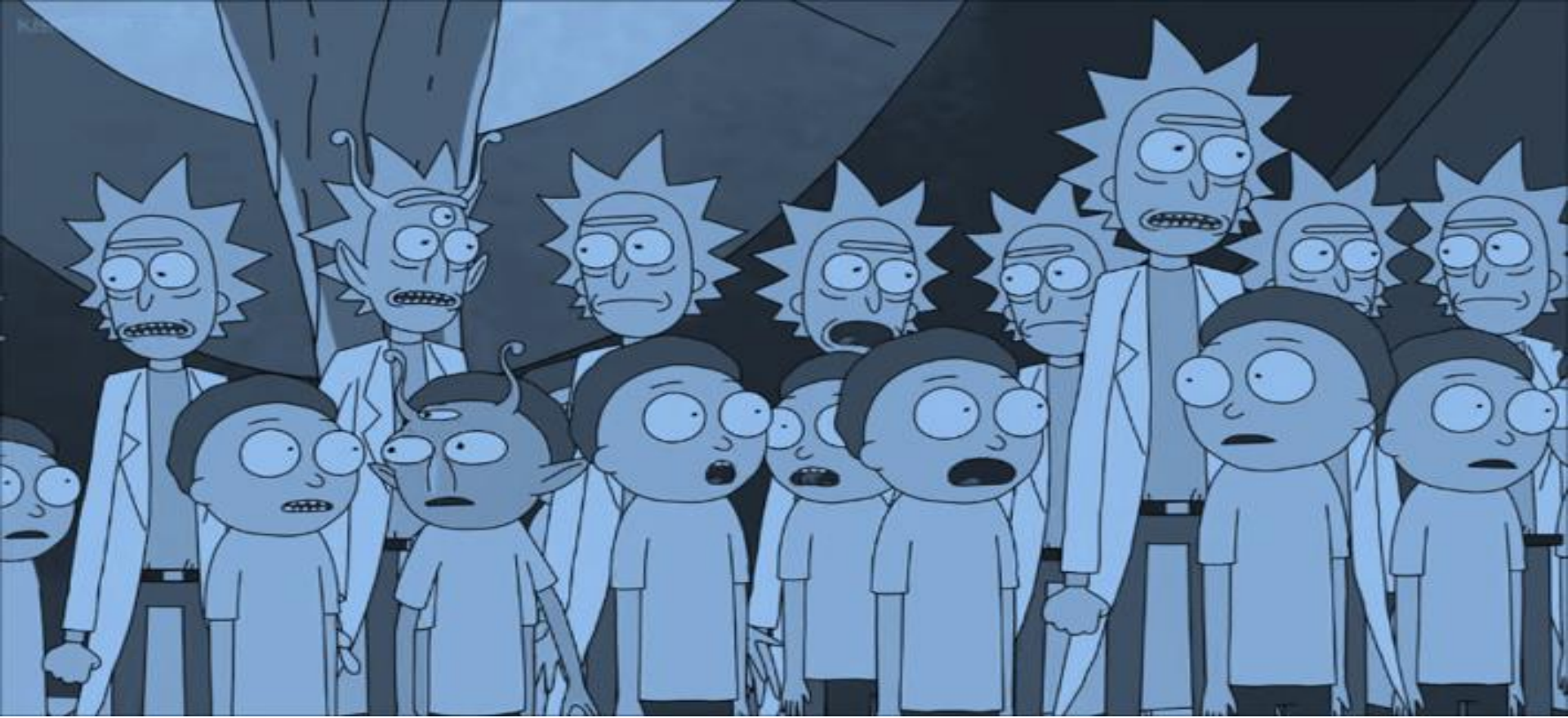

Advanced Object-Oriented Programming

CPT204 – Lab 12

Erick Purwanto

Xi'an Jiaotong-Liverpool University

西交利物浦大学

CPT204 Advanced Object-Oriented Programming

Lab 12

Comparator, Hash Table, Map

Welcome !

- Welcome to Lab 12 !
  - We are going to create a comparator for our hash-based set (hash table) based on the size,
  - and to create a hash-based map with average constant-time operations!

- You will find in this lab
  1. Lab Exercise 12.1 - 12.4, and their hints
  2. Exercise 12.1 - 12.3

- Download **lab12** zip files from Learning Mall
- Import the **lab12** files and the library to an IntelliJ project
  - Read **lab1** again for reference

Lab Exercise 12.1 HASet SIZE COMPARATOR

- Complete the inner non-static class SizeComparator and its getter method getSizeComparator() to create a comparator of two sets of HASet based on their size.
- Test case 1:

```
Comparator<HASet<String>> sizeComp = new HASet<String>().getSizeComparator();
HASet<String> set1 = new HASet<>();
set1.add("a");
set1.add("b");
HASet<String> set2 = new HASet<>();
set2.add("c");
sizeComp.compare(set1, set2) > 0                    →       true
set2.add("d");
sizeComp.compare(set1, set2)                        →       0
```

WARNING: Hints to the exercise on the next slide

Please try to solve the exercise by yourself first...

