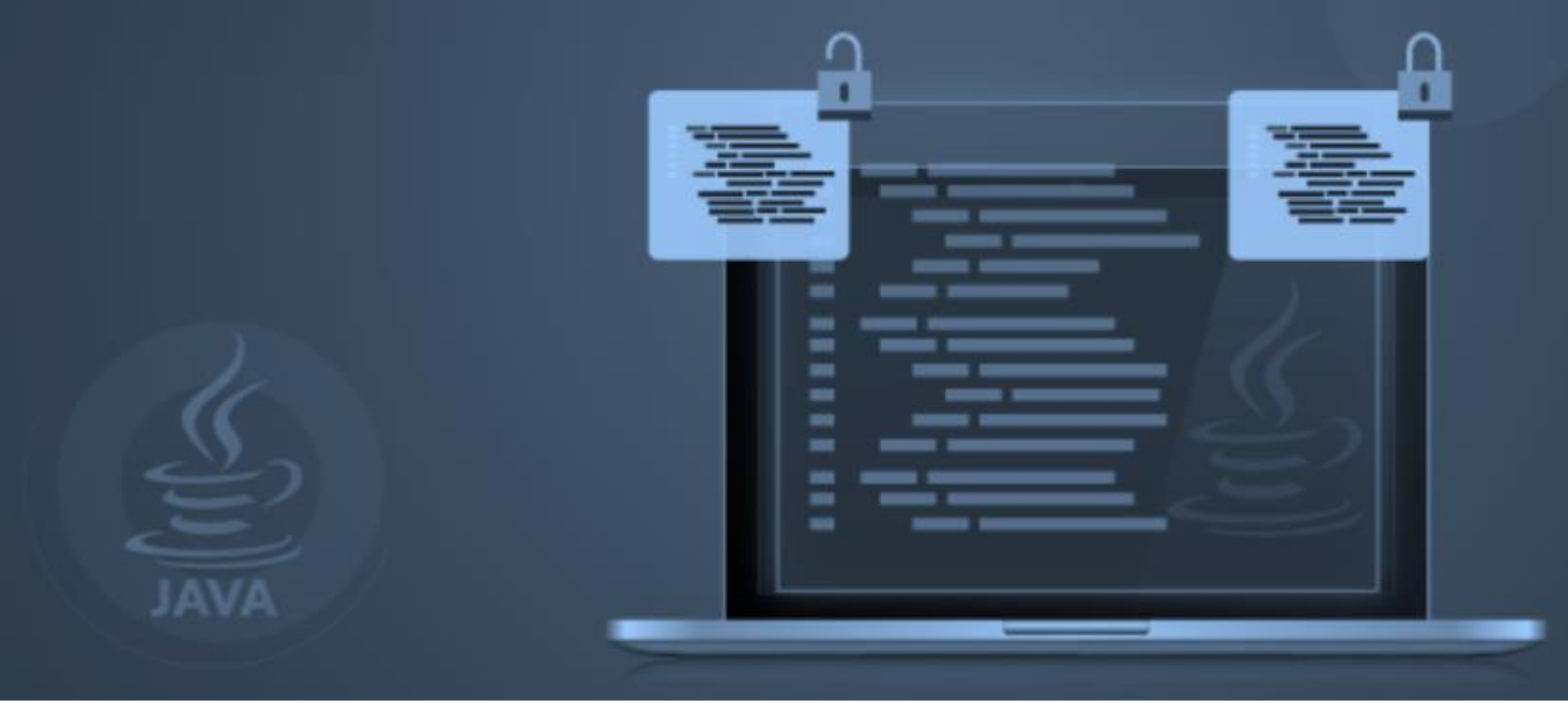

Advanced Object-Oriented Programming

CPT204 – Lab 2

Erick Purwanto

Xi'an Jiaotong-Liverpool University

西交利物浦大学

CPT204 Advanced Object-Oriented Programming

Lab 2

Checking 2, Testing 2, List, Map

Welcome !

