CS211 Fall 2012

Dr. Kinga Dobolyi

Final Examination

Name: _____

Part 1: Short Answer.

1. Examine the code below, and answer the following questions. (37 points)

```
import java.util.*;

public abstract class Exam implements Comparable{

     private int age;
     public ArrayList list;
     public static String type;
     private static boolean flag;

     public Exam(int ageIn){
          age = ageIn;
     }

     private void func1(){
          //question 2
     }

     private static int func2(){
          return 0;
     }

     public static void func3(){
     }

     public int func4(){
          return 1;
     }

     public boolean equals (Exam o){
          return false;
     }
}
```

a. List the attributes and methods of **`Exam`** that a direct subclass of **`Exam`** could access directly

b. List the attributes and methods of **`Exam`** that **`func1`** could access directly

c. List the attributes and methods of **`Exam`** that **`func2`** could access directly

d. List the attributes and methods of **`Exam`** that **`func3`** could access directly

e. List the attributes and methods of **`Exam`** that **`func4`** could access directly

f. Will the class above compile? Circle YES or NO and explain your answer:

g. Imagine I had the standard **`main`** method *inside this class*. Which attributes and methods could the **`main`** method call from itself?

h. Imagine I have the following statements (and assume they compile). What would the output be?

```
Object o1 = new Exam(3);
Object o2 = o2;
System.out.println(o1.equals(o2));
```

i. Imagine I have the following statements (and assume they compile). What would the output be?

```
Exam o1 = new Exam(3);
Exam o2 = o2;
System.out.println(o1.equals(o2));
```

2. Answer the following question assuming you have a primitive array of primitive integers called **`array`**, and a singly linked list of `IntegerNodes` called **`linkedList`**. The linked list does not have a counter for the number of elements, and it only has a pointer to the start of the list. Assume both lists are of some arbitrary size *n* (your answers must generalize to all *n*). You may not assume that for an array, there is always room in the array to add a new element. In the questions below, when discussing runtime, assume that we are measuring the number of *comparisons*, that is, how they increase as the size of the lists increases. Document your assumptions: (12 points)

    a. Assume both lists are unsorted. What is the best case insertion time for the array?

    b. Assume both lists are unsorted. What is the best case insertion time for the linkedList?

    c. Assume both lists are unsorted. What is the worst case insertion time for the array (assume that you use the public **`length`** attribute of the array)?

    d. Assume both lists are unsorted. What is the worst case insertion time for the linkedList?

e. Assume both lists are sorted. What is the best case insertion time for the array?

f. Assume both lists are sorted. What is the best case insertion time for the linkedList?

g. Assume both lists are sorted. What is the worst case insertion time for the array?

h. Assume both lists are sorted. What is the worst case insertion time for the linkedList?

i. A (circle one) SORTED UNSORTED linked list has a faster search time.

j. A (circle one) SORTED UNSORTED primitive array has a faster search time.

k. A (circle one) SORTED LINKED LIST BINARY SEARCH TREE has a faster search time.

3. Extend the **Exam** class from the first question with a concrete class called **MyExam**. Your **MyExam** class should store a **String name** and a primitive integer **score**. Any **MyExam** object must maintain the following invariants: the **name** must always be of length at least two, and the **score** must always be between zero and 100 inclusive. Your code must use good object-oriented design. You should write a constructor that sets as many of the fields of the parent class as it can, as well as the **name** and **score**, to incoming arguments. You must write one setter for either the **name** or the **score**. Make sure your **MyExam** class compiles. Overwrite the **toString** method from **Object** to return the **name** and the **score**. (23 points)

4. Examine the following method, and answer the questions below. Assume that the method compiles. (9 points)

```
public boolean equals (Object o){
          String message = "all good";
          try {
               Scanner scan = new Scanner(new File("log.txt"));
               message += SomeOtherClass.function();
               message += scan.readLine();
               message += ((MyClass) o).toString();
          }
          catch (IOException e){
               message = "le fail";
          }
          catch (NullPointerException e){
               message = "oh noes!";
          }
          catch (Exception e){
               message = "r u serious?";
          }
          finally{
               message += "\nat least I close my buffers...";
          }

          message += message.substring(0,7);
          System.out.println(message);
          return true;
}
```

a. What will the method print if log.txt does not exist?

b. What will the method print if the **`function`** in **`SomeOtherClass`** tries to divide by zero?

c. What will the method print if there is nothing to be read from log.txt?

d. What will the method print if the incoming object argument is of type **`String`**?

e. What will the method print if the incoming object argument is null?

f. Could this method ever throw an exception?

g. Is a **`NullPointerException`**, in general, a checked or unchecked exception?

h. Do we need the try-catch block? Explain.

5. The kitchen sink. Complete the following recursive method below. It takes as argument an **`ArrayList`** of **`String`** objects, and returns a **`String`** that is made up of all pieces of the strings inside in the list, in order, that match the regular expression described below. Your method must: (18 points)
    a. Be recursive. You cannot use any for or while loops.
    b. It must collect all strings that match the following regular expression: a name, which is made up of at least two and at most seven lower and uppercase letters, followed by one or more whitespace characters, followed by any positive integer, followed by a period, followed by either {the uppercase Q, or as many lowercase q-s as you would like}. Recall that the **`String`** class has a **`public boolean matches(String regex)`** method. Each element should be separated by a newline in the string you return.
    c. Your method must rely and use generics as indicated by the method signature below. You may not change the method signature.
    d. You may assume that the method, called from main below, would have the output below:

```
public static void main(String[] args){
     ArrayList<String> list = new ArrayList<String>();
     list.add("Kinga 1.Q");
     list.add("Kingaaaa 1.Q");
     list.add("2Kinga 1.Q");
     list.add("i 1.");
     list.add("Kinga  100.Q");
     list.add("Kinga  100.Qq");
     list.add("Kinga  100.qq");

     System.out.println(collect(list,0));
}
```

(more on next)

with output:

```
Kinga 1.Q
Kinga  100.Q
Kinga  100.qq
```

As we did in class, make sure to test your code before you turn in your exam! Think of some other test cases as well…

```
public static String collect (ArrayList<String> list, int ctr){
```