University of Maryland College Park
Department of Computer Science
CMSC131 Spring 2022
Exam #3

FIRSTNAME, LASTNAME (PRINT IN UPPERCASE): KEY

STUDENT ID (e.g. 123456789):

Instructions

- Please print your answers and use a pencil.
- Do not remove the staple from the exam. Removing it will interfere with the Gradescope scanning process.
- To make sure Gradescope can recognize your exam, print your name, write your directory id at the bottom of pages with the text DirectoryId, provide answers in the rectangular areas provided, and do not remove any exam pages. Even if you use the provided extra pages for scratch work, they must be returned with the rest of the exam.
- This exam is a closed-book, closed-notes exam, with a duration of 50 minutes and 100 total points.
- Your code must be efficient.
- Multiple choice questions have only one answer unless indicated otherwise.
- You don't need to use meaningful variable names; however, we expect good indentation.

Grader Use Only

#1	Problem #1 (Short Answers - 2pts each)	16
#2	Memory Map	24
#3	Coding	60
Total	Total	100

Problem #1 (Short Answers – 2 pts each)

1. (2 pts) **Double** is the wrapper class for the primitive type `double` (Your answer needs to be case-sensitive or will be marked wrong).

2. (2 pts) A Java interface can have **abstract** methods, meaning that the body of the method is missing.

3. (2 pts) The **signature** of a method includes its name and parameter list.

4. (2 pts) Show how you would declare a variable called `myList` that can reference an `ArrayList` of integers and assign to it an empty `ArrayList` object (should be all done in one line).

```
ArrayList <Integer> myList = new ArrayList<>();
```

Assume the following 4 files all in the same package. Use the code below for questions 5 to 8.

```
  public interface Exam3Interface {
         public String whichTerm();
  }
------------------------------------------------------------
  public class Exam3SP22 implements Exam3Interface{
        public String whichTerm() {
              return "SP22";
        }
  }
------------------------------------------------------------
  public class Exam3F21 implements Exam3Interface{
        public String whichTerm() {
              return "F21";
        }
  }
------------------------------------------------------------
public class Driver {
        public static void main(String[] args) {
              //code for problem 7 and 8
        }
}
```

5. (2 pts) Adding the field: `public int total;` to the class `Exam3SP22` will cause a compilation error since the field is not also in the `Exam3Interface`. **True or False**? **False**

6. (2 pts) Adding the method: `public int a() {return 0;}` to the class `Exam3SP22` will cause a compilation error since the method is not also in the `Exam3Interface`. **True or False**? **False**

7. (2 pts) The following statement in the `main` method would cause a compilation error.

   `Exam3Interface e = new Exam3Interface();` **True or False**? **True**

8. (2 pts) The following statement in the `main` method would cause a compilation error since the types are different.

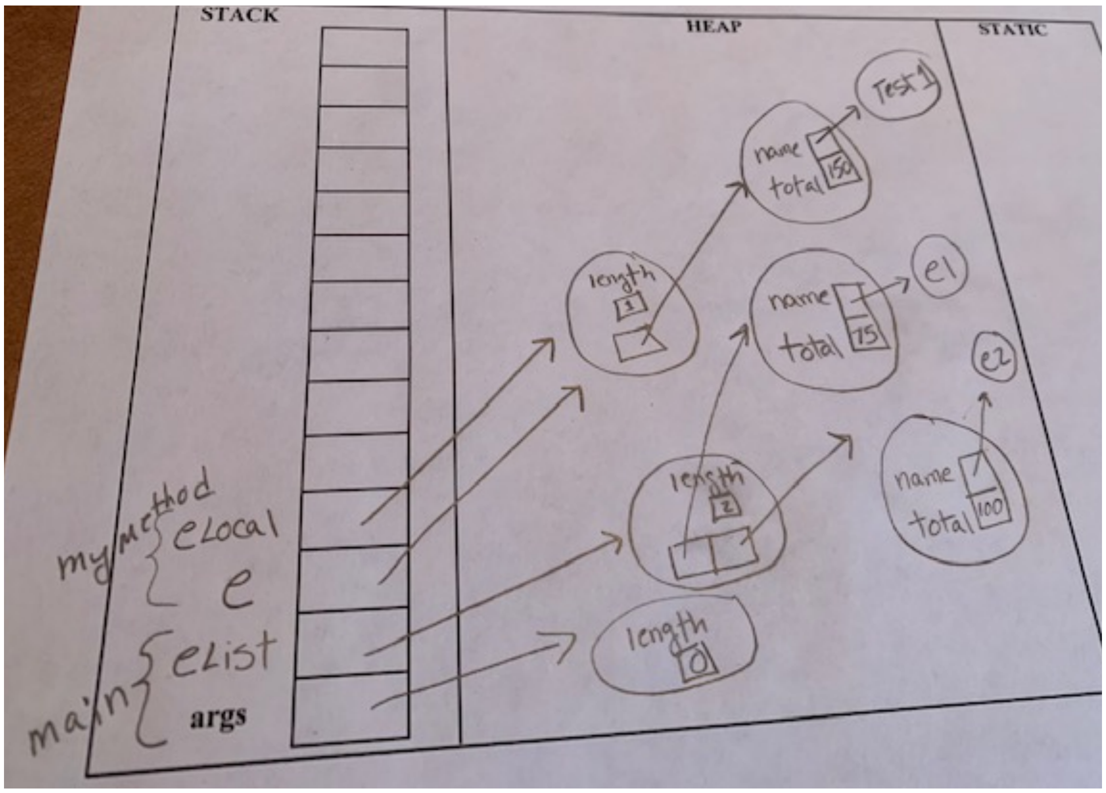
   `Exam3Interface e = new Exam3F21();` **True or False**? **False**

