COMP2026 Problem Solving Using Object Oriented Programming

Laboratory 13

Part A Discovery Exercises

Task 1: Lambda Expression

a) Given the following interface.

```
public interface NumChecker {
    public boolean check(int n);
}
```

Write lambda expressions to check whether the given integer is

i. divisible by 3
ii. in between -5 and 10 inclusive

```
public class MyMainClass {
    public static void main(String[] args){
        new  MyMainClass().runApp();
    }

    /**
     * Print all the elements in the given integer array that
     * pass the check
     *
     * @param a - an integer array
     * @param c - a NumberChecker
     */
    public void printElements(int[] a, NumChecker c){
        for (int i = 0; i < a.length; i++){
            if(c.check(a[i])){
                System.out.print(a[i] + " ");
            }
        }
        System.out.println();
    }

    public void runApp(){
        int[] intAry = {34, 6, 21, -1, -32, 24, -97, 76, 9};
        //Example
        System.out.print("Positive Elements: ");
        printElements(intAry,(n)->{return n > 0;});

        //your code goes here...

    }
}
```

b) Given the following interface.

```
public interface ArrayAnalyzer {
    double getResult(double[] a);
}
```

Write a lambda expression to return maximum element in the given array.

```
public class MyArrayProg {
    public static void main(String[] args){
        new MyArrayProg().runApp();
    }
    /**
     * Print the result of the ArrayAnalyzer
     *
     * @param a - a double array with size > 0
     * @param analyzer - an ArrayAnalyzer
     */
    public void printResult(double[] a, ArrayAnalyzer analyzer){
        System.out.println(analyzer.getResult(a));
    }
    public void runApp(){
        double[] array = {3.5, 54, 76.8, 48, 9.7, 8, 7};
        //Example
        System.out.print("Total: ");
        printResult(array, (a)->{
            double total = 0;
            for (int i = 0; i < a.length; i++) {
                total += a[i];
            }
            return total;
        });

        //your code goes here...



    }
}
```

Part B Programming Exercises

Task 1: Movable Objects

Modify the given **`Rectangle.java`** to implement the **`Movable`** interface.

- Add two attributes for the **`x`** and **`y`** coordinates of the position of the rectangle.
- Add a constructor to accept the name of the rectangle, x and y coordinates and the width and length of the rectangle as arguments.
- Modify the **`toString`** methods to print the information as follows:
  `Rectangle` **`name`** `(`**`x`**`,` **`y`**`)`
  `Width:` **`width`**
  `Length:` **`length`**
- Implement the abstract methods specified in the **`Movable`** interface. In the **`moveLeft`** and **`moveRight`** methods, reduce and increase the **`x`** coordinate by the **`DX`** value respectively.

Task 2: Movable Collection

Modify the given **`MovableCollection.java`** as follows.

- Add the following rectangle into the **`movableList`**.

name	x	y	width	length
D	7	8	10	20
E	9	10	5	10

- Print the list.
- Move all the objects in the **`movableList`** to the right by the **`moveRight`** method and then print the list again.

Sample output:

```
Point A (1, 2)

Point B (3, 4)

Point C (5, 6)

Rectangle D (7, 8)
Width: 10.0
Length: 20.0

Rectangle E (9, 10)
Width: 5.0
Length: 10.0
```

```
After moving right...
Point A (6, 2)

Point B (8, 4)

Point C (10, 6)

Rectangle D (12, 8)
Width: 10.0
Length: 20.0

Rectangle E (14, 10)
Width: 5.0
Length: 10.0
```

Task 3: Resizable Objects

a) Write an interface called **`Resizable`**. The interface has a double constant **`DEFAULT_FACTOR`** with 1.5 as value and an abstract method **`resize`** that has no input and no return value. The purpose of the **`resize`** method is to modify the dimension of the object by a factor.

b) Modify the given **`Circle.java`** to implement the **`Resizable`** interface. The **`resize`** method updates the **`radius`** by the **`DEFAULT_FACTOR`**. That is, the new **`radius`** becomes **`radius * DERAULT_FACTOR`**.

c) Modify the **`Rectangle.java`** in Task 1 to implement both **`Movable`** and **`Resizable`** interfaces. The **`resize`** method updates the **`width`** and **`length`** of the rectangle by the **`DEFAULT_FACTOR`**.

Task 4: Resizable Collection

a) Write a class called **`ResizableCollection`** that creates the following **`Resizable`** objects and add them into a **`Resizable`** array list.

Rectangles:

name	x	y	width	length
D	7	8	10	20
E	9	10	5	10

Circles:

name	radius
F	5
G	15

b) Print the list. Then, call the **`resize`** method to resize all the objects in the list and print the list again.

