Practice Sheet for COSC 121 Midterm2

Multiple choice questions:

1. What happens when a recursive method without a stopping condition is called?

   (a) The program does not terminate.

   (b) A `StackOverflowError` is thrown.

   (c) A `VirtualMachineError` is thrown.

   (d) The program terminates because Java automatically adds a stopping condition.

2. What is the relationship between text I/O and binary I/O?

   (a) Text I/O works with text files, while binary I/O works with any kind of file.

   (b) Text I/O classes inherit binary I/O classes.

   (c) Text I/O provides a layer of abstraction for text input/output over binary I/O.

   (d) None; they provide completely different functionalities.

3. When is the `finally` block executed in a `try-catch-finally` block?

   (a) When the execution flow exits the `try-catch-finally` block.

   (b) Before the `catch` blocks are checked.

   (c) At the end of the program.

   (d) Only when the `Exception` object is thrown to the caller method.

4. What is the complexity of the `get(int: index): E` method of a `LinkedList<E>` in terms of list size n?

   (a) $O(1)$.

   (b) $O(\log n)$.

   (c) $O(n^2)$.

   (d) $O(n)$.

5. ~~Programmers must be careful not to write inaccessible `catch` blocks when handling exceptions. Which pillar of OOP is expressed here?~~

   ~~(a) Polymorphism.~~

   ~~(b) Encapsulation.~~

   ~~(c) Inheritance.~~

   ~~(d) None.~~

Questions 6 to 10 refer to the following method:

```
public static int[] findWord(File file, String word) {
  int[] pos = new int[10000];
  int currentWord = 1;
  int currentPosInArray = 0;
  try (Scanner in = new Scanner(file)) {
    while(in.hasNext()) {
      if (in.next().equals(word)) {
        pos[currentPosInArray++] = currentWord;
      }
    currentWord++;
    }
  } catch (FileNotFoundException ex) {System.out.println("Cannot
find file"); return null;
  } catch (EOFException ex) {System.out.println("Read past end
of file"); return null;
```

```
  } catch (IOException ex) {System.out.println("Internal error");
return null;}
  return pos;
}
```

6. The `Scanner` class can be used to read files, but it might be too slow. Which of the following classes can be used to make reading the file faster?

   (a) `PrintWriter`

   (b) `BufferedReader`

   (c) `FileReader`

   (d) `InputStreamReader`

7. A programmer wants to put the `catch(IOException ex)` block in front of the other two `catch` blocks, but when doing so, the compiler shows an error. Why is this the case?

   (a) Some `catch` blocks will be inaccessible.

   (b) `IOException` is always thrown to the caller method.

   (c) The `finally` block is missing.

   (d) The `catch(FileNotFoundException ex)` block must be the first `catch` block here.

8. This method might not be robust in cases where there are a lot of words in the input file. What should be changed to fix this problem?

   (a) Make the size of `pos` larger.

   (b) Throw an exception when the size of `file` is too large.

   (c) Change the type of `pos` and the return type to `double[]`.

   (d) Change the type of `pos` and the return type to `ArrayList<Integer>`.

9. There is not a line of code that closes `in` explicitly in the method. Which of the following is the best reason why this is acceptable?

   (a) The JVM closes `in` when the program terminates.

   (b) The JVM closes `in` when the method returns to the caller.

   (c) `in` is closed when the flow exits the `try-catch` block.

   (d) Java automatically adds `in.close();` at the end of the method.

10. This method performs its task using text I/O, but binary I/O can also be used. Which of the following is a potential problem when adapting the method to binary I/O?

    (a) If input file includes both numeric and text items, it won't work (for example, when writeDouble is used to write to the file, you must use readDouble, otherwise it will be erroneous.

    (b) Binary I/O is slower than text I/O.

    (c) Binary I/O is not suitable for large files.

    (d) Java does not have built-in classes that allow binary input to be decoded.

    Questions 11-15 refer to the following method:

```
public static void printIterate(Iterable<Object> list) {
  Iterator<Object> it = list.iterator();
  while(it.hasNext()) {
    System.out.println(it.next());
    it.remove();
  }
}
```

11. We pass an `ArrayList` of `ArrayList` elements to the method. The large `ArrayList` has length n, and the small `ArrayList`s also have length n. As written, what is the complexity of the method? Assume that iteration is used in the `toString()` method.

    (a) $O(n)$

    (b) $O(n^2)$

    (c) $O(n^3)$

    (d) $O(1)$

12. Does the method destroy the `list` passed to it, and why?

    (a) No, `it.remove()` removes elements from another list.

    (b) No, `list` points to a different object from the argument.

    (c) Yes, `it.remove()` removes elements from the passed list because `it` is an `Iterator` from `list`.

