordering*, and the class's `compareTo` method is referred to as its *natural comparison method*.

```
public interface Comparable<in T> {
    public operator fun compareTo(other: T): Int
}
```

-

0

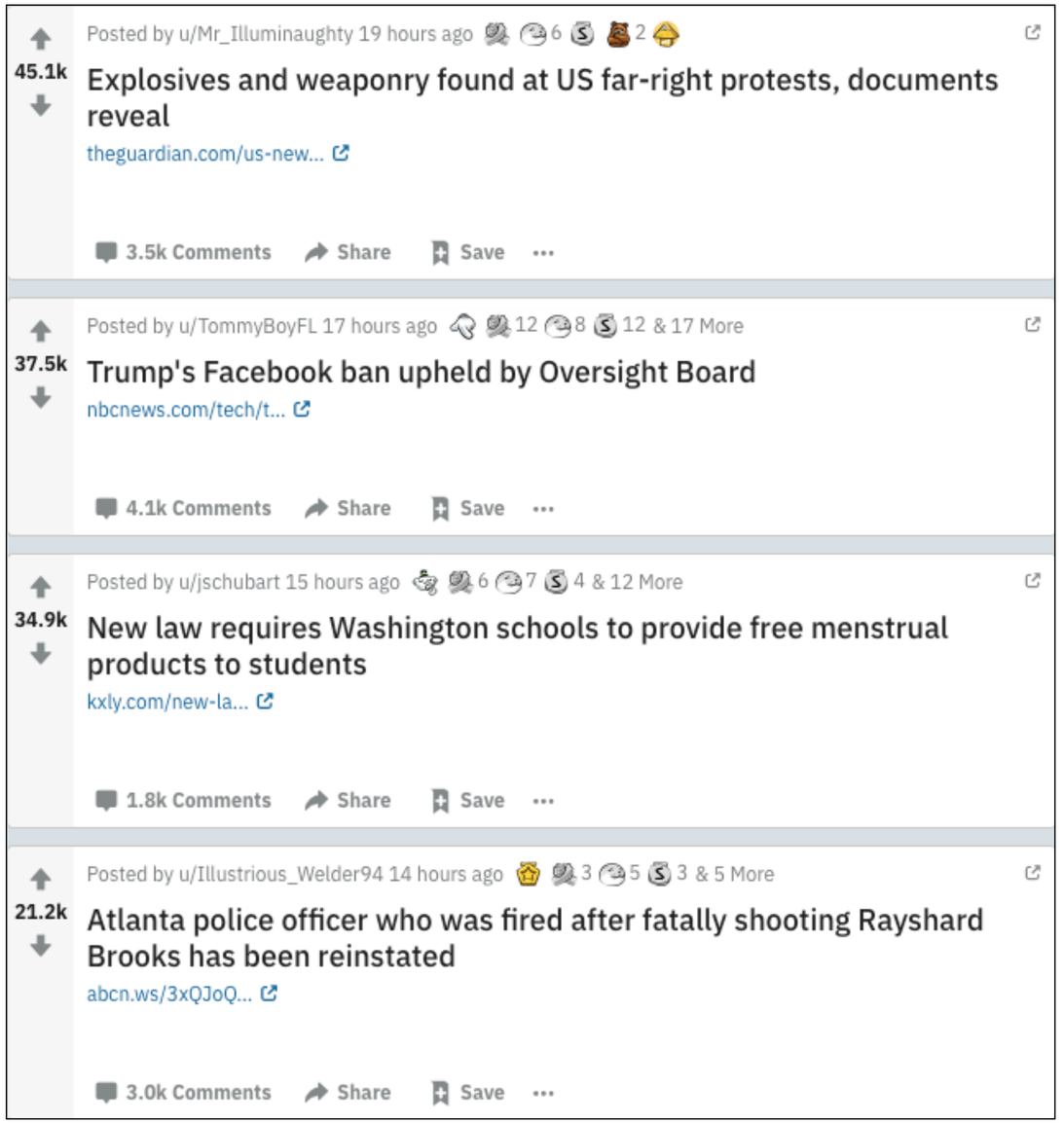
+

Other is bigger

Equal

This is bigger

Comparable analogy: RedditPost



Example: Implementing comparable interface

Exercises Comparable