COE 312 – Data Structures

Welcome to Exam I
Thursday October 25, 2018

Instructor: Dr. Wissam F. Fawaz

Name: ___Solution Key_____

Student ID: _____

Instructions:

1. This exam is **Closed Book**. Please do not forget to write your name and ID on the first page.
2. You have exactly **100 minutes** to complete the 4 required problems.
3. Read each problem carefully. If something appears ambiguous, please write your assumptions.
4. Points allocated to each problem are shown in square brackets.
5. Put your answers in the space provided only. No other spaces will be graded or even looked at.

Good Luck!!

Problem 1: Exception (**20 minutes**) [20 points]

1) Which of the following is not a **checked** exception?
   a. `IOException`
   b. **`NumberFormatException`**
   c. `Exception`
   d. Both (b) and (c)
2) Which of the following methods can be used to get the line number information associated with a given `StackTraceElement`?
   a. `getMessage()`
   b. `getNumber()`
   c. `getLine()`
   d. **None of the above**
3) What does the top row of the call stack trace corresponding to an exception occurrence represent?
   a. Detection point
   b. **The initial point (method) at which the exception occurred**
   c. An exception message describing what type of exception occurred
   d. None of the above
4) Which of the following static methods can be used to compare a floating-point value to `NaN`?
   a. `Integer.NaN`
   b. `Integer.isNaN`
   c. `Integer.isNumber`
   d. **None of the above**

For the next couple of questions (5, 6, and 7) consider the following code fragment. Assume that `double method1(int)` is a method that was properly created. Note that line numbers are placed at the beginning of each statement for ease of reference in the subsequent questions.

```
1. int value1 = 11;
2. double value2;
3. assert (value1 <= 0 || value1 >= 10);
4. value2 = method1(value1);
5. assert (value2 > 2) : "Invalid output";
```

5) Which of the following command line option can be used to enable assertions?
   a. `-enable`
   b. `-enableAssertion`
   c. `-ea`
   d. None of the above is a valid command line option for activating assertions
6) Which of the following statements about the code fragment above is **true**?
   a. The statement at line number 3 represents a post-condition
   b. The statement at line number 5 represents a pre-condition
   c. Both (a) and (b) are true
   d. **Both (a) and (b) are false**
7) Which of the following statements about the code fragment is **true**?
   a. The statement at line number 5 generates an exception if `value2` receives a value of `2` at line number 4.
   b. The statement at line number 3 generates an assertion error
   c. **Both (a) and (b) are false statements**
   d. Both (a) and (b) are true statements

8) What output does the following code fragment produce?

```
int[] arr = {1, 3, 4, 2};
int output = 1;
try {
      for(int i=0; i<=arr.length; i++)
            output *= arr[i];
} catch(IndexOutOfBoundsException e) {
      System.out.print("Done: ");
}
System.out.println(output);
```

   a. `Done:`
   b. `12`
   c. **`Done: 24`**
   d. None of the above

9) Consider the following method definition:

```
void foo() throw Exception {
      throws new Exception("Error in foo!");
}
```

   Which of the following is true about this definition?

   a. **This method definition does not compile correctly**
   b. The following method call: `foo();` produces an exception
   c. Both of the above are true
   d. None of the above is true

10) Which of the following methods is not defined in the `Exception` class?

   a. `getMessage()`
   b. `printStackTrace()`
   c. `getStackTrace()`
   d. **All of the above are defined in the `Exception` class**

11) Which of the following results in a compile-time error?

   a. `Throwable e = new NumberFormatException("Problem occurred");`
   b. **`NumberFormatExceptin e=new InputMismatchException("Error");`**
   c. Both of the above
   d. None of the above

12) Which of the following parent classes can be used to derive and create a new checked exception class?

   a. **`Exception`**
   b. `RunTimeException`
   c. Both of the above
   d. None of the above

13) What output does the following code fragment produce?

```
try {
      System.out.println(10/0.0);
} catch(Exception e) {
      System.out.println("Arithmetic exception occurred.");
}
```

   a. `Arithmetic exception occurred.`
   b. `NaN`
   c. The compiler generates a compile-time error since Java does allow division by zero with floating point values.
   d. **None of the above**

