Sets and Queues

- Queues/Stacks
  - No random access
  - LIFO: Last-in-first-out
  - FIFO: First-in-first-out
- Sets
  - Each element only contained once
  - No order, access via iterators
- Object identity
  - Equals, hashCode
  - Memory address

Relevant (for us):

AI and digital markets acts

General data protection regulation

Climate crisis

War and peace

EUROPEAN ELECTIONS
6-9 JUNE 2024

#USE
YOUR
VOTE

ELECTIONS.EUROPA.EU

UNIVERSITÄT
ZU KÖLN

Session 7: Collections, Part 3 (Maps) and Recursion, Part 1

Fortgeschrittene Programmierung (Java 2)

Nils Reiter
nils.reiter@uni-koeln.de

June 5, 2024

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Section 1

Maps

Looking Back: Exercise 2

```java
public class ATM {
  // ...
  protected Account getAccount(int id) {
    for (Account account : accounts)
      if (account.getId() == id)
        return account;
    return null;
  }
  // ...
}
```

Looking Back: Exercise 2

```
public class ATM {
  // ...
  protected Account getAccount(int id) {
    for (Account account : accounts)
      if (account.getId() == id)
        return account;
    return null;
  }
  // ...
}
```

- Getting the account based on an id value
- Alternatives?

Looking Back: Exercise 2

```
public class ATM {
  // ...
  protected Account getAccount(int id) {
    for (Account account : accounts)
      if (account.getId() == id)
        return account;
    return null;
  }
  // ...
}
```

- Getting the account based on an id value
- Alternatives?
  - Ensuring that account id and array index position are the same:

    ```
    protected Account getAccount(int id) { return accounts[id]; }
    ```

    - Not very flexible
    - Only works if id numbers are integers

Looking Back: Exercise 2

```
public class ATM {
  // ...
  protected Account getAccount(int id) {
    for (Account account : accounts)
      if (account.getId() == id)
        return account;
    return null;
  }
  // ...
}
```

- Getting the account based on an id value
- Alternatives?
  - Ensuring that account id and array index position are the same:

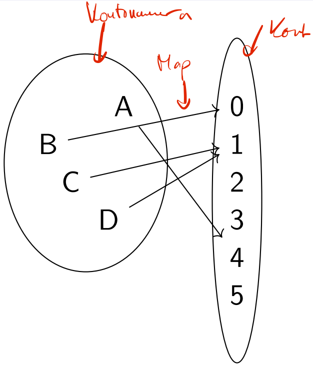
    ```
    protected Account getAccount(int id) { return accounts[id]; }
    ```

    - Not very flexible
    - Only works if id numbers are integers
  - Maps!

Map

Definition (Mapping)

Any prescribed way of assigning to each object in one set a particular object in another (or the same) set. Dictionary

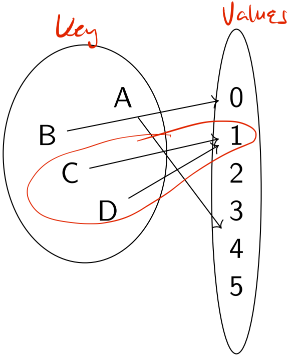


- A mapping from $\{A, B, C, D\}$ to $\{0, 1, \ldots, 5\}$
- Practically useful as “key value store”

Map

Definition (Mapping)

Any prescribed way of assigning to each object in one set a particular object in another (or the same) set. Dictionary



- A mapping from $\{A, B, C, D\}$ to $\{0, 1, \ldots, 5\}$
- Practically useful as “key value store”
- Arrays map integer numbers to objects or primitive values
  - ...with the usual restrictions of arrays

The Map Interface

java.util.Map<K,V>

- Unordered mapping between objects of type K and objects of type V
- Two generic variables: K (= keys) and V (= values)
  - E.g.: `Map<String, Student>`

The Map Interface

📖 java.util.Map<K,V>

- Unordered mapping between objects of type K and objects of type V
- Two generic variables: K (= keys) and V (= values)
  - E.g.: `Map<String, Student>`
- Getting and setting
  - `V put(K key, V value)`: Put something into the map, potentially overwriting a value
  - `V get(Object key)`: Retrieve some value from the map. Returns null if key not defined
  - `V getOrDefault(Object key, V defaultValue)`: Return a value or a default value

The Map Interface

java.util.Map<K,V>

- Unordered mapping between objects of type K and objects of type V
- Two generic variables: K (= keys) and V (= values)
  - E.g.: `Map<String, Student>`
- Getting and setting
  - `V put(K key, V value)`: Put something into the map, potentially overwriting a value
  - `V get(Object key)`: Retrieve some value from the map. Returns null if key not defined
  - `V getOrDefault(Object key, V defaultValue)`: Return a value or a default value
- Views: Non-independent "perspectives" on the object
  - `Set<K> keySet()`: Returns the keys as a set
  - `Collection<V> values()`: Returns the values as a collection
  - `Set<Map.Entry<K,V>> entrySet()`: Returns the entries as a set of pairs

