Unit 10: Maps

Unit Objectives. Here you will:

- recap the characteristics of maps
- learn about when to use maps
- explore some concrete implementations of maps in the JCF

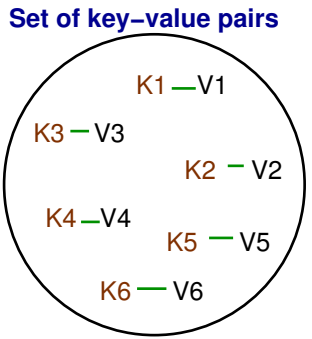
The Map ADT

- . . . maps **key**s to **value**s.
  - ➠ Each key is mapped with *one* value.

Unit 3

The Map ADT

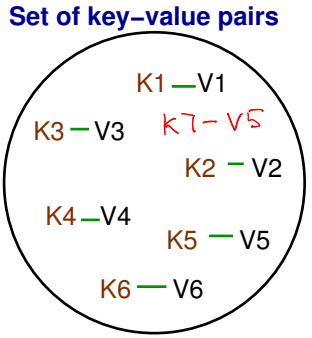
- . . . maps **keys** to **values**.
  - ➠ Each key is mapped with *one* value.



- Also known as **dictionary**, **associative array** and **lookup table**.

The Map ADT

- …maps **keys** to **values**.
  - ➠ Each key is mapped with *one* value.

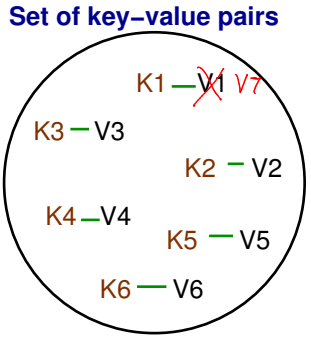


- Also known as **dictionary**, **associative array** and **lookup table**.

- Typical operations includes:
  - ◇ **Add**: Associate a new value with a new key.

The Map ADT

- . . . maps **keys** to **values**.
  - ➠ Each key is mapped with *one* value.

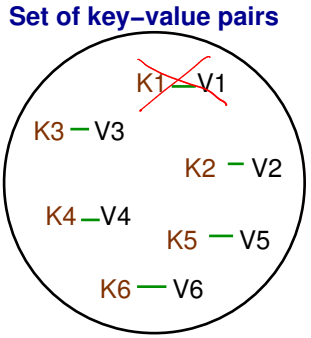


- Also known as **dictionary**, **associative array** and **lookup table**.

- Typical operations includes:
  - ◇ **Add**: Associate a new value with a new key.
  - ◇ **Reassign**: Associate an existing key in the map with a new value.

The Map ADT

- . . . maps **key**s to **value**s.
  - ➠ Each key is mapped with *one* value.

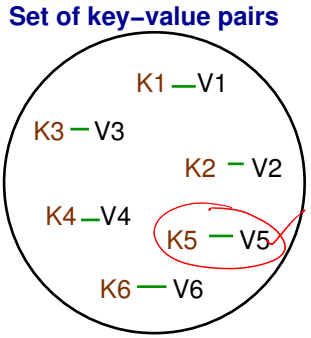


- Also known as **dictionary**, **associative array** and **lookup table**.

- Typical operations includes:
  - ◇ **Add**: Associate a new value with a new key.
  - ◇ **Reassign**: Associate an existing key in the map with a new value.
  - ◇ **Remove**: Remove a key with its associated value from the key set in the map.

The Map ADT

- . . . maps **keys** to **values**.
  - ➠ Each key is mapped with *one* value.



- Also known as **dictionary**, **associative array** and **lookup table**.