- Welcome to Lab 2 !
  - We are going continue practising with testing while reviewing list and map

- You will find in this lab
  1. Lab Exercise 2.1, 2.2, and their hints
  2. Exercise 2.1 - 2.4

- Download **lab2** zip file from Learning Mall
- Don't forget to import the **lab2** files and the library into an IntelliJ project
  - Read **lab1** again for reference

Lab Exercise 2.1 MaxStretch

- We define the **stretch** of a value in a list to be the number of elements between that two leftmost and rightmost values in that list, inclusive
  For example, the stretch of 2 in [5, 2, 2, 5, 2] is 4 (from **2**, 2, 5, **2**)

  A single value in a list has a stretch of 1
  Write a method that returns the *maximum* stretch found in the input list

- Test case 1:
  maxStretch([8, 5, 1, 2, 3, 4, 5, 10]) = 6 (from **5**, 1, 2, 3, 4, **5**)

- Test case 2:
  maxStretch([2, 7, 1, 2, 3, 7]) = 5 (from **7**, 1, 2, 3, **7**)

Lab Exercise 2.1 MaxStretch

- Skeleton code:

```
/**
  * Find the largest stretch in a list.
  * For example, maxStretch([8, 5, 1, 2, 3, 4, 5, 10]) = 6.
  * @param list is a list of integers.
  * @return the largest stretch in list.
  */
public static int maxStretch(List<Integer> list) {



}
```

Continue with Test-Driven Programming

We use the same approach as last week, before starting to write the maxStretch method, write the test code for it first !

- Open MaxStretchTest.java, create and add more test cases, for example:

```java
@Test
public void testOverlapMaxStretch() {
    List<Integer> list = Arrays.asList(2, 3, 2, 3, 3, 2, 2, 3);
    assertEquals( expected: 7, Stretch.maxStretch(list));
}
@Test
public void testEmptyList() {
    List<Integer> list = Arrays.asList();
    assertEquals( expected: 0, Stretch.maxStretch(list));
}
@Test
public void testSingletonList() {
    List<Integer> list = Arrays.asList(55555);
    assertEquals( expected: 1, Stretch.maxStretch(list));
}
@Test
public void testOneWholeStretch() {
    List<Integer> list = Arrays.asList(4, 4, 4, 4);
    assertEquals( expected: 4, Stretch.maxStretch(list));
}
```

use descriptive name for your test

include corner cases!

WARNING: Hints to the exercise on the next slide

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

Lab Exercise 2.1 MaxStretch Hints

- One way to solve this is by using two nested loops
  - two pointers: one from left to right, one from right to left
- Use a variable, say max, to keep track maximum so far
- The outer loop goes through every element starting from the beginning
- The inner loop goes from the end of the list, up until the outer loop index
  - stops when finding the same element pointed by the outer loop
- After inner loop finishes, compute the stretch by the difference of inner and outer loop indices
  - update max if larger stretch is found
- After outer loop finishes, the max stretch is found
  - return it

Lab Exercise 2.2 EvenAppend

- Given an input of a list of strings, write a method to build a result string as follows: when a string appears the 2nd, 4th, 6th, etc. time in the list, append the string to the result.
  Return the empty string if no string appears a 2nd time.
- Test case 1:
  evenAppend(["a", "b", "a"]) → "a"
- Test case 2:
  evenAppend(["a", "b", "b", "a", "a"]) → "ba"

Lab Exercise 2.2 EvenAppend

- Skeleton code:

```
/**
 * Append words that appear the 2nd, 4th, 6th, etc. time in a list.
 * For example, evenAppend(["a", "b", "b", "a", "a"]) → "ba".
 * @param list is a list of words.
 * @return a concatenation of even appearing words.
 */
public static String evenAppend(List<String> list) {



}
```

Continue with Test-Driven Programming

We use the same approach as last week, write the test code first

- Open EvenAppendTest.java, create and add more test cases, for example:

```
@Test
public void testManyEven() {
    List<String> list = Arrays.asList("a", "b", "b", "b", "a", "c", "a", "a", "a", "b", "a", "b", "c");
    assertEquals( expected: "baabac", EvenAppend.evenAppend(list));
}
@Test
public void testEmptyList() {
    List<String> list = Arrays.asList();
    assertEquals( expected: "", EvenAppend.evenAppend(list));
}
@Test
public void testSingletonList() {
    List<String> list = Arrays.asList("one");
    assertEquals( expected: "", EvenAppend.evenAppend(list));
}
@Test
public void testOverlapEven() {
    List<String> list = Arrays.asList("xxx", "xxx", "y", "yy", "xx", "xxx", "zz", "yy", "zz", "xx", "y");
    assertEquals( expected: "xxxyyzzxxy", EvenAppend.evenAppend(list));
}
```