Sample output:

```
Rectangle D (7, 8)
Width: 10.0
Length: 20.0

Rectangle E (9, 10)
Width: 5.0
Length: 10.0

Circle F
Radius: 5.0

Circle G
Radius: 15.0
```

```
After resize...
Rectangle D (7, 8)
Width: 15.0
Length: 30.0

Rectangle E (9, 10)
Width: 7.5
Length: 15.0

Circle F
Radius: 7.5

Circle G
Radius: 22.5
```

Task 5: Checkoutable Objects

We will implement a small component of the library system as the following:

- Books and Magazines can be checked out by implementing the following interface

```
import java.time.LocalDate;

public interface Checkoutable {

	void checkout();

	LocalDate returnDate();

}
```

  Refer to https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html for more information on using **`LocalDate`**.

- **`checkout()`** method should set today's date as checkout date and print the information for the object and today's date as checkout date.

- **`returnDate()`** method should return the return date for the object. For a **`Book`** object, return date should be 15 days from the checkout date. For a **`Magazine`**, return date should be 7 days from the checkout date.

a) Modify the **`Book.java`** and **`Magazine.java`** you have done in lab 12 to implement the **`Checkoutable`** interface. You may need to add an attribute to store the checkout date in the classes.

b) Write a driver program called **`CheckoutableCollection`** to test the functionality of your classes in the following manner – Create an array list of type **`Checkoutable`**. Add the following objects into the array list. Invoke the **`checkout()`** and **`returnDate()`** method on these objects.

Book:

Title	Publisher	Author
Cindy and the Candy Factory	AA Press	Ben Don
Secret Code	Ma House	Dim Green

Magazine:

Title	Publisher	Volume	Issue
Living	Person	5	3
Cooking	Person	3	10

KidsMagazine:

Title	Publisher	Volume	Issue	Age Range
Tinkering	Teens World	3	10	6-12
My Dream	Teens World	8	5	3-6

Sample output:

```
Title: Cindy and Candy Factory
Publisher: AA Press
Author: Ben Don
Checkout Date: 2021-11-25
Return Date: 2021-12-10

Title: Secret Code
Publisher: Ma House
Author: Dim Green
Checkout Date: 2021-11-25
Return Date: 2021-12-10

Title: Living
Publisher: Person
Volume: 5
Issue: 3
Checkout Date: 2021-11-25
Return Date: 2021-12-02

Title: Cooking
Publisher: Person
Volume: 3
Issue: 10
Checkout Date: 2021-11-25
Return Date: 2021-12-02

Title: Tinkering
Publisher: Teens World
Volume: 3
Issue: 10
Age Range: 6 - 12
Checkout Date: 2021-11-25
Return Date: 2021-12-02

Title: My Dream
Publisher: Teens World
Volume: 8
Issue: 5
Age Range: 3 - 6
Checkout Date: 2021-11-25
Return Date: 2021-12-02
```

Optional Exercises

Task 1: Take off the plane

In the Aeroplane chess, in order to take off a plane, a roll of 6 is needed. In the given **TakeOffPlane.java**, write a **recursive** method called **takeOff** to roll a dice (generate a number from 1 to 6) and return the number of rolls required to get a 6.

If you do not know how to play Aeroplane Chess, you may refer to https://youtu.be/7St_9-D9B_Q?t=19 (00:19-00:24)

```
public int takeOff() {

    if (…) {
        return 1;
    } else {
        return …;
    }
}
```

Sample outputs:

```
2
6
The plane takes off after 2 rolls.
```

Task 2: Landing the plane

In the Aeroplane chess, in order to land the plane, an exact roll is needed. In the given **LandingPlane.java**, write a **recursive** method called **landing** that accepts the number of remaining steps as input, rolls a dice (generate a number from 1 to 6) and returns the number of rolls required to land the plane.

Land the plane with an exact roll: https://youtu.be/7St_9-D9B_Q?t=643(1043-10:49)

If the roll value larger than remaining step https://youtu.be/7St_9-D9B_Q?t=708 (11:48-11:53)

```
public int landing(int remainingStep){
  …
}
```

Sample outputs:

```
Remaining Step: 5, Roll: 6
Now Remaining Step: 1
Remaining Step: 1, Roll: 1
The plane lands after 2 rolls.
```

Task 3: Bridge Crossing Game

There are 6 people. They spend 1, 2, 4, 6, 8, 12 seconds to cross the bridge respectively. Help then cross the bridge within 30 seconds. Up to 2 people can cross the bridge at the same time. People can only cross if one of them is holding the lamp. If you do not know how to play Bridge Crossing Game, you may refer to https://www.inwebson.com/demo/cross-the-bridge/

You need to complete the following **recursive** method so that it finds a path that can cross the bridge within 30 seconds.

Complete the given **BridgeCrossing.java**.

