Kotlin Collections

Applikationsudvikling: CS101

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

Agenda

ADT's & Kotlin Collections Framework

- Data Structures: A Primer
- Array & ArrayList
- ADT's: Set, List, Map
- Implementations: HashSet, ArrayList, HashMap
- Kt Collections Framework: Comparable

Abstract Programming Interface

Web API

API

More than network calls

https://kotlinlang.org/api/latest/jvm/stdlib/

API reference

- Standard library (stdlib) ↗
- Test library (kotlin.test) ↗
- Coroutines (kotlinx.coroutines) ↗
- Serialization (kotlinx.serialization) ↗
- Date and time (kotlinx-datetime) ↗
- JVM Metadata (kotlinx-metadata-jvm) ↗
- Ktor ↗

Android Developers > Develop > Reference

Android API reference

Start building your Android app with the Android Platform APIs. They are available in Kotlin and Java.

https://developer.android.com/reference

Data type | Data Structure

```
val names : ArrayList<String> = ArrayList<String>();
```

Data type

```
val name: String = "George";
val anotherName: String = String(StringBuilder("George"));
```

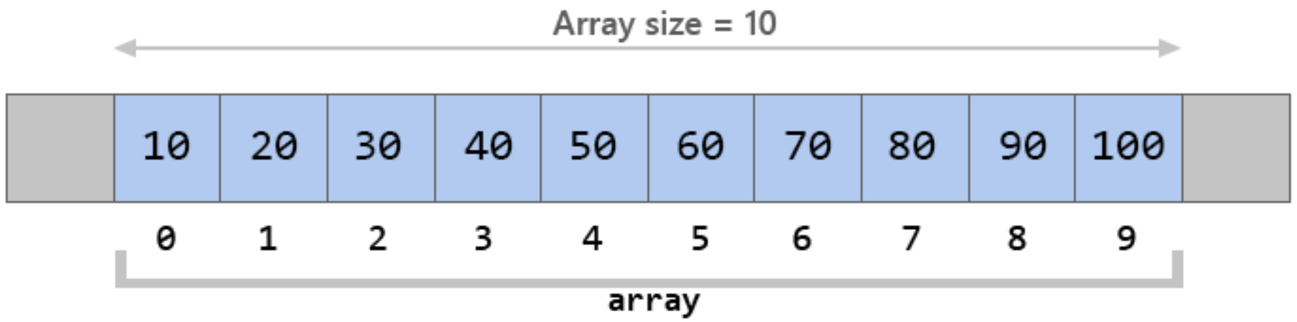
Data type

```
val age: Int = 3;
```

An Array

```
val ints: IntArray = IntArray(10);

val ints: Array<Int> = Array<Int>(10){0};
```



Fixed size

Example Working with indices

Abstract Data Types

Data Structures

- ArrayList
- HashMap
- HashSet

Data + Data structure

What are the downsides of an array

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:

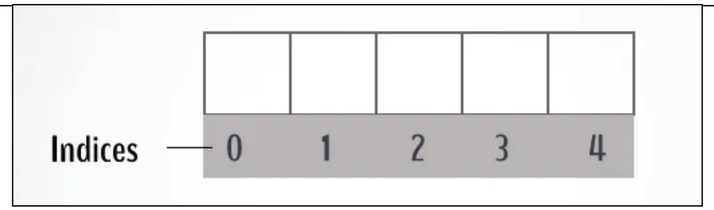
- **`insert(n, e)`**: insert the item `e` at position `n`,
- **`remove(n)`**: remove the item at position `n`,
- **`get(n)`**: get the item at position `n`,
- **`sort()`**: sort the items in the list,
- **`slice(start, end)`**: return a sub-list slice starting at the position `start` up until the position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

The List type exists in:

Same abstract operations

- Java as an ArrayList
- Javascript as an Array
- Python as a List
- C# as ArrayList
- Kotlin as ArrayList
- The List goes on…

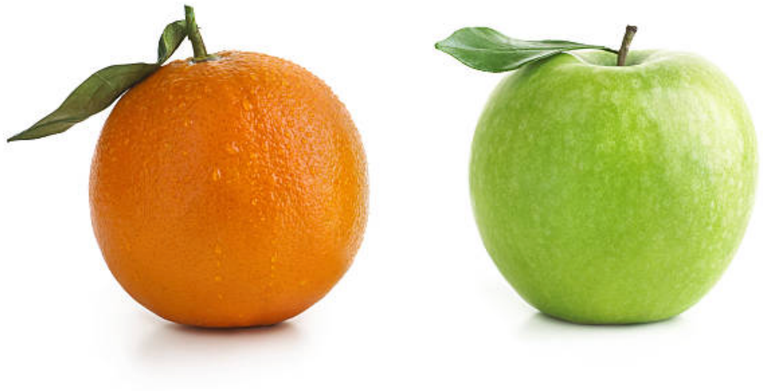
- **`insert(n, e)`**: insert the item **e** at position **n**,
- **`remove(n)`**: remove the item at position **n**,
- **`get(n)`**: get the item at position **n**,
- **`sort()`**: sort the items in the list,
- **`slice(start, end)`**: return a sub-list slice starting at the position **start** up until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

