COE 312 – Data Structures

Welcome to Exam I
Monday October 24, 2016

Instructor: Dr. Wissam F. Fawaz

Name: _____

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 **55 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: JDOM, DOM, and JSON (**10 minutes**) [20 points]

1) To create an instance of the `DocumentBuilderFactory` class, a factory method should be used. Which of the following classes of the classical `DOM` Java library are instantiated **using a factory method** as well?
   - **a. `DocumentBuilder`**
   - b. `Document`
   - c. Both of the above
   - d. None of the above

2) Which of the following is **false** about the following code fragment assuming that it executes correctly?

```
URL url = new URL("http://www.wissamfawaz.com/categories.json");
JSONTokener tokener = new JSONTokener(url.openStream());
JSONArray mainArray = new JSONArray(tokener);
```

   - a. `url.openStream()` returns an `InputStream`
   - **b. The `categories.json` network file begins with the `{` (curly brace) symbol**
   - c. Both of the above statements are false
   - d. (a) and (b) are both true

3) Which of the following classes is **not** part of `JDOM`?
   - a. `Element`
   - **b. `Node`**
   - c. `Attribute`
   - d. None of the above

4) Which of the following is **false** about the `Node` and `Element` classes of the classical `DOM` Java library?
   - a. Not all `Node` objects can have children
   - b. All `Element` objects may have attributes associated with them
   - c. An xml node's attribute is represented using a `Node` object
   - **d. None of the above statements is false**

5) How many **`JSON` objects** are present in the following `JSON-formatted` file content?

```
{"key1":[{"k1":"v1"}, {"k2":"v2"}], "key2":"value2"}
```

   - a. `1`
   - b. `2`
   - **c. `3`**
   - d. None of the above

6) How many **`JSON` arrays** are present in the `JSON-formatted` file content from the previous problem (5)?
   - **a. `1`**
   - b. `2`
   - c. `3`
   - d. None of the above

7) Which of the following statements is **false** about the `NodeList` class of the classical `DOM` Java library?
   a. An element at index `i` in the `NodeList` can be obtained via the `item` method
   b. The size of the `NodeList` can be obtained via the `length()` method
   c. A `NodeList` can be created using the `getElementsByTagName` of the `Document` class
   d. None of the above is false

8) Which of the following methods can be used to obtain **the size** of a `List` created using the `getChildren` method of `JDOM`?
   a. `size()`
   b. `length()`
   c. `getLength()`
   d. None of the above

9) In the classical `DOM` Java library, if `staffNode` is a `Node` object **representing** an `Element`, then `staffNode.getNodeType()` would return
   a. `Node.ELEMENT_NODE`
   b. `Element.ELEMENT_NODE`
   c. `Node.NODE_ELEMENT`
   d. None of the above

10) Which of the following is **not defined** in `JDOM`?
   a. `getChildText`
   b. `getTextContent`
   c. Both of the above
   d. None of the above

Problem 2: Exception, polymorphism and IO (**10 minutes**) [20 points]

1) Which of the following statements **does not** throw an `ArithmeticException`?
   a. `System.out.println(10/0.0);`
   b. `System.out.println(0.0/0.0);`
   c. Both of the above
   d. None of the above

2) Which of the following correctly represents the header of the method that a `FileFilter` uses to test whether or not a file **should be shown** to the user?
   a. `boolean accept(Path pathName)`
   b. `boolean accept(File filename)`
   c. Both of the above
   d. None of the above

3) If `Guitar` is derived from `Instrument` and `ElectricGuitar` extends `Guitar`, then which of the following references is **not polymorphic**?
   a. `Instrument i;`
   b. `Guitar g;`
   c. Both of the above
   d. None of the above

4) Which of the following statements can be used to create an object that can perform **byte-based input** from a webpage represented by a URL object called `url`?
   a. `BufferedInputStream bis = new BufferedInputStream(new InputStreamReader(url.openStream()));`
   b. `BufferedInputStream bis = new BufferedInputStream(url.openStream);`
   c. Both of the above
   d. None of the above

5) Consider a method called `sum` that was **overloaded twice**. In the following code fragment that uses both versions of `sum`, when does method binding happen?
   `int val1 = sum(2, 3); int val2 = sum(2);`
   a. At run-time
   b. At compile-time
   c. Either of the above
   d. None of the above