Lab Exercise 12.1 HASet SIZE COMPARATOR Hints

- Follow the design pattern discussed in the lecture, except that it is non-static.

Map

- **Map**, also known as Dictionary, is a data structure that stores key and value pairs (see Week 3)
  - you can add/put and remove pairs of key and value (mappings)
  - but most importantly, it supports ***fast*** searching of value based on its key
- We are going to implement Hash-based Map using
  - ArrayList to store and also to implement the buckets
  - HashSet (see next slide) to store all your keys
  - Separate Chaining and Resizing Techniques (see lecture slides) to achieve average constant-time operations
- Implement resizing such that the capacity is doubled when the load factor N/M exceeds the given loadFactor
  - there will be default values given in the skeleton code

HashSet

- Hash tables are the most popular implementation for sets
- In Java, they are implemented as java.util.**HashSet**
  - it can store any objects
  - you can store new items by add, and check membership by contains
  - it is Iterable
- You are using it to store keys in your HAMap
  - and for implementing many methods in the exercises and assignments

- Thus, in this lab, we are using ArrayList, HashSet and iterator libraries
  - you are not allowed to import any other libraries

Test Case for Lab Exercise 12.2 - 12.4, Exercise 12.1 - 12.3

- Test case 1:

```
HAMap<String, Integer> map = new HAMap<>();

map.containsKey("a");                  →        false
map.put("a", 1);
map.containsKey("a");                  →        true
map.get("a");                          →        1
map.size();                            →        1
map.put("b", 2);
map.put("c", 3);
map.remove("a", 1);                    →        1
for (String key : map) {
    System.out.println("(" + key + ", " + map.get(key) + ")");    → (b, 2)↲
}                                                                    (c, 3)
map.clear();
map.size();                            →        0
map.containsKey("b");                  →        false
map.containsKey("c");                  →        false
```

Lab Exercise 12.2 HAMap CONSTRUCTOR

- Complete **three constructors** of HAMap that take zero, one, or two arguments.
- Use the given default values for the absent arguments.
  - DEFAULT_CAPACITY = 16 and DEFAULT_LOAD_FACTOR = 1.5.
- Initialize all five member variables,
  - with the initialCapacity being the starting numBuckets.

WARNING: Hints to the exercise on the next slide

Please try to solve the exercise by yourself first...

Lab Exercise 12.2 HAMap CONSTRUCTOR Hints

- you can implement the constructor taking two arguments first,
  - and then calling it for the other two constructors passing the default value(s).
- initialize the buckets list with initialCapacity empty Arraylists.
- initialize the set of key with an empty HashSet.

Lab Exercise 12.3 HAMap CLEAR

- Complete the method void **clear**() of HAMap that removes all the entries in the map.
- Keep the current number of buckets the same.
  - We do not implement halving/resizing down this time.

WARNING: Hints to the exercise on the next slide

Please try to solve the exercise by yourself first...

Lab Exercise 12.3 HAMap CLEAR Hints

- reset the buckets and set of keys to the empty buckets and an empty set.
- reset the number of entries.

Lab Exercise 12.4 HAMap CONTAINSKEY and ITERATOR

- Complete the method boolean **containsKey**(K key) and method Iterator<K> **iterator**() of HAMap.
- *containsKey* returns true if the entry with the specified key exists in the map.
  - returns false otherwise.
- *iterator* returns an Iterator that iterates over the stored keys.

WARNING: Hints to the exercise on the next slide

Please try to solve the exercise by yourself first...

Lab Exercise 12.4 HAMap CONTAINSKEY and ITERATOR Hints

- use the contains and iterator methods of the HashSet.

Exercise 12.1 HAMap GET

- Complete the method V **get**(K key) of HAMap.
- It returns the value to which the specified key is mapped,
  - and return *null* if this map contains *no* entries of the key.

Exercise 12.2 HAMap PUT

- Complete the method void **put**(K key, V value) of HAMap that adds the key, value entry into the map.
- If the same key is added more than once, then the value must be ***replaced*** each time.
- Assume that *null* keys will *never* be added.
- Before adding the entry, check if the ratio of the number of entries and the number of buckets exceeds the load factor.
  - if so, resize and double the number of buckets, even if *no* new entry is added after.

Exercise 12.3 HAMap REMOVE

- Complete the method V **remove**(K key, V value) of HAMap.
- It removes the entry for the specified key only if it is currently mapped to the specified value.
  - in that case, return the value.
- If the key, value entry does **not** exist in the map, return *null*.
- We do not implement halving/resizing down this time.

Thank you for your attention !

- In this lab, you have learned:
  - To make a comparator comparing your data structure according to your preference
  - To create a map/dictionary using hash table techniques, with ArrayList and HashSet as the underlying data structures