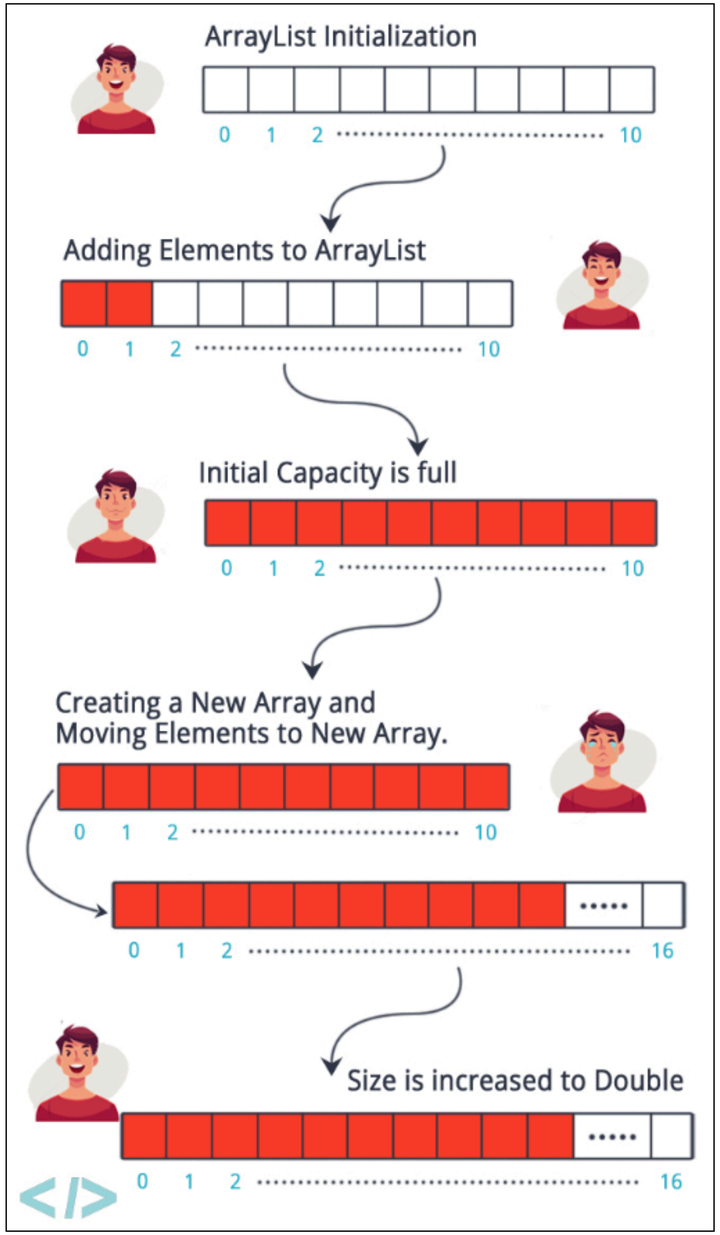


ArrayList

ImplementationADT

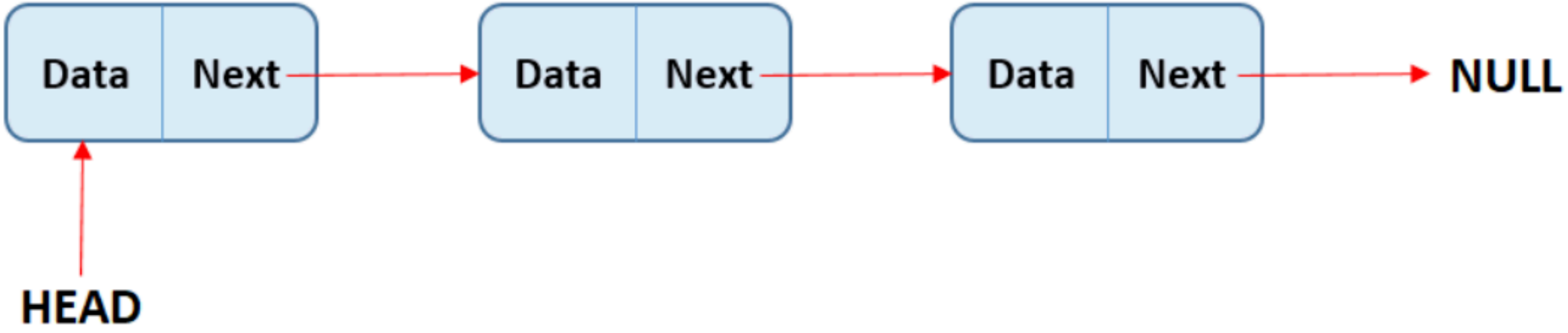
```
private void add(E e, Object[] elementData, int s) {
    if (s == elementData.length)
        elementData = grow();
    elementData[s] = e;
    size = s + 1;
}
```