6) Consider the following code fragment.
```
public class NewException extends Exception {
   public NewException(String msg){super(msg);}}
```
   What does the call to `super` **inside the constructor** do?
   a. It sets up the print stack trace
   b. It customizes the error message
   c. It calls `super` as defined in the `Throwable` class
   d. None of the above

7) Consider a `Speaker` interface along with two classes called `Philosopher` and `Politician` that implement `Speaker`. Assume that `Speaker` contains only one

method called `speak` that has the following signature: `public void speak()`. Suppose also that both `Philosopher` and `Politician` have default constructors. **What happens** if the following code fragment is executed?

```
Speaker speaker = new Philosopher();
int val = speaker.speak();
speaker = new Politician();
val = speaker.speak();
```

a. This code will result in a compile-time error
b. This code will result in a run-time error
c. The `speak` method as defined in the `Philosopher` class will be correctly called first and then the `speak` method as defined in the `Politician` class will be correctly called
d. None of the above will happen

8) Assigning a subclass reference to a superclass variable **is safe** because
   a. the subclass object is an object of its superclass
   b. the subclass object is related to its superclass by inheritance
   c. Both of the above
   d. None of the above

9) Suppose that the class `Rodent` has a child class called `Rat` and another child class called `Mouse`. Assume also that the class `Mouse` has a child class called `PocketMouse`. Examine the following:

```
Rodent rod;
Rat rat = new Rat();
Mouse mouse = new Mouse();
PocketMouse pkt = new PocketMouse();
```

Which of the following will cause a **compile-time** error to occur?
   a. `rod = rat;`
   b. `rod = pkt;`
   c. `pkt = rat;`
   d. None of the above

10) Say that the situation is the same as described in the previous question. Which of the following statements correctly creates an array list that can hold `Rat` objects?
   a. `ArrayList<Object> list = new ArrayList<>();`
   b. `ArrayList<Rodent> list = new ArrayList<>();`
   c. Both of the above
   d. None of the above

Problem 3: I/O streams (**15 minutes**) [30 points]

You are given a binary file called "srcFile.jpg" and a text file called "srcFile.txt". Design and implement a Java program that:

1. copies the **first half of** "srcFile.txt" into a target file called "trgtFile.txt", and
2. then copies the **second half of** "srcFile.jpg" into a target file called "trgtFile.jpg".

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

```
import java.io.*;
import java.util.Scanner;
public class ProblemIII {
public static void main(String[] args) throws IOException {
	File srcFile = new File("srcFile.txt");
	File trgtFile = new File("trgtFile.txt");
	FileWriter fw = new FileWriter(trgtFile);
	BufferedWriter bw = new BufferedWriter(fw);
	PrintWriter outToFile = new PrintWriter(bw);
	Scanner fileScan1 = new Scanner(srcFile);
	Scanner fileScan2 = new Scanner(srcFile);
	int lineCount = 0;
	String line;
	while(fileScan1.hasNext()) {
		line = fileScan1.nextLine();
		lineCount++;
	}
	if(lineCount % 2 != 0)
		lineCount+=1;

	for(int i=1; i<=lineCount/2; i++) {
		line = fileScan2.nextLine();
		outToFile.println(line);
	}
	outToFile.close();

	File srcImg = new File("MyPhoto.jpg");
	File trgtImg = new File("TrgtPhoto.jpg");

	FileInputStream fis1 = new FileInputStream(srcImg);
	FileInputStream fis2 = new FileInputStream(srcImg);
	FileOutputStream fos = new FileOutputStream(trgtImg);
	int bytes = 0;
	int len;
	byte[] buffer1 = new byte[1024];
	while((len = fis1.read(buffer1)) > 0) {
		bytes+=len;
	}
	byte[] buffer3 = new byte[bytes/2];
	byte[] buffer4 = new byte[bytes - bytes/2];
	fis2.read(buffer3);
	len = fis2.read(buffer4);
	fos.write(buffer4, 0, len);

	fos.close();

	}
}
```

Problem 4: JDOM (**20 minutes)** [30 points]