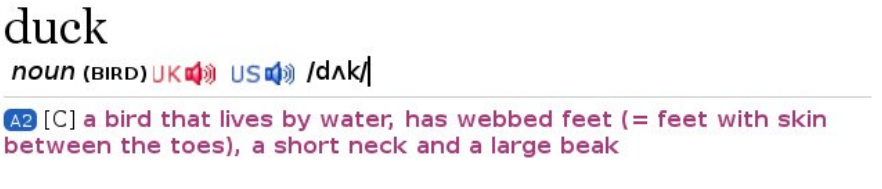
- Typical operations includes:
  - ◇ **Add**: Associate a new value with a new key.
  - ◇ **Reassign**: Associate an existing key in the map with a new value.
  - ◇ **Remove**: Remove a key with its associated value from the key set in the map.
  - ◇ **Lookup**: Find the value (if any) that is associated with a key.

When would maps be useful?

- to model the contents of a *dictionary*
- to model the contents of a *phone book*
- to provide various *indexing* for a table of database records

Modelling dictionary contents using Map

Consider the data in a dictionary entry, e.g.:


duck

noun (BIRD) UK US /dʌk/

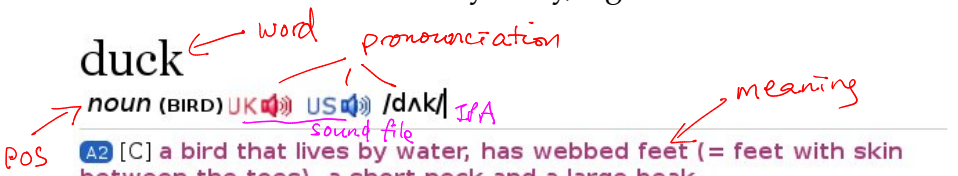
A2 [C] a bird that lives by water, has webbed feet (= feet with skin between the toes), a short neck and a large beak

Source: http://dictionary.cambridge.org/dictionary/british/duck_1?q=duck

How to model the dictionary entry in Java?

Modelling dictionary contents using Map

Consider the data in a dictionary entry, e.g.:



Source: http://dictionary.cambridge.org/dictionary/british/duck_1?q=duck

How to model the dictionary entry in Java?

Define a Java class named `Entry`.

Modelling dictionary contents using Map

What kind of **fields** should class `Entry` have?

✔

Which **field** should be used as the **key**?

Modelling dictionary contents using Map

What kind of **fields** should class `Entry` have?

```
1  public class Entry {
2      private String word;
3      private POS pos;    // part-of-speech, e.g.  noun
4      private Pronunciation pronunciation;
5      private String meaning;
6
7      // other details omitted
8  }
9
10 enum POS {
11     PRONOUN, NOUN, VERB, ADJECTIVE, ADVERB, PREPOSITION
12 }
```

Which **field** should be used as the **key**?

Modelling dictionary contents using Map

What kind of **fields** should class `Entry` have?

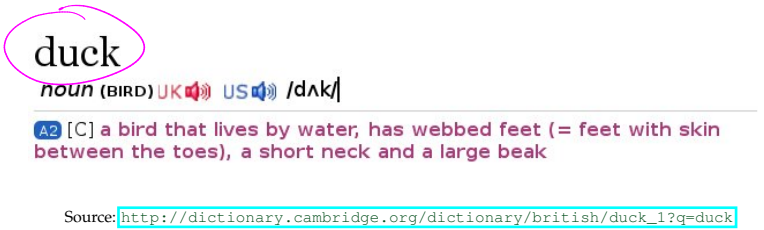
```
1  public class Entry {
2      private String word;
3      private POS pos;    // part-of-speech, e.g.  noun
4      private Pronunciation pronunciation;
5      private String meaning;
6
7      // other details omitted
8  }
9
10 enum POS {
11     PRONOUN, NOUN, VERB, ADJECTIVE, ADVERB, PREPOSITION
12 }
```

Which **field** should be used as the **key**?

`word`

Modelling dictionary contents using Map

- How to model dictionary contents such as the below to facilitate efficient dictionary lookup?