From ArrayList



LinkedList

ImplementationADT

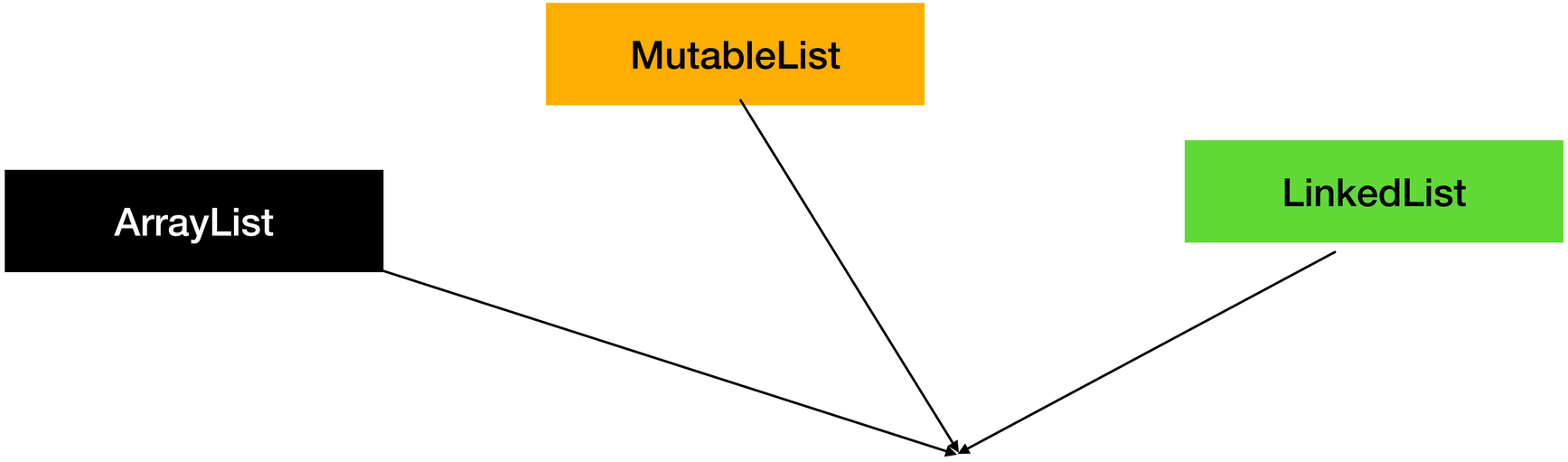


MutableList

LinkedList

ArrayList

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




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

Boys

- Liam
- Noah
- Oliver
- Elijah
- William
- James
- Benjamin
- Lucas
- Henry
- Alexander

Top 10 popular baby names: List of strings

List of articles

```
class Article (
    val headLine: String,
    val author: String,
    val category: String,
    private val isFrontPage: Boolean
){
}
```

```
val articles: ArrayList<Article> = ArrayList<Article>();
```

```
val names: List<String> = ArrayList<String>();
```

Abstract Data Types

Data Structures

- ArrayList
- HashMap
- HashSet

Implementation | Abstract Type

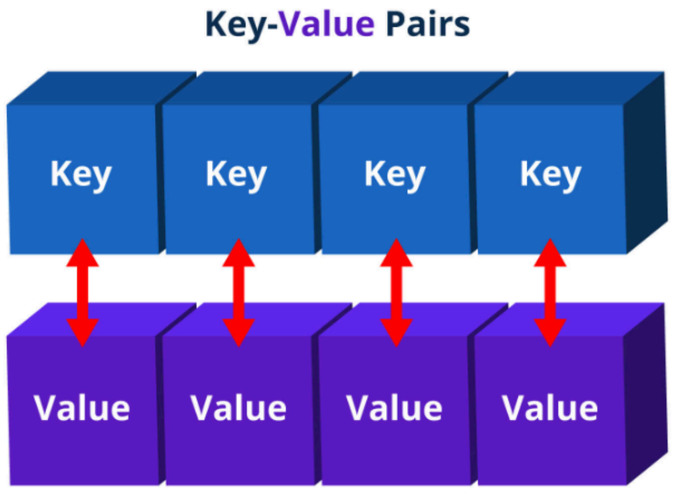
Data + Data structure

What is hashing

Abstract Data Types

Map Type

Javascript object-like



HashMap

ImplementationADT

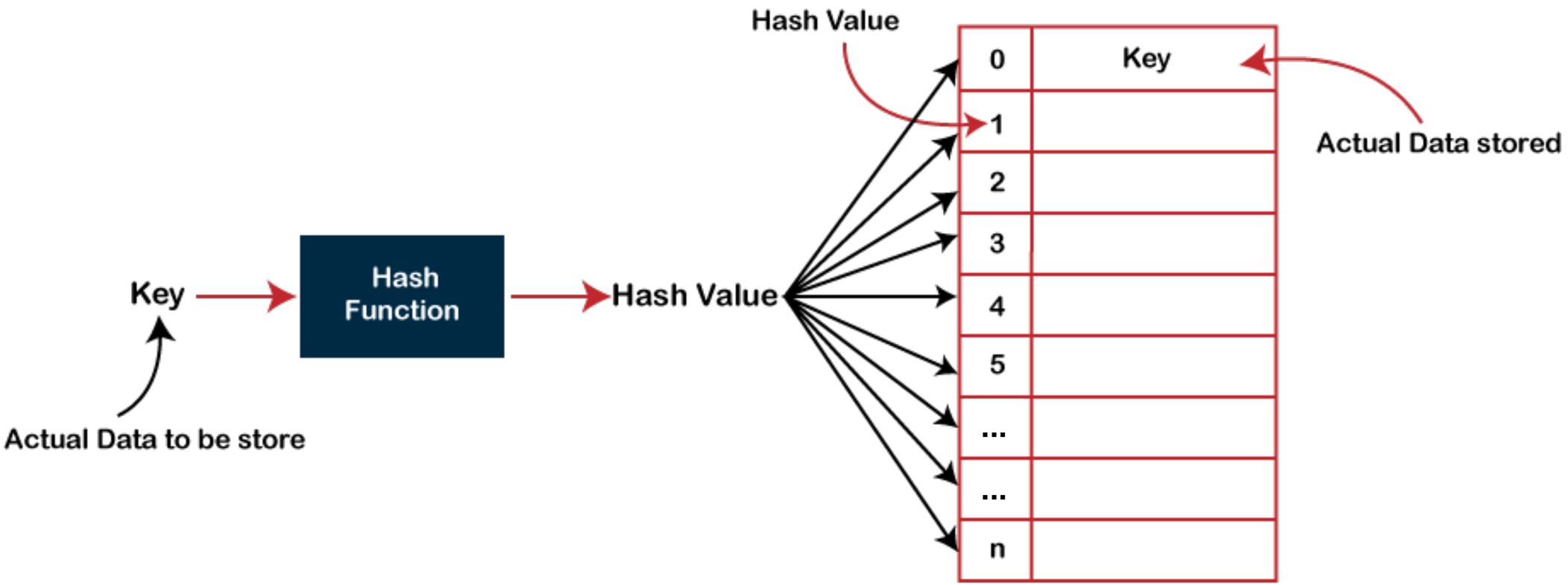
- **`set(key, value)`**: add a key-value mapping,
- **`delete(key)`**: remove `key` and its associated value,
- **`get(key)`**: retrieve the value that was associated to `key`.

