Advanced Object-Oriented Programming

CPT204 – Lab 10
Erick Purwanto

Xi'an Jiaotong-Liverpool University
西交利物浦大學

CPT204 Advanced Object-Oriented Programming

Lab 10

Interface, Inheritance, Iterable, Set

Welcome !

- Welcome to Lab 10 !
    - We are going to create an interface for ARDeque and make it iterable,
    - and implement an array-based set ARSet
    - We will also create a new data structure by extending another data structure, and equip it with a new method

- You will find in this lab
    1. Lab Exercise 10.1 - 10.3, and their hints
    2. Exercise 10.1 - 10.3

- Download **lab10** zip files from Learning Mall
- Don't forget to import the **lab10** files and the library into an IntelliJ project
    - Read **lab1** again for reference

Lab Exercise 10.1 ARDeque DEQUE INTERFACE

- Complete the interface of ARDeque called **Deque**.

- It will be implemented by ARDeque as follows:
  public class ARDeque<T> implements Deque<T> { … }

- In ARDeque: size, methods in Lab Exercise 8.2 - 8.4 and Exercise 8.1 - 8.3 will be annotated with **@Override**

Lab Exercise 10.1 ARDeque DEQUE INTERFACE Test Case

- Test case 1:

```
Deque<String> deque = new ARDeque<>();
deque.size();                              →       0
deque.addFirst("a");
deque.addLast("b");
deque.addLast("c");
deque.get(0);                              →       "a"
deque.get(2);                              →       "c"
deque.printDeque();                        →       "a b c↵"
deque.delFirst();                          →       "a"
deque.delLast();                           →       "c"
deque.get(0);                              →       "b"
deque.size();                              →       1
```

WARNING: Hints to the exercise on the next slide

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

Lab Exercise 10.1 ARDeque DEQUE INTERFACE Hints

- Use keyword interface and type parameter
- There will be seven method signatures
  - end with semicolon

Lab Exercise 10.2 RSLList ROTATE RIGHT

- Complete the class public **RSLList** and method void **rotateRight**().
- RSLList is a subclass of SLList.
- It implements a new method called rotateRight that moves the back item to the front.
  - for example, suppose we have ["a", "b", "c"], after rotateRight, we will have ["c", "a", "b"].
  - do nothing if the list is empty.
- The class will be instantiated and the method will be called as in the test case on the next page.

Lab Exercise 10.2 RSLList ROTATE RIGHT Test Case

- Test case 1:

```
RSLList<String> rlist = new RSLList<>();
rlist.addLast("a");
rlist.addLast("b");
rlist.addLast("c");
rlist.rotateRight();
rlist.get(0);                       →       "c"
rlist.get(1);                       →       "a"
rlist.get(2);                       →       "b"
```

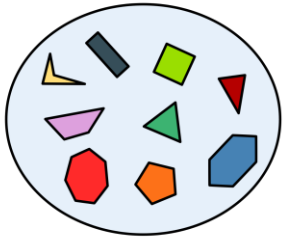
WARNING: Hints to the exercise on the next slide

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

Lab Exercise 10.2 RSLList ROTATE RIGHT Hints

- Set RSLList to be a subclass of SLList
  - using the keyword extend, followed by SLList with the same type parameter
- Write the definition of rotateRight
  - you can simply use methods inherited from SLList to implement this method
  - you need to take special care if the list is empty

Sets



- The next exercise and two assignments are about Sets
    - Set stores a group of values with **no** duplicates
    - Set has **no** sense of order: either an item is in the set, or not

- We will implement it using an array and generics
    - The array has size 100
    - We assume that the items added will **not** exceed 100

Lab Exercise 10.3 ARSet ITERATOR

- Complete the method **iterator**, and the private class **ARSetIterator**.
- The class ARSet implements the Iterable interface.
    - The java.util.Iterator is imported.
- The iterator method and the private class will be used in for-each / enhanced for loop as in the case next page.
    - Although the order of items in a set actually does not matter, iterate in the order of the items *added*.

Lab Exercise 10.3 ARSet ITERATOR Test Case

- Test case 1:

```
ARSet<String> set = new ARSet<>();
set.add("a");
set.add("b");
set.add("c");

for (String item : set) {
    System.out.print(item + " ");
}                                             →       "a b c "
```

WARNING: Hints to the exercise on the next slide

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

Lab Exercise 10.3 ARSet ITERATOR Hints

- Follow the implementation to enable the for-each/enhanced for loop in the lecture notes
    - it will be very similar

Exercise 10.1 ARSet CONTAINS

- Complete the method boolean **contains**(T item).
- It checks whether an item is inside the set.
    - the method returns *true* iff the set contains the item.

Exercise 10.2 ARSet ADD

- Complete the method void **add**(T item).
- It adds an item into the set if it is **not** already inside.
- It throws IllegalArgumentException if item is *null*.

Test Case for Exercise 10.1, 10.2

- Test case 1:

```
ARSet<String> set = new ARSet<>();
set.add("a");
set.contains("a");                          →     true
set.size();                                 →     1
for (String item : set) {
    System.out.println(item);
}                                           →     "a↵"
```

Exercise 10.3 ARDeque ITERATOR

- Complete the method **iterator**, and the private class **ARDequeIterator**.
- The class ARDeque implements the Iterable interface.
    - The java.util.Iterator is imported.
- The iterator method and the private class will be used in for-each / enhanced for loop as in the case next page.
    - Iterate from the *first* to the *last* item in the deque.

Exercise 10.3 ARDeque ITERATOR Test Case

- Test case 1:

```
ARDeque<String> deque = new ARDeque<>();
deque.add("a");
deque.add("b");
deque.add("c");

for (String item : deque) {
    System.out.print(item + " ");
}                                         →      "a b c "
```

Thank you for your attention !

- In this lab, you have learned:
    - To create an interface
    - To create a subclass and to use the superclass's methods to create a new method
    - To equip a data structure with an iterator to enable enhanced for loop
    - To create a data structure called Set