WARNING: Hints to the exercise on the next slide

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

Lab Exercise 2.2 EvenAppend Hints

- Create a map, with String as key, and Integer as value
- Create a string res to store the result
- Iterate over all Strings in the list
  - If it is not in the map yet, store it, with value 1
  - Else (it is in the map)

    check the value
    - if it is odd (the next one would make it even occurrence)

      append the key (string) into res
    - else do nothing

    increment the value
- Return res

Week 2 Online Programming Exercises

- Start with creating a good set of test cases first !
  - Include the **corner cases**, such as empty list for countRuns;
    a list with one element for isPartitionable;
    empty list and list with one/two non-empty strings for sameFirstLetter and matchSwap;
    as well as when it contains empty strings

- Use IntelliJ & JUnit to write and test your code

Exercise 2.1 Count Runs

- We define a **runs** in a list is a series of 2 or more adjacent elements of the same value
- Write a method to return the number of runs in the input list

- Test case 1:
  countRuns([1, 2, 2, 2, 3]) = 1 (which is 2, 2, 2)
- Test case 2:
  countRuns([1, 1, 2, 3, 4, 5, 5]) = 2 (which is 1, 1; and 5, 5)

Exercise 2.1 Count Runs

- Skeleton code:

```
/**
  * Count the number of runs in a list.
  * For example, countRuns([1, 2, 2, 2, 3]) = 1.
  * @param list is a list of integers.
  * @return the number of runs in list.
  */
public static int countRuns(List<Integer> list) {



}
```

Exercise 2.2 Partitionable

- We define a list to be **partitionable**, if there is a place in between two indices in that list where the *sum* of the numbers on one side is *equal to* the *sum* of the numbers on the other side
- Given as an input a *non-empty* list, write a method that returns true if and only of the list is partitionable

- Test cases:
  - isPartitionable([1, 1, 1, 2, 1]) --> true
  - isPartitionable([2, 1, 1, 2, 1]) --> false

Exercise 2.2 Partitionable

- Skeleton code:

```
/**
 * Decide whether a list is partitionable.
 * For example, isPartitionable([1, 1, 1, 2, 1]) -> true,
 * and isPartitionable([2, 1, 1, 2, 1]) -> false.
 * @param list is a non-empty list of integers.
 * @return true iff list is partitionable.
 */
public static boolean isPartitionable(List<Integer> list) {



    return true;
}
```

Exercise 2.3 Same First Letter

- Given a list of non-empty strings, write a method that returns a Map<String, String> with a key for every different first letter seen, with the value of all the strings starting with that letter appended with a comma (,) together in the order they appear in the list
- Test case 1:
  sameFirstLetter(["alice", "bob", "apple", "banana"])
  → {"a": "alice,apple", "b": "bob,banana"}
- Test case 2:
  sameFirstLetter(["after", "all", "this", "time", "always"])
  → {"a": "after,all,always", "t": "this,time"}

Exercise 2.3 Same First Letter

- Skeleton code:

```
/**
 * Create a map with first letter as key and words with that same
 * first letter separated by comma.
 * For example, numWords(["alice", "bob", "apple", "banana"]) →
 * {"a": "alice,apple", "b": "bob,banana"}.
 * @param list is a list of strings.
 * The strings are non-empty.
 * @return a map with first letter and comma-separated-words pair.
 */
public static Map<String, String> sameFirstLetter(List<String> list) {


}
```

Exercise 2.4 Match Swap

- We define that 2 strings **match** if they are non-empty and their *first letters* are the same
- Given a list of *non-empty* strings, return that list modified as follows: if a string matches an earlier string in the list, swap those 2 strings in the list. After a position in the list has been swapped, it no longer matches anything.
- Hint: Using a map, this can be solved by making just one pass over the list.
- Test case 1:
  matchSwap(["apple", "avocado"]) → ["avocado", "apple"]
- Test case 2:
  matchSwap(["ab", "ac", "ad", "ae", "af"]) → ["ac", "ab", "ae", "ad", "af"]
- Test case 3:
  matchSwap(["ap", "bp", "cp", "aq", "cq", "bq"])
  → ["aq", "bq", "cq", "ap", "cp", "bp"]

Exercise 2.4 Match Swap

- Skeleton code:

```
/**
 * Modify a list of strings such that two strings with same
 * first letter are swapped.
 * For example, matchSwap(["ap", "bp", "cp", "aq", "cq", "bq"]) →
 * ["aq", "bq", "cq", "ap", "cp", "bp"].
 * @param list is a list of strings.
 * The strings are non-empty.
 * @return the modified list.
 */
public static List<String> matchSwap(List<String> list) {


}
```

Thank you for your attention !

- In this lab, you have learned:
  - Creating good test cases
    - Including corner cases
  - Test-driven Programming
  - Reviewing operations on Lists and Maps