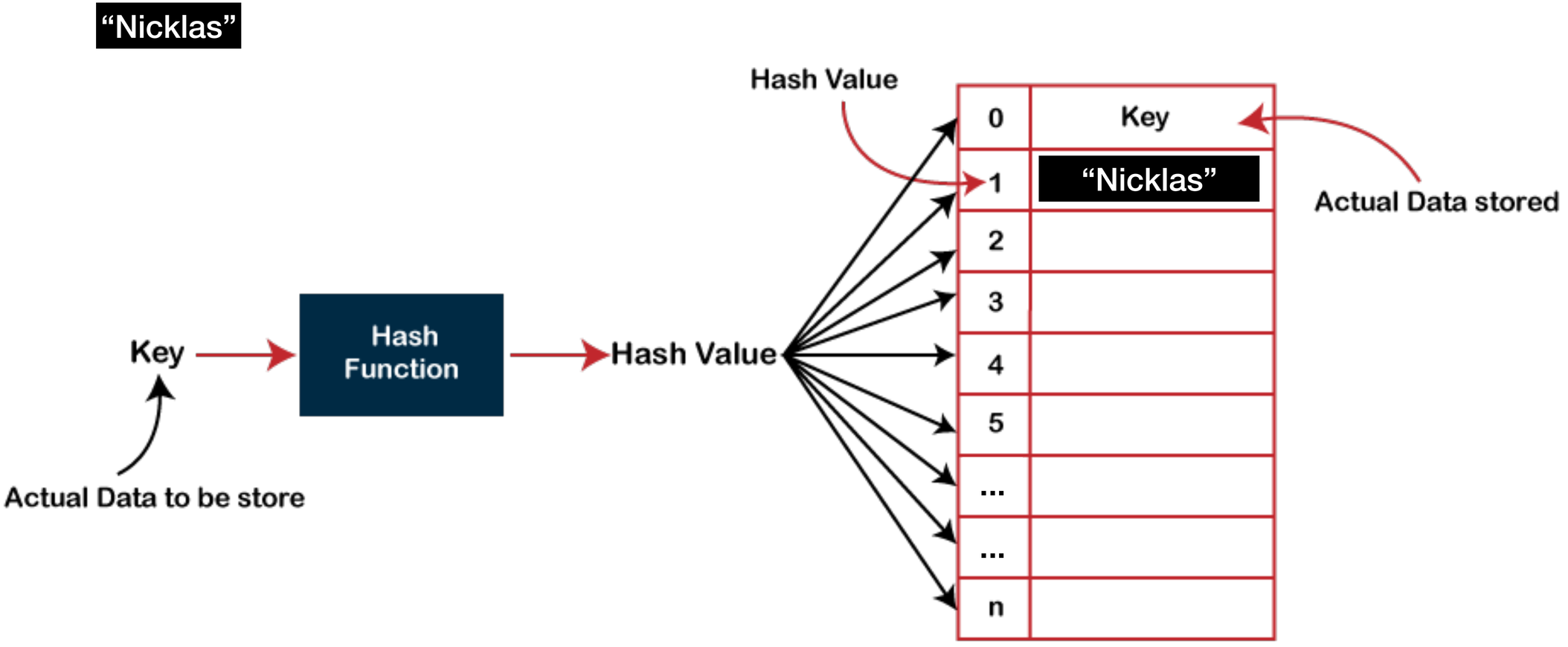
```
//Data looks like this:
//Nicklas , 20436262
//Jakob , 88009872
HashMap<String,Integer> phonebook = new HashMap<~>();

phonebook.get("Nicklas");
//Returns 20436262
```

