Maps

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming Concepts using Java

Week 6

Maps

- The `Collection` interface abstracts properties of grouped data
  - Arrays, lists, sets, ...
  - But not key-value structures like dictionaries

Maps

- The `Collection` interface abstracts properties of grouped data
  - Arrays, lists, sets, ...
  - But not key-value structures like dictionaries
- Key-value structures come under the `Map` interface
  - Two type parameters
  - `K` is the type for keys
  - `V` is the type for values
  - `get(k)` fetches value for key `k`
  - `put(k,v)` updates value for key `k`

```
public interface Map<K,V>{
  V get(Object key);
  V put(K key, V Value);

  boolean containsKey(Object key);
  boolean containsValue(Object value);
  ...
}
```

Maps

- The `Collection` interface abstracts properties of grouped data
  - Arrays, lists, sets, ...
  - But not key-value structures like dictionaries
- Key-value structures come under the `Map` interface
  - Two type parameters
  - `K` is the type for keys
  - `V` is the type for values
  - `get(k)` fetches value for key `k`
  - `put(k,v)` updates value for key `k`

```
public interface Map<K,V>{
  V get(Object key);
  V put(K key, V Value);

  boolean containsKey(Object key);
  boolean containsValue(Object value);
  ...
}
```

- As expected, keys form a set
  - Only one entry per key-value
  - Assigning a fresh value to existing key overwrite the old value
  - `put(k,v)` returns the previous value associated with `k`, or `null`

Updating a map

- Key-value stores are useful to accumulate quantities
  - Frequencies of words in a text
  - Total runs in a tournament

Updating a map

- Key-value stores are useful to accumulate quantities
  - Frequencies of words in a text
  - Total runs in a tournament
- The initialization problem
  - Update the value if the key exists
  - Otherwise, create a new entry

Updating a map

- Key-value stores are useful to accumulate quantities
  - Frequencies of words in a text
  - Total runs in a tournament
- The initialization problem
  - Update the value if the key exists
  - Otherwise, create a new entry
- `Map` has the following default method

```
V getOrDefault(Object key, V defaultValue)
```

Updating a map

- Key-value stores are useful to accumulate quantities
  - Frequencies of words in a text
  - Total runs in a tournament
- The initialization problem
  - Update the value if the key exists
  - Otherwise, create a new entry
- `Map` has the following default method

```
V getOrDefault(Object key, V defaultValue)
```

- For instance

```
Map<String, Integer> scores = ...;
int score = scores.getOrDefault(bat,0);
```

sets `score` to `0` if key `bat` is not present

Updating a map

- Key-value stores are useful to accumulate quantities
  - Frequencies of words in a text
  - Total runs in a tournament
- The initialization problem
  - Update the value if the key exists
  - Otherwise, create a new entry
- `Map` has the following default method

```
V getOrDefault(Object key, V defaultValue)
```

- For instance

```
Map<String, Integer> scores = ...;
int score = scores.getOrDefault(bat,0);
```

sets `score` to `0` if key `bat` is not present

- Now, we can update the entry for key `bat` as follows

```
scores.put(bat,
   scores.getOrDefault(bat,0)+newscore);
   // Add newscore to value of bat
```

Updating a map

- Key-value stores are useful to accumulate quantities
  - Frequencies of words in a text
  - Total runs in a tournament
- The initialization problem
  - Update the value if the key exists
  - Otherwise, create a new entry
- `Map` has the following default method

```
V getOrDefault(Object key, V defaultValue)
```

- For instance

```
Map<String, Integer> scores = ...;
int score = scores.getOrDefault(bat,0);
```

sets `score` to `0` if key `bat` is not present

- Now, we can update the entry for key `bat` as follows

```
scores.put(bat,
   scores.getOrDefault(bat,0)+newscore);
   // Add newscore to value of bat
```

- Alternatively, use `putIfAbsent()` to initialize a missing key

```
scores.putIfAbsent(bat,0);
scores.put(bat,scores.get(bat)+newscore);
```

Updating a map

- Key-value stores are useful to accumulate quantities
  - Frequencies of words in a text
  - Total runs in a tournament
- The initialization problem
  - Update the value if the key exists
  - Otherwise, create a new entry
- `Map` has the following default method

```
V getOrDefault(Object key, V defaultValue)
```

- For instance

```
Map<String, Integer> scores = ...;
int score = scores.getOrDefault(bat,0);
```

sets `score` to `0` if key `bat` is not present

- Now, we can update the entry for key `bat` as follows

```
scores.put(bat,
   scores.getOrDefault(bat,0)+newscore);
   // Add newscore to value of bat
```

- Alternatively, use `putIfAbsent()` to initialize a missing key

```
scores.putIfAbsent(bat,0);
scores.put(bat,scores.get(bat)+newscore);
```

- Or use `merge()`

```
scores.merge(bat,newscore,Integer::sum);
```

  - Initialize to `newscore` if no key `bat`
  - Otherwise, combine current value with `newscore` using `Integer::sum`