    (d) Yes, `Iterator` always removes elements from its underlying list.

13. Is it possible for the method to accept arguments not of the type `List`?

    (a) Yes, there are other classes that implement `Iterable` that do not implement `List`.

    (b) Yes, but there will be an error because `iterator()` is only visible to `List`.

    (c) No, `List` is a subtype of `Iterable`.

    (d) No, it is not possible to iterate over `Collection`s that are not `List`s.

14. Elements in `list` are processed with `System.out.println(it.next());`.
    What is the result if we replace it with `System.out.println(it.next().compareTo(Integer`

    (a) The console prints out all 1s.

    (b) The console prints out all -1s.

    (c) There will be a runtime error.

    (d) There will be a compilation error.

15. We pass a `PriorityQueue` to the method. What is the order of traversal in this method?

    (a) Ascending order.

    (b) Descending order.

    (c) Unknown order.

    (d) Order in which elements are added to the `PriorityQueue`.

Code analysis questions:

1. Consider the following design for a data structure:

```
public class Processor<T> {
  ArrayList<T> list;
  public Processor() {this(10);}
  public Processor(int size) {if (size > 0) list = new ArrayList<>(size);
else list = new ArrayList<>();}
  public void add(T item) {list.add(item);}
  public void set(T item) {list.set(0, item);}
  public T get() {return list.remove(0);}
}
```

   What kind of data structure is `Processor` emulating? Justify your answer with reference to the code. What can be changed to improve the performance of `Processor`?

2. Consider the following pair of methods:

```
private static ArrayList<Integer> primeDecomposition(int num,
int currentPrimePos) {
  ArrayList<Integer> primes = new ArrayList<>();
  int currentPrime = listOfPrimes[currentPrimePos];
  if (num == 1 || num == -1) return primes;
  else if (primes % currentPrime == 0) {
    primes.add(Integer.valueOf(currentPrime));
```

```
    primes.addAll(primeDecomposition(num / currentPrime, currentPrimePos));
  }
  else primes.addAll(primeDecomposition(num, ++currentPrimePos));
}
public static ArrayList<Integer> primeDecomposition(int num)
{
  return primeDecomposition(num, 0);
}
```

The methods attempt to find the prime decomposition of `num`, i.e. they try to write `num` as a product of prime numbers. To do this, when `num` is a multiple of the current prime number `currentPrime`, `currentPrime` is added to the prime decomposition of `num`, and the procedure is repeated on `num / currentPrime`. If not, the program moves onto the next prime number to see if `num` is a multiple of the new prime number.

The methods involve a type of repetition to perform their purpose. What kind of repetition are they using: iteration or recursion, and how do you know this? What is the relationship between the two methods? Write down one disadvantage of using the kind of repetition you answered for the first question to solve the problem.

3. Consider the following method:

```
public static void survey(ArrayList<File> files) {
  DataInputStream in;
  ListIterator<File> it = files.iterator();
  while (it.hasNext()) {
    try {
      in = new DataInputStream(new FileInputStream(in.next()));
      boolean end = false;
      while (!end) {
        try {
        System.out.println(in.readUTF());
        } catch(EOFException ex) {end = true;}
```

```
      }
    } catch (IOException ex) {it.remove();}
  }
}
```

(a) Suppose the size of the collection of files to be read is n and the average size of each file is m. Write down an estimate of the complexity of the method in terms of n and m.

(b) Explain how the method detects the end of each file. What happens to a file in the array if it does not exist?

(c) Does the performance of the method change if the data structure containing the files is changed to a `LinkedList`? Justify your answer.

Short answer questions:

1. Implement a recursive method that calculates the average of numbers in an array of type `double[]`. You may write additional helper methods. Your solution is acceptable as long as it uses recursion somewhere.

2. Consider the following program:

```
public class FileManager {
  public static void main(String[] args) {
    Scanner inKey = new Scanner(System.in);
    System.out.println("Enter a file name to print:");
    String filename = inKey.next();
    inKey.close();
    Scanner inFile = new Scanner(new File(filename));
    while(inFile.hasNext()) System.out.println(inFile.next());
    inFile.close();
  }
}
```

As written, the program will not run. Rewrite the program with exception handling in order to enable the program to run and make it robust against errors. Use the most specific exceptions possible.

Long answer question:

1. In Java, a `PriorityQueue` is implemented using a data structure called a heap, but it can also be implemented using data structures you have already learned even though it is much less efficient. Implement a data structure that mimics a `PriorityQueue`. You may use any data structure covered in class and the `sort` method from `Collections`. For brevity, only implement a no-arg constructor, `offer`, `poll`, `peek`, `size` and `isEmpty`.