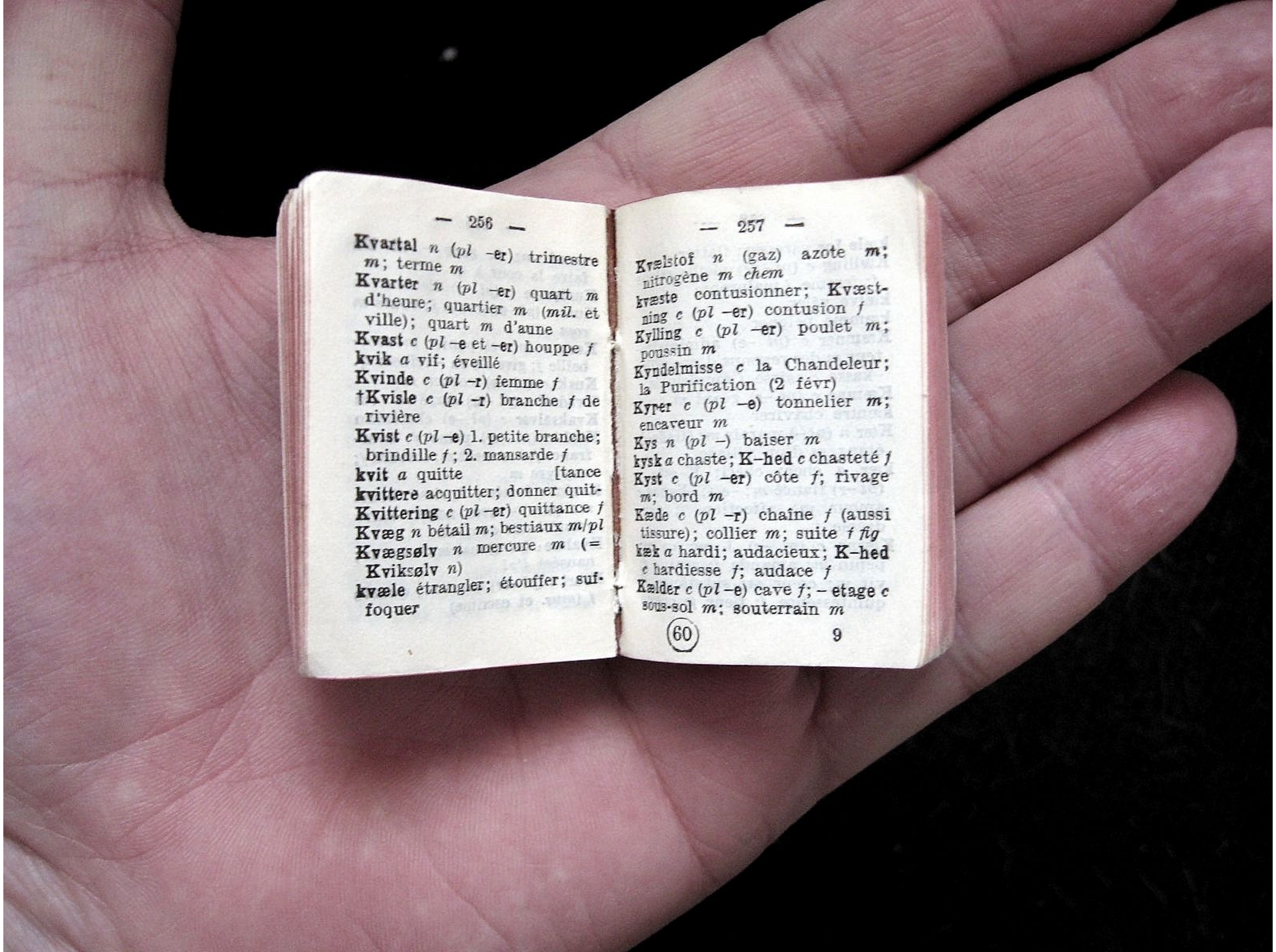
Java SE 17 Collections

Storing Key / Value Pairs: Maps

Richard Warburton

Java champion, Author and Programmer

@richardwarburto www.monotonic.co.uk



Kvartal *n* (*pl* -**er**) trimestre *m*; terme *m*

Kvarter *n* (*pl* -**er**) quart *m* d'heure; quartier *m* (*mil.* et ville); quart *m* d'aune