Extracting keys and values

- Methods to extract keys and values

```
Set<K> keySet();
Collection<V> values();
Set<Map.Entry<K, V>> entrySet()
```

Extracting keys and values

- Methods to extract keys and values

```
Set<K> keySet();
Collection<V> values();
Set<Map.Entry<K, V>> entrySet()
```

- Keys form a `Set` while values form an arbitrary `Collection`

Extracting keys and values

- Methods to extract keys and values

```
Set<K> keySet();
Collection<V> values();
Set<Map.Entry<K, V>> entrySet()
```

- Keys form a `Set` while values form an arbitrary `Collection`
- Key-value pairs form a set over a special type `Map.Entry`

Extracting keys and values

- Methods to extract keys and values

```
Set<K> keySet();
Collection<V> values();
Set<Map.Entry<K, V>> entrySet()
```

- Keys form a `Set` while values form an arbitrary `Collection`
- Key-value pairs form a set over a special type `Map.Entry`
- Java calls these views

Extracting keys and values

- Methods to extract keys and values

```
Set<K> keySet();
Collection<V> values();
Set<Map.Entry<K, V>> entrySet()
```

- Keys form a `Set` while values form an arbitrary `Collection`
- Key-value pairs form a set over a special type `Map.Entry`
- Java calls these views

- Can now iterate through a `Map`

```
Set<String> keys = strmap.keySet();
for (String key : keys) {
  do something with key
}
```

Extracting keys and values

- Methods to extract keys and values

```
Set<K> keySet();
Collection<V> values();
Set<Map.Entry<K, V>> entrySet()
```

- Keys form a `Set` while values form an arbitrary `Collection`
- Key-value pairs form a set over a special type `Map.Entry`
- Java calls these views

- Can now iterate through a `Map`

```
Set<String> keys = strmap.keySet();
for (String key : keys) {
  do something with key
}
```

- Use `entrySet()` to operate on key and associated value without looking up map again

```
for (Map.Entry<String, Employee> entry :
                        staff.entrySet()){
  String k = entry.getKey();
  Employee v = entry.getValue();
  do something with k, v
}
```

Concrete implementations of `Map`

`HashMap`

- Similar to `HashSet`
- Use a hash table to store keys and values
- No fixed order over keys returned by `keySet()`

Concrete implementations of `Map`

`HashMap`

- Similar to `HashSet`
- Use a hash table to store keys and values
- No fixed order over keys returned by `keySet()`

`TreeMap`

- Similar to `TreeSet`
- Use a balanced search tree to store keys and values
- Iterator over `keySet()` will process keys in sorted order

Concrete implementations of `Map`

`HashMap`

- Similar to `HashSet`
- Use a hash table to store keys and values
- No fixed order over keys returned by `keySet()`

`TreeMap`

- Similar to `TreeSet`
- Use a balanced search tree to store keys and values
- Iterator over `keySet()` will process keys in sorted order

`LinkedHashMap`

- Remembers the order in which keys were inserted
- Hash table entries are also connected as a (doubly) linked list
- Iterators over both `keySet()` and `value()` enumerate in order of insertion

Concrete implementations of `Map`

`HashMap`

- Similar to `HashSet`
- Use a hash table to store keys and values
- No fixed order over keys returned by `keySet()`

`TreeMap`

- Similar to `TreeSet`
- Use a balanced search tree to store keys and values
- Iterator over `keySet()` will process keys in sorted order

`LinkedHashMap`

- Remembers the order in which keys were inserted
- Hash table entries are also connected as a (doubly) linked list
- Iterators over both `keySet()` and `value()` enumerate in order of insertion
- Can also use access order
  - Each `get()` or `put()` moves key-value pair to end of list
  - Process entries in least recently used order — scheduling, caching

Concrete implementations of `Map`

`HashMap`

- Similar to `HashSet`
- Use a hash table to store keys and values
- No fixed order over keys returned by `keySet()`

`TreeMap`

- Similar to `TreeSet`
- Use a balanced search tree to store keys and values
- Iterator over `keySet()` will process keys in sorted order

`LinkedHashMap`

- Remembers the order in which keys were inserted
- Hash table entries are also connected as a (doubly) linked list
- Iterators over both `keySet()` and `value()` enumerate in order of insertion
- Can also use access order
  - Each `get()` or `put()` moves key-value pair to end of list
  - Process entries in least recently used order — scheduling, caching
- Similarly, `LinkedHashSet`

Summary

- The `Map` interface captures properties of key-value stores
  - `get()`, `put()`, `containsKey()`, `containsValue()`, ...
- Parameterized by two type variables, `K` for keys and `V` for values
- Keys form a set
- Different ways to update a key entry, depending on whether the key already exists
  - `getOrDefault()`, `putIfAbsent()`, `merge()`
- Extract keys as a `Set`, values as a `Collection`, key-value pairs as a `Set`
  - `keySet()`, `values()`, `entrySet()`
- Use these "views" to iterate over all key-value pairs in the map
- Concrete implementations: `HashMap`, `TreeMap`, `LinkedHashMap`