Implementation

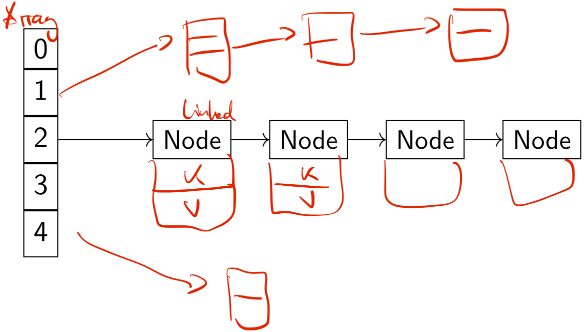
Most commonly used: java.util.HashMap<K,V>

- Relies on `hashCode()` and `equals(...)` methods of the keys
- Hash map consists of an array of length n, which stores linked lists (“buckets”)
- Linked lists contain Node<K,V> objects

Implementation

Most commonly used: java.util.HashMap<K,V>

- Relies on `hashCode()` and `equals(...)` methods of the keys
- Hash map consists of an array of length n, which stores linked lists ("buckets")
- Linked lists contain Node<K,V> objects



Retrieval

- How to retrieve an object given key K?
  1. Identify the correct bucket via `hashCode()`
  2. Walk the linked list, until `K.equals(L)` returns true ☑

Retrieval

12 : 7 = 1 Rest 5

23 : 7 = 3 Rest 2

/

%

- How to retrieve an object given key K?
  1. Identify the correct bucket via `hashCode()`
  2. Walk the linked list, until `K.equals(L)` returns true ☑

Identify the correct bucket

- `hashCode()` returns an arbitrary int number, but we only have a limited amount of buckets (n) – How to go from an arbitrary int to an int in a small range?

16

Retrieval

- How to retrieve an object given key K?
  1. Identify the correct bucket via `hashCode()`
  2. Walk the linked list, until `K.equals(L)` returns true ☑

Identify the correct bucket

- `hashCode()` returns an arbitrary int number, but we only have a limited amount of buckets (n) – How to go from an arbitrary int to an int in a small range?
- Two options
  1. Modulus operator: `int bucketIndex = K.hashCode() % buckets.length;`
  2. Bitwise and operator: `int bucketIndex = K.hashCode() & buckets.length - 1;`

Bitwise And

- Rarely used operation: `&`
  - Not the same as the boolean operator `&&`
- Considering each binary position, set this position to 1 if both operands have a 1

Bitwise And

- Rarely used operation: `&`
  - Not the same as the boolean operator `&&`
- Considering each binary position, set this position to 1 if both operands have a 1

```
5 & 3 // yields 1, because 101 & 011 = 001
15 & 7 // yields 7, because 1111 & 0111 = 0111
16 & 7 // yields 0, because 10000 & 00111 = 00000
```

Bitwise And

- Rarely used operation: `&`
  - Not the same as the boolean operator `&&`
- Considering each binary position, set this position to 1 if both operands have a 1

```
5 & 3 // yields 1, because 101 & 011 = 001
15 & 7 // yields 7, because 1111 & 0111 = 0111
16 & 7 // yields 0, because 10000 & 00111 = 00000
```

Fun fact: You can define integers as binary literals: `int a = 0b1011011; //yields 91` demo

Maps and Efficiency

- With a constant number of buckets, a larger hash map will be very slow eventually
  - Because we have to iterate over a very long list
- More buckets require more space, but make lookup faster

Maps and Efficiency

- With a constant number of buckets, a larger hash map will be very slow eventually
  - Because we have to iterate over a very long list
- More buckets require more space, but make lookup faster
- 📘 java.util.HashMap<K,V> internally increases the number of buckets if the map is too full
  - "Capacity": Number of buckets
  - "size": Number of entries
  - If $\frac{\text{size}}{\text{capacity}} >$ load factor, increase number of buckets (default load factor: 0.75)

Section 2

Recursion, part 1

```java
public class MyLinkedList<T> implements List<T> {

  // ...

  public int size() {
    // TODO Implement!
    int i = 0;
    for (T x : this)
      i++;
    return i;
  }

  // ...

}
```

MIEJSCE
ZAREZERWOWANE

Recursive Implementation

demo

Exercise

`https://github.com/idh-cologne-java-2-summer-2024/exercise-07`