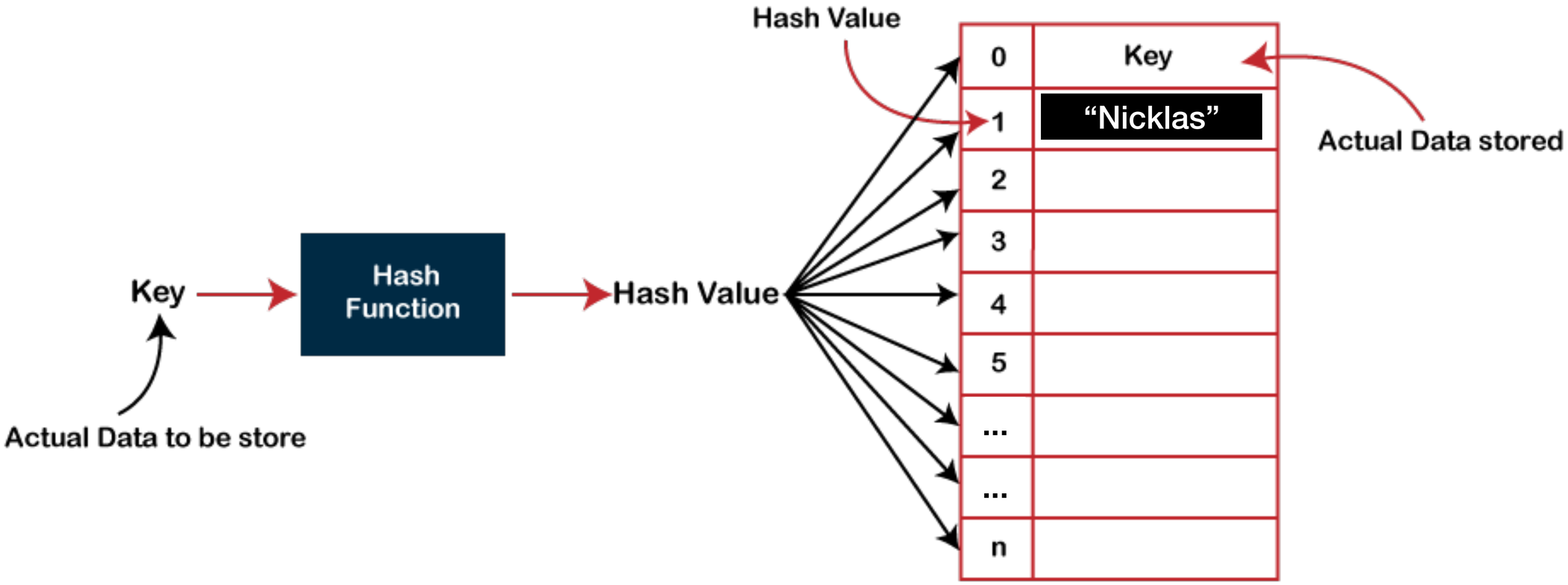
Key	**Value**
Nicklas	20436262
Karsten	20202020
Evander	29392291
...	...
N	N

“Nicklas”

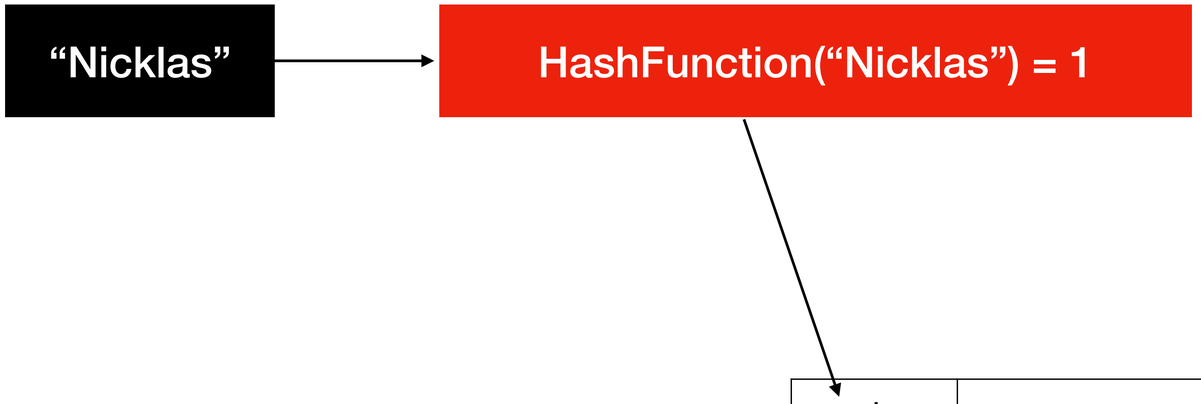

Hash Value
0
Key
1
2
3
4
5
...
...
n
Actual Data stored
Key
Hash Function
Hash Value
Actual Data to be store


"Nicklas"
Hash Value
0
Key
1
"Nicklas"
2
3
4
5
...
...
n
Actual Data stored
Key
Hash Function
Hash Value
Actual Data to be store

Hash Value	Key	Value
1	Nicklas	20436262
3	Karsten	20202020
12	Evander	29392291
	…	…
	N	N



```
phonebook.get("Nicklas")
```