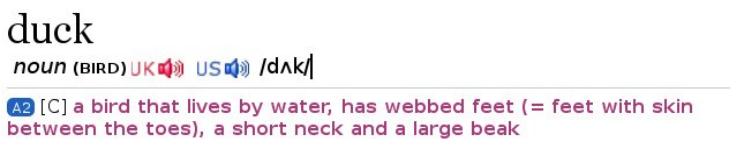
- What would be the **key** and **value**? Entry

- How to model a dictionary with $5,000$ entries in `Java`?

✔

Modelling dictionary contents using Map

- How to model dictionary contents such as the below to facilitate efficient dictionary lookup?



Source: http://dictionary.cambridge.org/dictionary/british/duck_1?q=duck

- What would be the **key** and **value**?

K = word to be looked up, V = dictionary entry

- How to model a dictionary with $5,000$ entries in `Java`?

✔

Modelling dictionary contents using Map

- How to model dictionary contents such as the below to facilitate efficient dictionary lookup?

load factor .75

duck

noun (BIRD) UK US /dʌk/

A2 [C] a bird that lives by water, has webbed feet (= feet with skin between the toes), a short neck and a large beak

Source: http://dictionary.cambridge.org/dictionary/british/duck_1?q=duck

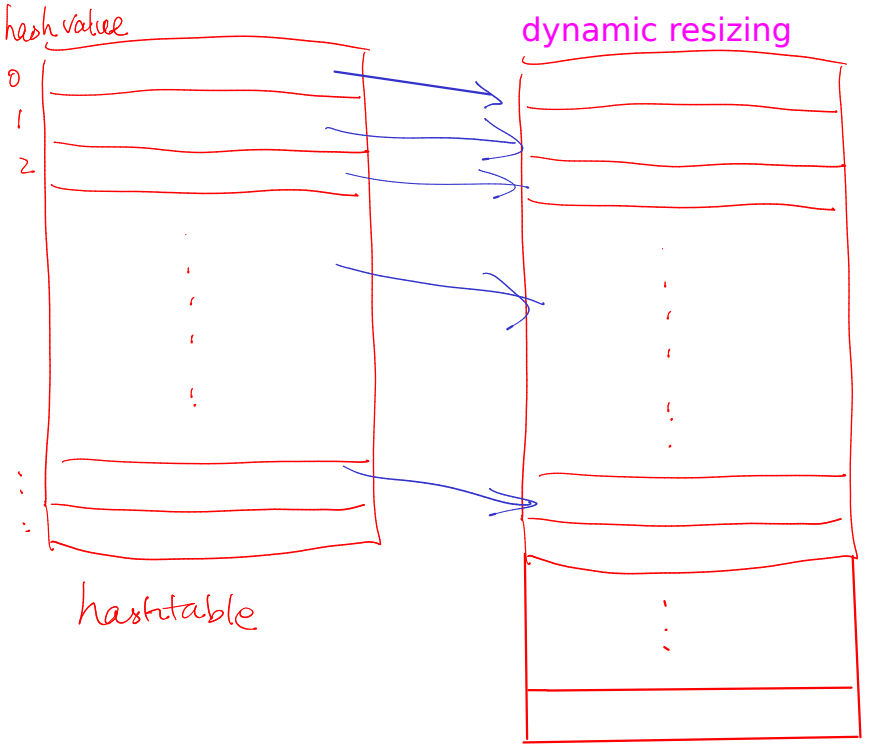
- What would be the **key** and **value**?

K = word to be looked up, V = dictionary entry

- How to model a dictionary with 5,000 entries in Java? 5000*1.5

```
Map<String, Entry> dictionary = new HashMap<>(7500);
```

O(1)


hash value
0
1
2
hashtable
dynamic resizing

Modelling contents of a phone book

Consider the data in a phone book entry, e.g.:

Aston University Nursery

The Aston Triangle, B4 7ET

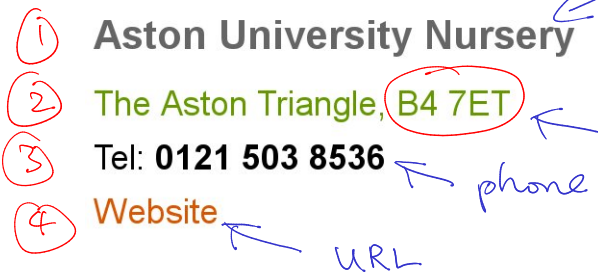
Tel: **0121 503 8536**

Website

How to model the phone book entry in Java? ✔

Modelling contents of a phone book

Consider the data in a phone book entry, e.g.:



How to model the phone book entry in Java? ✔

Define a Java class named `Contact`.