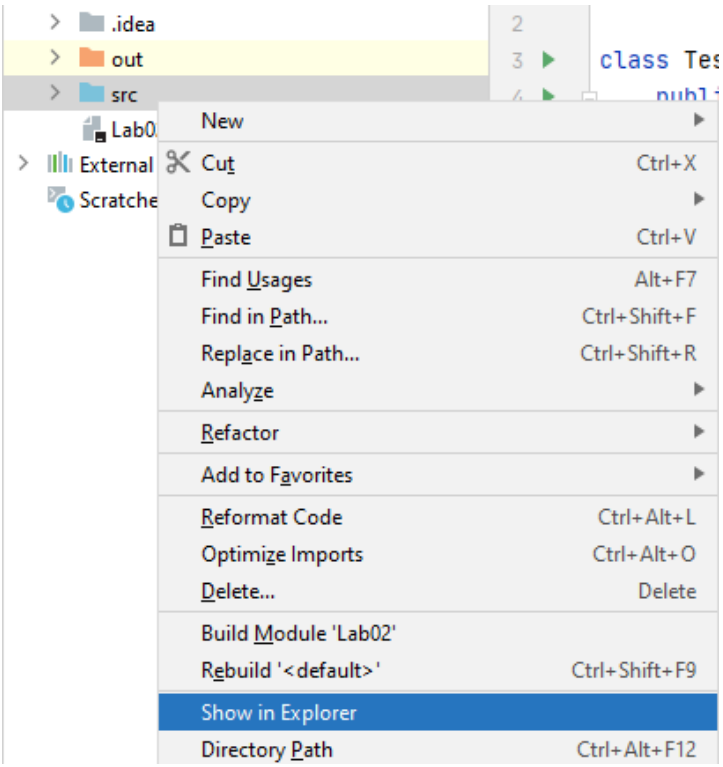
```
int crossing(boolean[] objects, int timeLimit) {
    //base case
    if (...) return 0;

    //copy the array to avoid its children mess-it up.
    ...
    //check the lamp position
    if (...){
        // if the lamp at the right
        // from right to left - always pick one only and go back.
        ...


    } else {
        // the lamp is at the left
        // from left to right - always pick two to go.
        ...

    }
}
```

Sample outputs:

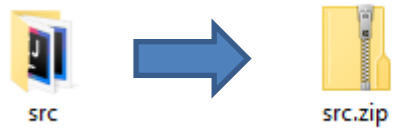
```
Let's cross some bridges!
1 2 >   time spent: 2 time remaining: 0
< 2     time spent: 2 time remaining: 2
8 12 >  time spent: 12 time remaining: 4
< 1     time spent: 1 time remaining: 16
1 2 >   time spent: 2 time remaining: 17
< 2     time spent: 2 time remaining: 19
4 6 >   time spent: 6 time remaining: 21
< 1     time spent: 1 time remaining: 27
1 2 >   time spent: 2 time remaining: 28
The log should be read in reverse order
Total steps required: 30
```

Part C Submitting Exercises

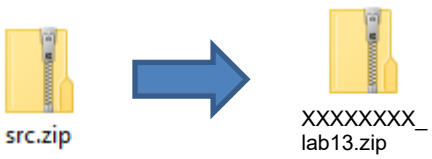
Step 1: Right-click the **src** folder and select **Show in Explorer**



Step 2: Zip the **src** folder into **src.zip**



Step 3: Rename the src.zip file to **XXXXXXXX_lab13.zip** where **XXXXXXXX** is your **student id**



Step 4: Submit **XXXXXXXX_lab13.zip** and **XXXXXXXX_lab13.docx** to Moodle.

References

[1] Bravaco, R., & Simonson, C. (2009). *Java programming: From the ground up*. *Dubuque*, IA: McGraw-Hill.

[2] Dean, J., & Dean, R. (2008). *Introduction to programming with Java: A problem solving approach*. Boston: McGraw-Hill.

[3] Farrell, J. (2012). *Java programming. Boston, MA: Course Technology Cengage Learning*

[4] Forouzan, B. A., & Gilberg, R. F. (2007). *Computer science: A structured programming approach using C (3rd ed.)*. Boston, MA: Thomson Course Technology.

[5] Gaddis, T. (2016). *Starting out with Java (6th ed.)*. Pearson.

[6] Liang, Y. D. (2013). *Introduction to Java programming: Comprehensive version*. (8th ed.). Pearson.

[7] Schildt, H. (2006). *Java a beginner's guide*. New York: McGraw Hill.

[8] Schildt, H., & Skrien, D. J. (2013). Java programming: A comprehensive introduction. New York: McGraw-Hill.

[9] Wu, C. T. (2010). *An introduction to object-oriented programming with Java*. Boston: McGraw Hill Higher Education

[10] Xavier, C. (2011). *Java programming: A practical approach*. New Delhi: Tata McGraw Hill.

[11] yet another insignificant Programming Notes. (n.d.). Retrieved from https://www3.ntu.edu.sg/home/ehchua/programming

[12] Zakhour, S., Kannan, S., & Gallardo, R. (2013). *The Java tutorial: A short course on the basics (5th ed.)*.

[13] Edpresso Team. (2019, July 1). What is a Java lambda function? Educative: Interactive Courses for Software Developers. https://www.educative.io/edpresso/what-is-a-java-lambda-function