1	Nicklas	20436262
3	Karsten	20202020
12	Evander	29392291
	...	...
	N	N

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");
```

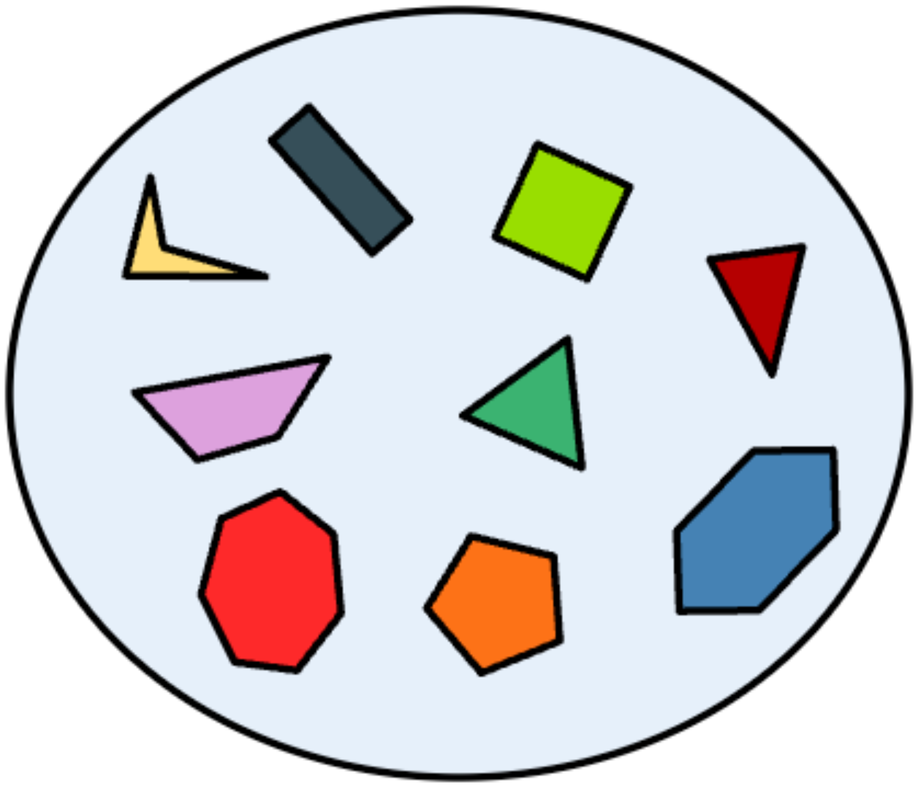
The Set

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.

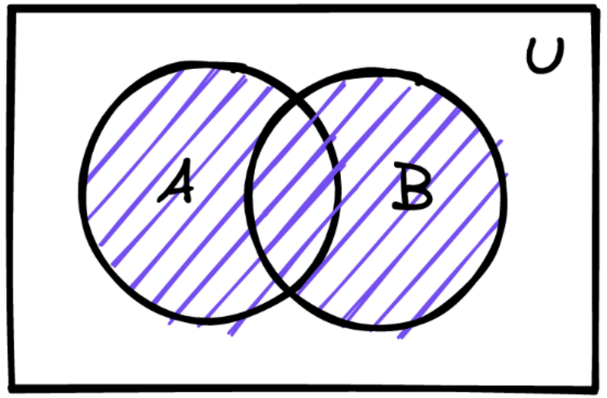
A Mathematical Set

A collection of different things

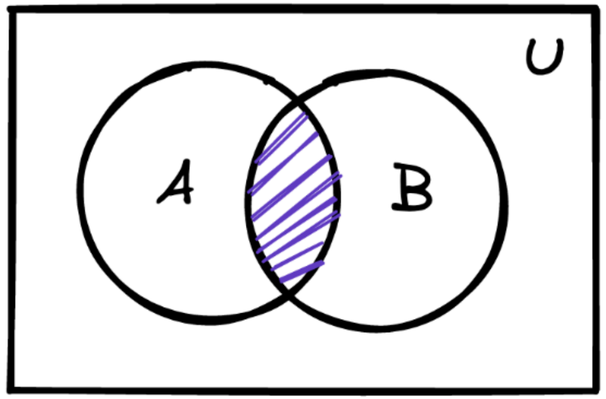


https://en.wikipedia.org/wiki/Set_%28mathematics%29

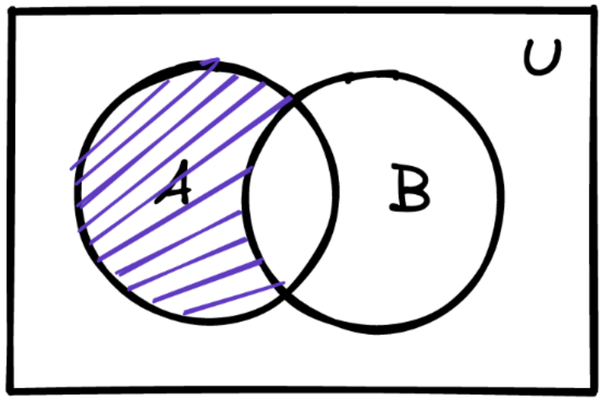
A set only has unique values



union



intersection



difference

https://ivov.dev/notes/typescript-and-set-theory

MutableSet / HashSet Implementation

Unique Elements

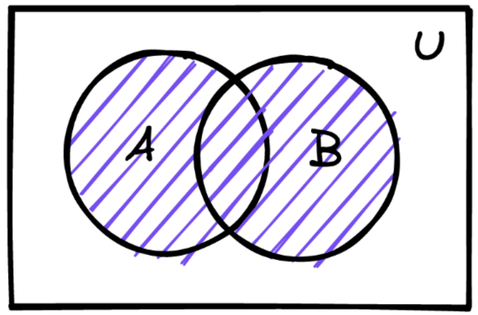
- Value store
- Inconsistent (May change order over time)
- Uses hashing for value generation
  - Create: add(value)
  - ~~Read:~~ first( ) / toList( )
  - ~~Update:~~
  - Remove: remove( )