Modelling contents of a phone book

What **fields** should class `Contact` have?

Which **field** should be used as the **key**?

✔

Modelling contents of a phone book

What **fields** should class `Contact` have?

```
1  public class Contact {
2
3      private String name;
4      private Address address;
5      private String phone;
6      private String url;
7
8      // other details omitted
9
10 }
```

Which **field** should be used as the **key**?

Modelling contents of a phone book

What **fields** should class `Contact` have?

```
1  public class Contact {
2
3      private String name;
4      private Address address;
5      private String phone;
6      private String url;
7
8      // other details omitted
9
10 }
```

Which **field** should be used as the **key**?

✔

`name`

Modelling contents of a phone book

- How to model a phone book with contact information such as the below to facilitate efficient lookup?

Aston University Nursery

The Aston Triangle, B4 7ET

Tel: **0121 503 8536**

Website

- What would be the **key** and **value**?

- How to model the data in a phone book in `Java`?

Modelling contents of a phone book

- How to model a phone book with contact information such as the below to facilitate efficient lookup?

Aston University Nursery

The Aston Triangle, B4 7ET

Tel: **0121 503 8536**

Website

- What would be the **key** and **value**?

K = name of person/organisation, V = correspondence detail

- How to model the data in a phone book in `Java`?

Modelling contents of a phone book

- How to model a phone book with contact information such as the below to facilitate efficient lookup?

Aston University Nursery

The Aston Triangle, B4 7ET

Tel: **0121 503 8536**

Website

- What would be the **key** and **value**?

K = name of person/organisation, V = correspondence detail

- How to model the data in a phone book in `Java`?

hash table implementation

```
Map<String, Contact> phoneBook = new HashMap<>();
```

Indexing database records

Consider the following Supplier table:

Supplier ID

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

Source: http://c2.com/cgi/wiki?SupplierPartsDatabase

How to model a supplier record in Java? ✔

Consider the following Supplier table:

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

Source: http://c2.com/cgi/wiki?SupplierPartsDatabase

How to model a supplier record in Java? ✔

Define a Java class named `Supplier`. with 4 fields

Indexing database records

What **fields** should class `Supplier` have?

value

```
1  public class Supplier {
2
3      private String number;
4      private String name;
5      private int status;
6      private String city;
7
8      // other details omitted
9
10 }
```

✔

Key ?

Indexing database records

With the following supplier table:

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

Source: http://c2.com/cgi/wiki?SupplierPartsDatabase

How to model the records in a supplier table such as the below to facilitate efficient *efficient* search over the supplier's:

1. name (`SNAME`), or
2. location (`CITY`)?

✔

➡ A means to facilitate search is by indexing data.

Indexing database records

With the following supplier table:

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

Source: http://c2.com/cgi/wiki?SupplierPartsDatabase

How to model the records in a supplier table such as the below to facilitate efficient *efficient* search over the supplier's:

1. name (`SNAME`), or
2. location (`CITY`)?

Use `HashMap` to index the data in the supplier table.

➡ A means to facilitate search is by indexing data.

Indexing database records

We will need *two* **maps**...

- What would be the **key** and **value**?

- How to model the required indexing in `Java`?

Indexing database records

We will need *two* **maps**...

- What would be the **key** and **value**?
  1. *K = data in* `SNAME`, *V = supplier record* Map 1
  2. *K = data in* `CITY`, *V = supplier records* map 2 Collection
- How to model the required ~~indexing~~ in `Java`?

✔

We will need *two* **maps**...