Problem 2: Polymorphism and IO (**20 minutes**) [20 points]

1) Which of the following methods is not defined in the `java.io.File` class?
   a. **`exist()`**
   b. `lastModified()`
   c. `list()`
   d. All of the above methods are part of the `File` class

Consider the following code fragment for the next couple of questions. Note that line numbers are placed at the beginning of each statement for ease of reference in the subsequent questions.

```
1. File srcFile = new File("In.jpg");
2. File trgtFile = new File("Out.jpg");
3. FileInputStream fis=new FileInputStream(srcFile);
4. FileOutputStream fos=new FileOutputStream(trgtFile);
5. byte[] buffer = new byte[(int) srcFile.length()];
6. int len;
7. while((len = fis.read(buffer)) > 0)
8.       fos.write(buffer, 0, len);
9. fos.close(); fis.close();
```

2) What does the above code fragment do?
   a. It copies some of the content of `srcFile` into `trgtFile`
   b. It copies all of the bytes except one byte from `srcFile` into `trgtFile`
   c. **It copies the content of `srcFile` in its entirety to `trgtFile`**
   d. None of the above describes accurately what the code fragment does
3) Assuming that the above code is placed inside the body of a `main` method, which of the following headers can be used for that `main` method?
   a. `public static void main(String[] args) throws IOException`
   b. `public static void main(String[] args) throws Exception`
   c. **Both of the above**
   d. None of the above
4) How many iterations are performed by the while loop present at line number 7 if the source file `srcFile` has a non-zero size?
   a. `2`
   b. **`1`**
   c. It is not possible to determine the number of iterations as it depends on the size of `srcFile`
   d. None of the above
5) What is the maximum number of bytes that can be written by the statement at line number 8 to `trgtFile`?
   a. `(len+1) bytes`
   b. **`len bytes`**
   c. `(len-1) bytes`
   d. None of the above
6) Given that the `FileInputStream` class is derived from the class `InputStream` and that the `FileOutputStream` class is derived from `OutputStream`, which of the following method headers can be used for a method called `foo` that can receive both `FileInputStream` and `FileOutputStream` objects as parameters?
   a. `public void foo(InputStream obj)`
   b. `public void foo(OutputStream obj)`
   c. **`public void foo(Object obj)`**
   d. None of the above

For the following questions, consider a class hierarchy rooted at an abstract superclass called `StaffMember`. Assume that `StaffMember` has two direct child classes, namely the `Volunteer` and `Employee` subclasses. Suppose further that Employee has two direct child classes called `Executive` and `Hourly`, respectively. `StaffMember` defines an abstract method called pay, which is overridden by all of its descendant classes. Moreover, `Executive` defines a method called `awardBonus(double)` and `Hourly` implements a method called `addHours(int)`. Note finally that all classes of the considered hierarchy have default constructors; that is, constructors taking `0` parameters.

7) Given the above class hierarchy, which of the following statements results in a compile-time error?
   a. `StaffMember member = new StaffMember();`
   b. `Hourly hourly = new Hourly();`
      `hourly.addHours(23.5);`
   c. **Both of the above**
   d. None of the above
8) Consider the following array:
   `Employee[] employees = {new Executive(), new Hourly()};`
   Which of the following statements generates an exception?
   a. `employees[2].pay();`
   b. `((Executive) employees[1]).awardBonus(24.7);`
   c. **Both of the above**
   d. None of the above
9) Given the `employees` array defined in the previous question, which of the following `for` loops correctly pays all the elements of that array?
   a. `for(StaffMember staff : employees)`
      `staff.pay();`
   b. `for(Employee employee : employees)`
      `employee.pay();`
   c. **Both of the above**
   d. None of the above
10) Which of the following correctly captures the definition of the constructor of the `Volunteer` class?
   a. `public void Volunteer() { }`
   b. `public void Volunteer() { super(); }`
   c. Both of the above
   d. **None of the above**
11) Assume that `StaffMember` has a `toString()` method that is not overridden by the `Volunteer` class. In light of this assumption, which of the following `toString()` method definitions is equivalent to the `toString()` as defined in the `Volunteer` class?
   a. `public String toString() { super.toString(); }`
   b. `public String toString() { super(); }`
   c. Both of the above
   d. **None of the above**
12) Consider the following statement: `StaffMember member = new Hourly();`
   Which of the following correctly calls the `addHours` method as defined in `Hourly`?
   a. `member.addHours(12);`
   b. `((Employee) member).addHours(12);`
   c. **`((Hourly) member).addHours(12);`**
   d. Both (b) and (c)