```
const list = [
  {
    name: 'Michael Scott',
    company: 'Dunder Mufflin',
    designation: 'Regional Manager',
    show: 'The Office'
  },
  {
    name: 'Barney Stinson',
    company: 'Golaith National Bank',
    designation: 'Please',
    show: 'How I met your mother'
  },
  {
    name: 'Jake Peralta',
    company: 'NYPD',
    designation: 'Detective',
    show: 'Brooklyn 99'
  },
]
```

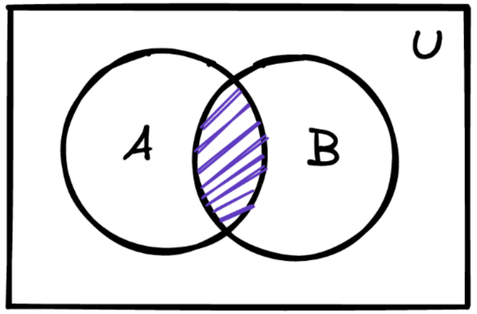
Set Operations in Kotlin

Implementation

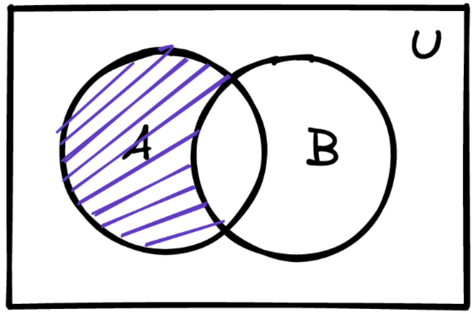
- Set operations will always return a set
- Sets only contains unique values
- Set operations are very fast
- Sets are unordered
- Sets cannot be sorted
- Sets are not very practical for storage or organisation - they are fast and practical for set operations



union



intersection



difference

Example Set

Reading a file in Kotlin