- What would be the **key** and **value**?
  1. *K = data in* `SNAME`, *V = supplier record*
  2. *K = data in* `CITY`, *V = supplier records*
- How to model the required indexing in `Java`?

```
Map<String, Supplier> suppliersByName =
                                    new HashMap<>();

Map<String, Set<Supplier>> suppliersByCity =
                                    new TreeMap<>();
```

O(1)

Collection

sorted alphabetically by city name

O(log n)

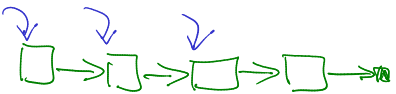
Potential Implementations of `Map`

- Which implementation of **map** is more efficient?
  - ◇ **array**

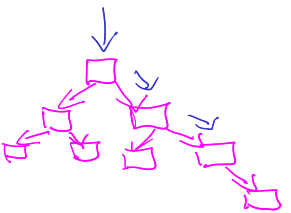
O(n) Key-value

  - ◇ **linear linked structure**

O(n)



  - ◇ **tree**



O(log n)

  - ◇ **hash table** ✔

O(1) hash function

The `Map` interface in JCF

- The `java.util` package includes the interface `Map<K,V>`.
- A *map* is a mapping from a finite set of *keys* to *values*.
- Each key is mapped to a **single** value.
- Given a key, one can look up the corresponding value from a map.

The `Map` interface in JCF

- The `java.util` package includes the interface `Map<K,V>`.
- A *map* is a mapping from a finite set of *keys* to *values*.
- Each key is mapped to a **single** value.
- Given a key, one can look up the corresponding value from a map.
- New key-value pairs can be *added* to a map.
- Existing key-value pairs can be *removed* from the map.

✔

Some Methods in the `Map` interface in JCF

The main methods defined in the `Map` interface are:

- `V get(Object key)`

  ➠ Returns the value to which the specified key is mapped. (**Look up**)

Some Methods in the `Map` interface in JCF

The main methods defined in the `Map` interface are:

- `V get(Object key)`
  - ➠ Returns the value to which the specified key is mapped. (**Look up**)

- `V put(K key, V value)`
  - ➠ Associates the specified value with the specified key (**Reassign**)
  - ➠ If the key is not yet in the map, add a new key-value mapping. (**Add**)

Some Methods in the `Map` interface in JCF

The main methods defined in the `Map` interface are:

- `V get(Object key)`
  - ➠ Returns the value to which the specified key is mapped. (**Look up**)

- `V put(K key, V value)`
  - ➠ Associates the specified value with the specified key (**Reassign**)
  - ➠ If the key is not yet in the map, add a new key-value mapping. (**Add**)

- `V remove(Object key)`
  - ➠ Removes the mapping for a key. (**Remove**) ✔

Concrete Implementations of `Map` in JCF

- The main implementations of `Map<K,V>` are:
  - `HashMap<K,V>` – **hash table** implementation of **map**.
    - ➠ Used when the order of the data in the map is irrelevant, e.g. for *storing* data in an electronic dictionary.

Concrete Implementations of `Map` in JCF

- The main implementations of `Map<K,V>` are:
  - `HashMap<K,V>` – **hash table** implementation of **map**.
    - ➠ Used when the order of the data in the map is irrelevant, e.g. for *storing* data in an electronic dictionary.
  - `ConcurrentHashMap<K,V>` – a thread-safe, **hash table** implementation of **map** that supports concurrent operations.

Concrete Implementations of `Map` in JCF

- The main implementations of `Map<K,V>` are:
  - `HashMap<K,V>` – **hash table** implementation of **map**.
    - ➠ Used when the order of the data in the map is irrelevant, e.g. for *storing* data in an electronic dictionary.
  - `ConcurrentHashMap<K,V>` – a thread-safe, **hash table** implementation of **map** that supports concurrent operations.
  - `TreeMap<K,V>` – an *ordered* map, ordered by the *natural order* of keys.
    - ➠ Used when the data in the map need to be ordered, e.g. for storing data in a dictionary for *printing* a paper dictionary:

```
private TreeMap<String, LexicalUnit> dictionary;
```

✔

SortedMap

How to Iterate over `Map`?

