UBC OKANAGAN

COSC 121
Computer Programming II

Midterm #2 - Review

Dr. Mostafa Mohamed

Acknowledgement: The slides mainly rely on the textbook of Y. D. Liang titled “Introduction to Java Programming, 10th Ed.”, Pearson Edu. Inc.

Midterm Format

Part 1: Multiple choice questions - 26 marks

Part 2: Code analysis questions - 10 marks

- **For example,**
  - What is the output of a given code?
  - Given some code, what will happen if...?
  - Add/change statements in order for your code to have a certain behavior
  - Big(O)

Part 3: Programming questions - 10 Marks

- **Short / Long coding-questions**

Recursion

- When to use recursion vs. iteration.
- How to use recursion (with or w/t recursive helper methods)
- How recursion works in the memory.
- What is tail-recursive methods?

Collections

Java Collections

- Collections vs. Arrays
- Using collection classes (and their methods)
  - ArrayList, LinkedList, Stack, Queue, PriorityQueue
- Performances – when to use which
- Intro to Generics
- Object Wrapper Classes *and* Autoboxing
  - int → Integer, double → Double, etc
  - Index or value of item when calling a remove(..)
- Useful methods from Collections and Arrays classes
  - Array ← to → List

    list= Arrays.asList(array) and list.toArray(array)
- Collections' sort(), max(), min(), shuffle()

Note

The list of examples in the following section is **NOT inclusive.**
In other words, do **not** just focus on these examples and assume the exam will be based on them.

Your preparation for the exam should include **ALL** exercises we worked on in this course (up to the cut-off lecture of this exam) such as those discussed in the lecture notes and in the lab assignments.

Analysis Questions: Recursion Example

What is the output? What does the program do?

Also, identify base cases and recursive calls.

```
public class Practice1 {
    public static void main(String[] args) {
        System.out.println("Sum: " + m1(5));
    }

    public static int m1(int n) {
        if (n == 1)
            return 1;
        else
            return n + m1(n - 1);
    }
}
```

o/p: 15

Analysis Questions: Recursion Example

Tail-recursive or not? Explain

(a)

```
public static int factorial(int n){
    if(n == 0)                          //stopping condition
        return 1;
    else
        return n * factorial(n-1);  //recursive call
}
```

(b)

```
public static boolean isPalindrome(String s) {
    if (s.length() <= 1)                                    // Base case1
        return true;
    if (s.charAt(0) != s.charAt(s.length() - 1))    // Base case2
        return false;
    return isPalindrome(s.substring(1, s.length() - 1));
}
```

(a) **Not tail** , because there is a pending operation, namely multiplication, to be performed on return from each recursive call.

(b) **Tail**. No pending operations after the recursive call.

Coding Questions: Recursion

We have seen many recursive methods in this course. I understand that some of you are having trouble with the "Tower of Hanoi" example – don't focus too much on this one as it is used to show you how recursive could significantly simplify your code as opposed to just using loop.

For this part, you could be asked to recursively do a certain task such as

- Traversing a file system.
- Processing a string or an array.
- Compute a series of numbers that have a certain pattern.
- Etc.

For example, write a recursive method m(int i) to compute the following series:

$$m(i) = \frac{1}{2} + \frac{2}{3} + \frac{3}{4} + \cdots + \frac{i}{i+1}$$

Coding Questions: Recursion

Write a method that computes Fibonacci number given its index.
Given: fib(i) = fib(i-1) + fib(i-2), fib(0)=0, fib(1)=1.

OR

Rewrite the following code using recursion:

```
// NON-recursive Fibonacci method
public static long fib(long index) {
    if (index == 0) return 0;
    if (index == 1) return 1;

    int f0 = 0, f1 = 1, currentFib = 1;
    for (int i = 2; i <= index; i++) {
        currentFib = f0 + f1;
        f0 = f1;
        f1 = currentFib;
    }
    return currentFib;
}
```

Solution

```
//recursive Fibonacci method
public static long fib(long index) {
    if (index == 0)          //Stopping condition 1 (Base case 1)
        return 0;
    else if (index == 1)     //Stopping condition 2 (Base case 2)
        return 1;
    else                     // Reduction and recursive calls
        return fib(index - 1) + fib(index - 2);
}
```

Priority Queue

Given the Patient class below, write a program for a hospital that will keep a waiting list of patients. Patients are usually sorted based on first-come-first-served. However, emergency cases should be treated first. Whenever you have two emergency cases, the case arrived first is treated first.

```
public class Patient {
    //attributes
    private int order;
    private String name;
    private boolean emergencyCase;

    //constructor
    public Patient(int order, String name, boolean emergencyCase) {

    //getters and setters
    public int getOrder() {
    public void setOrder(int order) {
    public String getName() {
    public void setName(String name) {
    public boolean isEmergencyCase() {
    public void setEmergencyCase(boolean emergencyCase) {

    public String toString() {
}
```

Solution

```
public class PatientComparator implements Comparator<Patient>{
    public int compare(Patient p1, Patient p2) {
        if(p1.isEmergencyCase() && !p2.isEmergencyCase())
            return -1;                              //place p1 first
        else if(!p1.isEmergencyCase() && p2.isEmergencyCase())
            return 1;                               //place p2 first
        else            //if both are emergency or both are not emergency
            return p1.getOrder()-p2.getOrder(); //place smaller order first
    }
}
```

```
public class PatientTest {
    public static void main(String[] args) {
        PriorityQueue<Patient> waitingList =
                new PriorityQueue<>(5, new PatientComparator());

        waitingList.offer(new Patient(1, "p1", false));
        waitingList.offer(new Patient(2, "p2", false));
        waitingList.offer(new Patient(3, "p3", true));
        waitingList.offer(new Patient(4, "p4", false));
        waitingList.offer(new Patient(5, "p5", true));

        while(waitingList.size()>0)
            System.out.println(waitingList.poll());
    }
}
```

Output

p3
p5
p1
p2
p4