Exercises Arrays, HashMap & Set

Comparable Interface

Comparable interface

Imposes natural order - Ordinal data

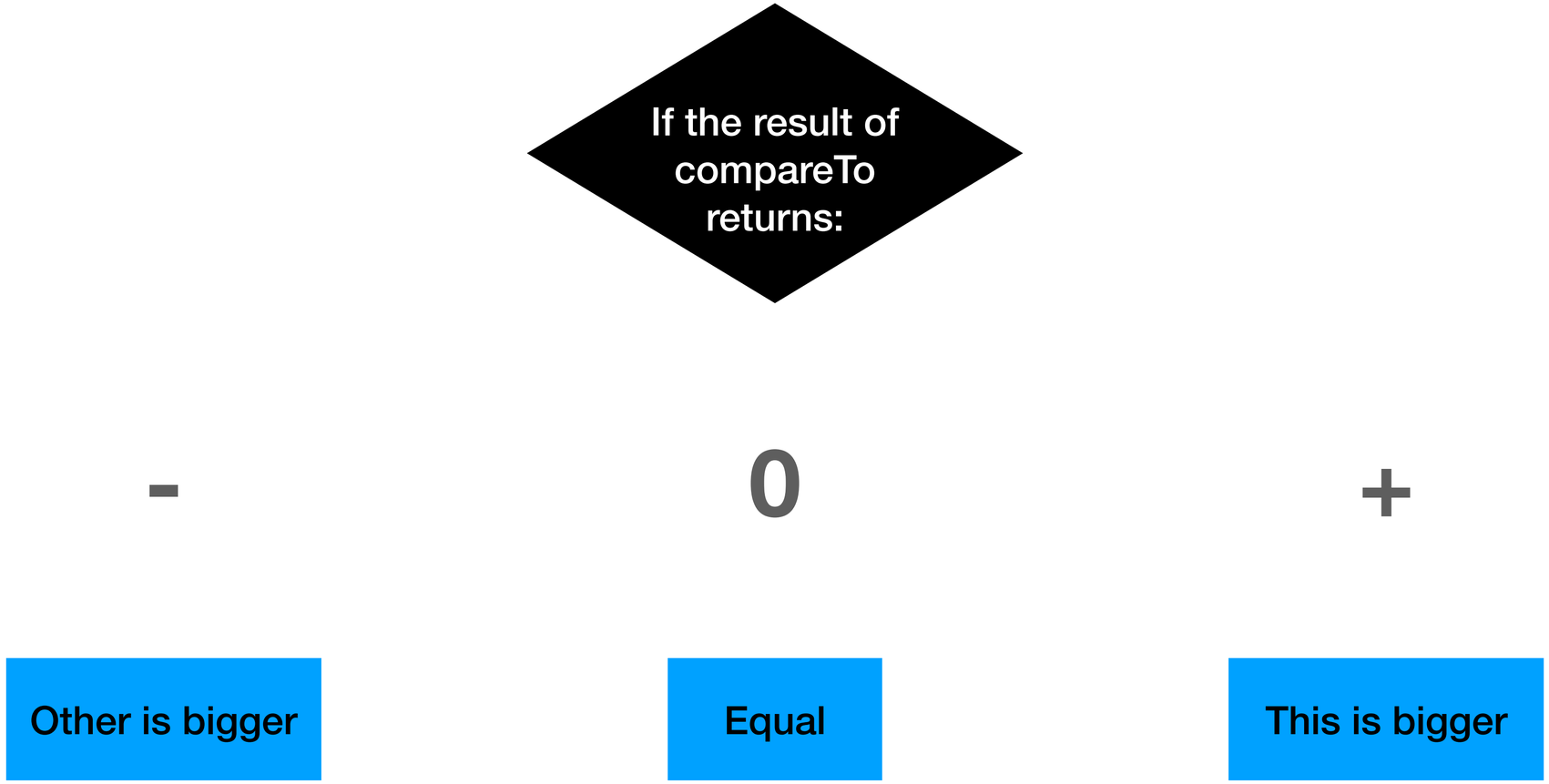
```
public interface Comparable<T>
```

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*.

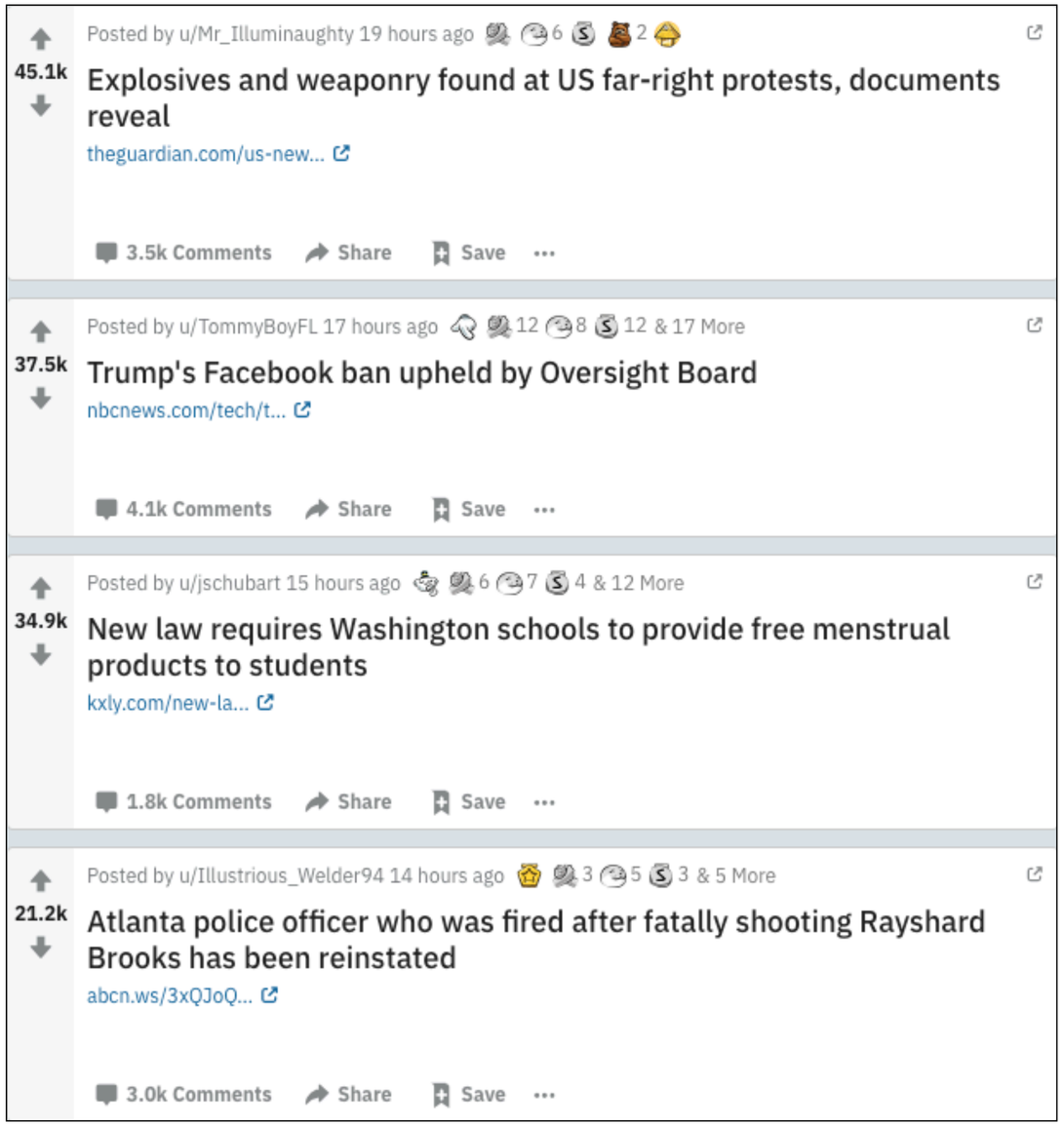
```
public interface Comparable<in T> {
    public operator fun compareTo(other: T): Int
}
```

“A non programming analogy for an interface is a professional certification. It’s possible for a person to become certified as a teacher, nurse, accountant, or doctor. To do this, the person must demonstrate certain abilities required of members of those professions”

Reges, Stepp, Building Java Programs - Chapter 9 P. 653


If the result of compareTo returns:
-
0
+
Other is bigger
Equal
This is bigger

Comparable analogy: RedditPost



Example: Implementing comparable interface

Exercises Comparable