Consider the following XML-formatted document that shows a list of words along with their phishing related score values:

```
<phishing>
<word>
<properties>
<value>label</value>
<score>15</score>
</properties>
</word>
<word>
<properties>
<value>invoice</value>
<score>13</score>
</properties>
</word>
<word>
<properties>
<value>post</value>
<score>11</score>
</properties>
</word>
<word>
<properties>
<value>document</value>
<score>10</score>
</properties>
</word>
<word>
<properties>
<value>postal</value>
<score>9</score>
</properties>
</word>
</phishing>
```

The information given above is stored in an online **XML file** called "phishing.xml" that resides inside a **folder** named "XML" on the "http://www.wissamfawaz.com" **webserver**. So, "phishing.xml" can be referenced through the following URL: "http://www.wissamfawaz.com/XML/phishing.xml". This xml file might be used in a phishing scanner tool to determine whether or not a given file contains a fraudulent message.

1. Using the **JDOM parser**, write a Java program that:
    a. retrieves all the words contained in this online xml file as well as their associated score values, and then
    b. outputs the words having the highest, lowest, and average score value. If you don't find words with scores matching the average, then select words whose scores are **the closest** to the integer portion of the average.

```
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;
import java.util.ArrayList;
```

```
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.filter.ElementFilter;
import org.jdom2.input.SAXBuilder;
import org.jdom2.util.IteratorIterable;
public class PhishingScanner {
	public static void main(String[] args) throws Exception {
		URL phishingURL = new
		URL("http://www.wissamfawaz.com/phishing.xml");
		ArrayList<String> words = new ArrayList<>();
		ArrayList<Integer> scores = new ArrayList<>();
		InputStream phishingAsIS = phishingURL.openStream();
		SAXBuilder builder = new SAXBuilder();
		Document document = builder.build(phishingAsIS);
		IteratorIterable<Element> wordsIterator =
			document.getDescendants(new ElementFilter("value"));
		IteratorIterable<Element> scoresIterator =
			document.getDescendants(new ElementFilter("score"));

		int min=Integer.MAX_VALUE, max = Integer.MIN_VALUE;
		double average = 0;
		for(Element wordElt : wordsIterator) {
			words.add(wordElt.getText());}
		for(Element scoreElt:scoresIterator) {
			int score = Integer.parseInt(scoreElt.getText());
			scores.add(score);
			if(score < min)
				min = score;
			if(score > max)
				max = score;
			average += score;
		}
		average /= scores.size();

		ArrayList<String> wordsWithMinScore = new ArrayList<>();
		ArrayList<String> wordsWithMaxScore = new ArrayList<>();
		ArrayList<String> wordsWithAvgScore = new ArrayList<>();
		int[] differences = new int[scores.size()];
		int index = 0;
		for(int score : scores) {
			differences[index] = score - (int) average;
			index++;
		}
		Arrays.sort(differences);
		for(int i=0; i<scores.size(); i++) {
			int score = scores.get(i);
			if(score == min)
				wordsWithMinScore.add(words.get(i));
			if(score == max)
				wordsWithMaxScore.add(words.get(i));
			if((score-(int)average) == differences[0])
				wordsWithAvgScore.add(words.get(i));}

System.out.println("Words with min score: " + wordsWithMinScore);
System.out.println("Words with max score: " + wordsWithMaxScore);
System.out.println("Words with avg score: " + wordsWithAvgScore);}}
```

Appendix: Classes and Methods

1. JDOM parser related classes along with their associated methods:

org.jdom2.Document • Element getRootElement() **org.jdom2.Element** • List<Element> getChildren(String) • IteratorIterable<Element> getDescendants(ElementFilter) • String getText() • String getChildText(String)	**org.jdom2.filter.ElementFilter** • ElementFilter(String) **org.jdom2.util.IteratorIterable** • boolean hasNext() • Element next() **java.util.List<Element>** • int size() • Element get(int index)

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. Classes of the `java.nio` package:

java.nio.file.Files • static BufferedReader newBufferedReader(Path) • static BufferedWriter newBufferedWriter(Path) • static InputStream newInputStream(Path) • static Path copy(Path src, Path trgt) • static void write(Path trgt, byte[] buff)	**java.nio.file.Paths** • static Path get(String) **java.nio.file.Path** • Path getFileName() • File toFile()

4. `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()
String next()	void add(T)
int nextInt()	boolean contains(T)
double nextDouble()	int indexOf(T)