Kvast *c* (*pl* -**e** et -**er**) houppe *f*

kvik *a* vif; éveillé

Kvinde *c* (*pl* -**r**) femme *f*

†**Kvisle** *c* (*pl* -**r**) branche *f* de rivière

Kvist *c* (*pl* -**e**) 1. petite branche; brindille *f*; 2. mansarde *f*

kvit *a* quitte

kvittere acquitter; donner quittance

Kvittering *c* (*pl* -**er**) quittance *f*

Kvæg *n* bétail *m*; bestiaux *m/pl*

Kvægsølv *n* mercure *m* (= **Kviksølv** *n*)

kvæle étrangler; étouffer; suffoquer

Kvælstof *n* (gaz) azote *m*; nitrogène *m* *chem*

kvæste contusionner; **Kvæstning** *c* (*pl* -**er**) contusion *f*

Kylling *c* (*pl* -**er**) poulet *m*; poussin *m*

Kyndelmisse *c* la Chandeleur; la Purification (2 févr)

Kyper *c* (*pl* -**e**) tonnelier *m*; encaveur *m*

Kys *n* (*pl* –) baiser *m*

kysk *a* chaste; **K-hed** *c* chasteté *f*

Kyst *c* (*pl* -**er**) côte *f*; rivage *m*; bord *m*

Kæde *c* (*pl* -**r**) chaîne *f* (aussi tissure); collier *m*; suite *f* *fig*

kæk *a* hardi; audacieux; **K-hed** *c* hardiesse *f*; audace *f*

Kælder *c* (*pl* -**e**) cave *f*; – **etage** *c* sous-sol *m*; souterrain *m*

Key -> Value

Outline

Why use a map?

Views over maps

Advanced Operations

Implementations

Correctly using HashMap

Why Use a Map?

Map API

```
V put(K key, V value)

void putAll(Map<? extends K, ? extends V> values)
```

Adding and Replacing

put for a single value, putAll for another Map

Null keys and values are implementation specific

```
V

get(Object key)


boolean

containsKey(Object key)


boolean

containsValue(Object value)
```

- Looking up elements

- Separate contain methods for key and value

- Objects allow more flexible generic contracts

```
V remove(Object key)

void clear()
```

Removing

Querying Size

```
int size()

boolean isEmpty()
```

```
Map.Entry<String, Integer> entry =
  Map.entry(“Richard”, 38);


Map<String, Integer> personToAge =
  Map.of(“Richard”, 38);


personToAge = Map.ofEntries(
  Map.entry(“Richard”, 38));


Map<String, Integer> copy =
  Map.copyOf(personToAge);
```

◀ **Immutable Map Factories**

◀ **Individual key / value pairs**

◀ **Up to 10 value specific overload Factories**

◀ **For > 10 varargs factory takes entry objects**

◀ **Immutable Copies of an existing Map**

Collection and Map

Map is the only collections that don't extend or implement the Collection interface

Views over Maps

Advanced Operations

Altering and Removing

replace(key, value)

Update a single value

replaceAll(BiFunction<K, V, V>)

Replace elements using a function

remove(key, value)

Remove a key only if it has a value

Updating

- getOrDefault
- putIfAbsent
- compute
- computeIfAbsent
- computeIfPresent
- merge

forEach

Convenient callback based iteration

Advanced Operations Demo

Implementations

General Purpose Implementations

HashMap

Good general purpose implementation

TreeMap

Defines sort order and adds functionality

HashMap

Good general purpose implementation

Uses the .hashCode() method

Maintains an array of buckets

rehash(hash) % bucket_count

Buckets are linked lists to accommodate collisions

Buckets can be trees

The number of buckets increases with more elements

TreeMap

Comparator

Key elements have a sort order

Red / Black Tree

A Balanced binary tree

Navigable & Sorted

Provides functionality that HashMap doesn’t

Performance Comparison

	put	get	containsKey	next
HashMap	O(N) , Ω(1)	O(N), Ω(1)	O(N), Ω(1)	O(Capacity/N)
TreeMap	O(log(N))	O(log(N))	O(log(N))	O(log(N))

Special Purpose Implementations

LinkedHashMap

First Special Purpose Map implementation

When

Useful for implementing Size based caches

Maintains Order

Either Insertion or Access

removeEldestEntry

Subclass and Override method in order to control cache

IdentityHashMap

IdentityHashMap	HashMap
`System.identityHashCode()`	`obj.equals() & obj.hashCode()`
Use for Serialisation or Graphs	**Use normally**
Faster & Lower Memory	**Avoids coupling Map to Keys**
Low Collission Likelihood	
Violates Map Contract	

WeakHashMap

Useful for a Memory Bounded Cache

Keys have weak references, can be collected if nothing else references them

Entries can be removed after the key is collected

EnumMap

Keys are Enums

Faster and Low memory usage

Bitset Implementation

Only a single long if < 64 elements

Correctly Using HashMap

Summary

Maps associate keys and values

2 general implementations

4 special purpose implementations

API still improving in Java 17 and beyond

Whatever you need, Java has you covered

Up Next:
Introduction to Java Streams