Problem 3: I/O streams (**30 minutes**) [30 points]

You are given a text files called "In.txt". Moreover, you are given a binary file called "In.jpg". Note that "In.txt" consists of 4 paragraphs that are separated by a blank line. That is, a single blank line is used to separate any two consecutive paragraphs in "In.txt". It is important to highlight in this regard that a blank line does not contain any characters and as such it is equivalent to an empty string.

In light of the above, you are tasked with designing and implementing a Java program that:

1. Modifies the content of "In.txt" as follows. Your Java program should re-order the paragraphs of "In.txt" such that the 4th paragraph of the original file appears first, followed by the 1st paragraph of the original file, then the 3rd paragraph, and finally the 2nd paragraph of the original file should appear last in the modified version of the file.
2. Then, your program should process "In.jpg" as follows. The first half of "In.jpg" should be exchanged with its second half in the sense that your program should swap the contents of the first and second halves of the "In.jpg" file.

Assume in both cases that the source files belong to the same directory as your Java project.

```
import java.io.*;
public class FileManagement {
      public static void main(String[] args) throws IOException {
            File txtFile = new File("In.txt");
            File jpgFile = new File("In.jpg");

            FileWriter fw = new FileWriter(txtFile);
            BufferedWriter bw = new BufferedWriter(fw);
            PrintWriter outToFile = new PrintWriter(bw);
            Scanner fileScan = new Scanner(txtFile);

            StringBuilder[] pgs = new StringBuilder[4];
            for(int i=0; i<pgs.length; i++)
                  pgs[i] = new StringBuilder();

            int currentPg = 0; String line;
            while(fileScan.hasNextLine()) {
                  line = fileScan.nextLine();
                  if(line.equals(""))
                        currentPg++;
                  else
                        pgs[currentPg].append(line + "\n");
            }
            outToFile.println(pgs[3]);
            outToFile.println(pgs[0]);
            outToFile.println(pgs[2]);
            outToFile.println(pgs[1]);
            outToFile.close();
            fileScan.close();

            FileInputStream fis = new FileInputStream(jpgFile);
            FileOutputStream fos = new FileOutputStream(jpgFile);
            byte[] firstHalf = new byte[(int) jpgFile.length()/2];
```

```
            byte[] secondHalf = new byte((int) (jpgFile.length() -
            jpgFile.length()/2)];
            fis.read(firstHalf);
            fis.read(secondHalf);
            fos.write(secondHalf, 0, secondHalf.length);
            fos.write(firstHalf, 0, firstHalf.length);
            fos.close();
            fis.close();
      }
}
```

Problem 4: XML, JSON and Timer (**30 minutes)** [30 points]

Consider the following XML-formatted and JSON-formatted documents.

```
<?xml version="1.0"?>
<timers>
<timer>
<id>1</id>
<delay>2</delay>
</timer>
<timer>
<id>2</id>
<delay>2</delay>
</timer>
<timer>
<id>3</id>
<delay>1</delay>
</timer>
<timer>
<id>4</id>
<delay>1</delay>
</timer>
<timer>
<id>5</id>
<delay>2</delay>
</timer>
<timers>
```

```
[{
"Decrement":2,
"StartingValue":19
},
{
"Decrement":5,
"StartingValue":15
},
{
"Decrement":4,
"StartingValue":10
},
{
"Decrement":2,
"StartingValue":14
},
{
"Decrement":4,
"StartingValue":14
}]
```

The information given above is stored in two separate online **files** called, respectively, "timers.xml" and "timers.json", with both residing inside a **folder** named "Timer" on the "http://www.wissamfawaz.com" **webserver**. So for instance, "timers.xml" can be referenced through the following URL: "http://www.wissamfawaz.com/Timer/timers.xml".