- `java.util.Map<K,V>` does **not** extends the interface `Iterable`.
  - ➠ Hence, an enhanced-for statement *cannot* be used to iterate over the contents of a `Map` object.

How to Iterate over `Map`?

- `java.util.Map<K,V>` does **not** extends the interface `Iterable`.
  ➠ Hence, an enhanced-for statement *cannot* be used to iterate over the contents of a `Map` object.

- `Map<K,V>` provides convenient methods for obtaining the *contents* of the map:

  ◇ `Set<K> keySet()`:
  Returns a set of the keys contained in this map.

  ◇ `Set<Map.Entry<K,V>> entrySet()`:
  Returns a set of key-to-value mappings contained in this map.

  ◇ `Collection<V> values()`:
  Returns a collection of the values contained in this map.
  ➡ The collection can contain the same object more than once. *Why?*

How to Iterate over `Map`?

- `java.util.Map<K,V>` does **not** extends the interface `Iterable`.
  ➟ Hence, an enhanced-for statement *cannot* be used to iterate over the contents of a `Map` object.

- `Map<K,V>` provides convenient methods for obtaining the *contents* of the map:

  ◇ `Set<K> keySet()`:
  Returns a set of the keys contained in this map.

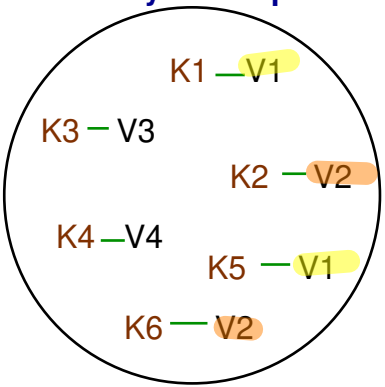
  ◇ `Set<Map.Entry<K,V>> entrySet()`:
  Returns a set of key-to-value mappings contained in this map.

  ◇ `Collection<V> values()`:
  Returns a collection of the values contained in this map.
  ➟ The collection can contain the same object more than once. *Why?*

  ➟ An enhanced-for statement can be used to iterate through the returned data: `Set` and `Collection`. ✔

Different keys may be mapped to the same value

Set of key–value pairs



Different keys can be mapped to the same value. ✔

Iterating over `Map`: An Example

Lab 1

```
1  public void results() {
2      System.out.println(raffle.title() + "\nThe winners are... ");
3
4      // Get info about who has won what.
5      Map<Prize, Ticket> winners = raffle.luckyDraw();
6
7      /* Use an enhanced for statement to obtain the prizes and
8       * their winners one-by-one... */
9      for (Map.Entry<Prize, Ticket> map : winners.entrySet()) {
10
11         Prize prize = map.getKey();      // the prize
12         Ticket winner = map.getValue(); // the winning ticket
13
14         System.out.println(prize.toString() + " goes to " +
15                            winner.buyer());
16     }
17     System.out.println("Many Congratulations!!");
18 }
```

✔

See the `lucky_draw` eclipse Java project in Lab 2 for details.

Iterating over `Map`: Another Example

```
1  public void results() {
2      System.out.println(raffle.title() + "\nThe winners are... ");
3
4      // Get info about who has won what.
5      Map<Prize, Ticket> winners = raffle.luckyDraw();
6
7      /* Use an enhanced for statement to obtain the prizes and
8       * their winners one-by-one... */
9      for (Prize prize : winners.keySet()) {
10
11         // the winning ticket
12         Ticket winner = winners.get(prize);
13
14         System.out.println(prize.toString() + " goes to " +
15                            winner.buyer());
16     }
17     System.out.println("Many Congratulations!!");
18 }
```

lookup value from the key

✔

Learning Outcomes

Learning Outcomes. You should now be able to:

- identify when to use map
- use some implementations of maps in JCF appropriately
- explain the difference between the data structure hash table and the ADT map