1. Write a Java program that manages a bunch of timer objects as illustrated below. Each one of the managed timer instances will independently perform a count-down from some **starting value** until reaching a value that is below zero. This is achieved by decrementing a running counter by a certain **decrement** value every **delay** seconds. Note that each individual timer has an unique identifier called **id** that differentiates it from the remaining counters. In particular, your application should accomplish the following tasks:
    a. It should retrieve the **id**, **delay**, **decrement**, and **startingValue** attributes from the online files given above,
    b. Your program should then start multiple simultaneous count-down instances having an id, delay, and starting values matching the one retrieved from the online files,
    c. Lastly, your program should make sure to:
        1. stop all timer instances once they completed their associated count-down operation.
        2. Compile a list of the ids of those timer objects that completed their count-down operations **first** as well as another list of those timer objects that finalized their count-down operations **last**.

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.InputStream;
import java.io.PrintWriter;
```

```
import java.net.URL;
import java.util.ArrayList;
import javax.swing.Timer;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.json.JSONArray;
import org.json.JSONObject;
import org.json.JSONTokener;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
public class TimerManager {
      private Timer[] timers; private int[] counters;
      private ArrayList<Integer> idsAL, delaysAL, decrementsAL,
                                 startingValsAL;
      public TimerManager() {
            timers = null; counters = null;
            idsAL = new ArrayList<>();
            delaysAL = new ArrayList<>();
            decrementsAL = new ArrayList<>();
            startingValsAL = new ArrayList<>();}
      public static void main(String[] args) throws Exception {
            TimerManager manager = new TimerManager();
            manager.retrieveInfo();
            manager.startTimers();
            manager.results();
      } // End of main
      public void retrieveInfo() throws Exception {
            URL url1 =
            new URL("http://www.wissamfawaz.com/Timer/timers.xml");
            URL url2 =
            new URL("http://www.wissamfawaz.com/Timer/timers.json");
            InputStream is1 = url1.openStream(), is2=url2.openStream();
            DocumentBuilderFactory dbf =
            DocumentBuilderFactory.newInstance();
            DocumentBuilder db = dbf.newDocumentBuilder();
            Document doc = db.parse(is1);
            NodeList idsNodeList = doc.getElementsByTagName("id");
            NodeList delaysNodeList =
            doc.getElementsByTagName("delay");
            Node idNode, delayNode;
            for(int i=0; i<idsNodeList.getLength(); i++) {
                  idNode = idsNodeList.item(i);
                  idsAL.add(Integer.parseInt(idNode.getTextContent()));
                  delayNode = delaysNodeList.item(i);
                  delaysAL.add(
                  Integer.parseInt(delayNode.getTextContent()));
            }
            int nbOfTimers = idsNodeList.getLength();
            Timers = new Timer[nbOfTimers];
            counters = new int[nbOfTimers];
            JSONTokener tokener = new JSONTokener(is2);
            JSONArray mainArr = new JSONArray(tokener);
            JSONObject obj; int decrement, startingVal;
            for(int i=0; i<mainArr.length(); i++) {
                  obj = mainArr.getJSONObject(i);
                  decrement = obj.getInt("Decrement");
```

```
            startingVal = obj.getInt("StartingValue");
            decrementsAL.add(decrement);
            startingVals.add(startingValue);
      }
      startTimers();
} // End of retrieveInfo
private void startTimers() {
      int delayInt;
      for(int i=0; i<timers.length(); i++) {
            delayInt = delaysAL.get(i);
            counters[i] = startingValsAL.get(i);
            timers[i] = new Timer(delayInt*1000, new
            CountDownHandler(i));
            timers[i].start();
      }
} // End of startTimers
private class CounrDownHandler implements ActionListener {
      private int timerIndex;
      public CountDownHandler(int index) {
            timerIndex = index;
      }
      public void actionPerformed(ActionEvent e) {
            if(counter[timerIndex]>0)
                  counter[timerIndex] -=
                  decrementsAL.get(timerIndex);
      }
} // End of CountDownHandler
private void results() throws Exception {
      ArrayList<Integer> indexesRunningTimers =
      new ArrayList<>();
      ArrayList<Integer> indexesFinishedTimers =
      new ArrayList<>();
      ArrayList<Integer> idsWinningTimers =
      new ArrayList<>();
      ArrayList<Integer> idsLosingTimers =
      new ArrayList<>();
      boolean winnersIdentified = false;
      while(true) {
            if(indexesRunningTimers.isEmpty())
                  break;
            for(int timerIndex : indexesRunningTimers) {
                  if(counters[timerIndex] < 0) {
                        indexesFinishedTimers.add(timerIndex);
                        timers[timerIndex].stop();
                        if(!winnersIdentified)
                              idsWinningTimers.add(
                              idsAL.get(timerIndex));
                  }
                  if(!idsWinningTimer.isEmpty()&&
                  !winnersIdentified)
                        winnersIdentified = true;
            }
            if(indexesFinishedTimers.size() ==
            indexesRunningTimers.size()) {
                  for(int index : indexesFinishedTimers)
                        idsLosingTimers.add(
                        idsAL.get(index));
```

```
                }
                for(int timerIndex : indexesFinishedTimers)
                      indexesRunningTimers.remove(
                      new Integer(timerIndex));
                indexesFinishedTimers.clear();
          }
          System.out.println("Winners: " + idsWinningTimers);
          System.out.println("Losers: " + idsLosingTimers);
    } // End of results
```

```
}// End of TimerManager class
```

Appendix: Classes and Methods

1. XML parsing related classes along with their associated methods:

DocumentBuilderFactory
- static DocumentBuilderFactory newInstance()
- DocumentBuilder newDocumentBuilder()

Document
- NodeList getElementsByTagName(String)

Element
- NodeList getElementsByTagName(String)

DocumentBuilder
- Document parse(InputStream)

NodeList
- int getLength()
- Node item(int index)

Node
- String getTextContent()
- NodeList getChildNodes()

2. Classes of the `java.io` package:

java.io.File
- File(String)
- long length()

java.io.FileOutputStream
- FileOutputStream(File)
- void write(byte[] buff, int off, int len)
- void close()

java.io.FileInputStream
- FileInputStream(File)
- int read(byte[])
- void close()

java.io.FileWriter
- FileWriter(File)

java.io.PrintWriter
- PrintWriter(BufferedWriter)
- void print(String)
- void println(String)
- void close()

java.io.InputStreamReader
- InputStreamReader(InputStream)

java.io.BufferedReader
- BufferedReader(InputStreamReader)
- String readLine()

3. `Scanner` and `ArrayList` and some of their related methods.

java.util.Scanner	**java.util.ArrayList<T>**
Scanner(File)	ArrayList<>()
String nextLine()	int size()
boolean hasNext()	T get()
boolean hasNextLine()	void add(T)
String next()	boolean contains(T)
int nextInt()	int indexOf(T)
double nextDouble()	

4. JSON parser classes and some of their related methods.

JSONArray	**JSONObject**
JSONArray(JSONTokener)	JSONObject(JSONTokener)
int length()	JSONArray getJSONArray(String key)
JSONObject getJSONObject(int i)	JSONObject getJSONObject(String key)
JSONArray getJSONArray(int i)	String getString(String key)
double getDouble(int i)	double getDouble(String key)
int getInt(int i)	int getInt(String key)
String getString(int i)	

5. Timer class and ActionListener interface and some of their associated methods.

Timer	**ActionListener**
void start()	void actionPerformed(ActionEvent e)
void stop()	
boolean isRunning()	