Problem #2 (Memory Map – 24 pts)

Draw a memory map for the following program up to the point in the program execution indicated by the comment **/*HERE*/**. Indicate where each frame starts and ends in the stack. Remember not to draw the frame for methods that would have finished by the time we reach the stopping point (/* HERE */).

```
public class Exam {
        public String name;
        public int total;
        public Exam(String name, int total) {
                this.name = name;
                this.total = total;
        }
        public void setTotal(int total) {
                this.total = total;
        }

}
```

```
public class MemMaps {

        public static void myMethod(Exam []e) {
                Exam []eLocal = {new Exam("Test1",200)};
                e[1].setTotal(100);
                e = eLocal;
                e[0].setTotal(150);
                /* HERE */
        }

        public static void main(String[] args) {
                Exam []eList = {new Exam("e1",75),
                                new Exam("e2", 80)};
                myMethod(eList);
         }

}
```



Directory id:

Problem #3 (Coding – 60 pts)

Complete the implementation of a class called `csStudent` that represents a CS student. A `csStudent` has an id (`id` instance variable), and 2D array where each row holds information about the projects completed by the CS student during one academic year (`proj` instance variable). You are given all the code of the `csStudent` and only need to complete the following 3 methods: 1) `arrayCleaner`, 2) `csStudent` constructor, 3) `makePortfolio`. You are also given the entire code for the `Project` class that represents a CS project, where the difficulty level is an integer from 0 to 5 with 0 being the easiest and 5 being the hardest.

```
public class Project {
   private String name;
   private int difficultyLevel;

   public Project(String name, int difficultyLevel) {
        this.name = name;
        this.difficultyLevel = difficultyLevel >=0 &&  difficultyLevel <=5? difficultyLevel: 5;
   }
   public int getDifficultyLevel() {
        return difficultyLevel;
   }
   public String toString() {
        return "(" + name + ": " + difficultyLevel + ")";
   }

}
----------------------------------------------------------------------------------------------------

import java.util.Arrays;

public class csStudent {

   private int id;
   private Project [][]proj;

   public static void arrayCleaner(Project[][] p){
        //YOUR CODE IS ANSWER TO P3 #1
   }

   public csStudent(int id, Project[][] p) {
        //YOUR CODE IS ANSWER TO P3 #2
   }

   public Project[] makePortfolio() {
        //YOUR CODE IS ANSWER TO P3 #3
   }

   public static String str2DArray(Project[][] p){

        String retVal ="";
        if(p!=null) {
               for (int i =0; i<p.length; i++)
                      retVal+=Arrays.toString(p[i])+"\n";
        }
        return retVal;
   }

   public String toString() {
        return "id=" + id + "\n" + str2DArray(proj);
   }

   public static void main(String[] args) {

        Project[][] p = { {new Project ("P1", 3), null, new Project ("P2", 3)},
                          null, //row is null
                          {},  //row is empty
                          {new Project ("p1", 4),  new Project ("p2", 5), new Project ("p3", 4)},
                          {null,  new Project ("A", 5), new Project ("B", 5), new Project ("C", 5)}
        };


        System.out.println(str2DArray(p));
        System.out.println("------------------------");

        csStudent s1 = new csStudent(123, p);
        System.out.println(str2DArray(p));  //the argument after running arrayCleaner in constructor
        System.out.println("-------------------------");
        System.out.println(s1);
        System.out.println("-------------------------");

        p[0]=p[1]=p[3]= null;
        p[4][0] =new Project ("Changed",3);
        p[4][1] =null;

        System.out.println(str2DArray(p));   //p has now changed
        System.out.println("-------------------------");
        System.out.println(s1);   //but not proj field of s1
        System.out.println("-------------------------");
        System.out.println(Arrays.toString(s1.makePortfolio()));
    }
}
```

Output of the `main` method

```
[(P1: 3), null, (P2: 3)]
null
[]
[(p1: 4), (p2: 5), (p3: 4)]
[null, (A: 5), (B: 5), (C: 5)]

-------------------------
[(P1: 3), (Easy Project: 0), (P2: 3)]
[(Easy Project: 0)]
[(Easy Project: 0)]
[(p1: 4), (p2: 5), (p3: 4)]
[(Easy Project: 0), (A: 5), (B: 5), (C: 5)]

-------------------------
id=123
[(P1: 3), (Easy Project: 0), (P2: 3)]
[(Easy Project: 0)]
[(Easy Project: 0)]
[(p1: 4), (p2: 5), (p3: 4)]
[(Easy Project: 0), (A: 5), (B: 5), (C: 5)]

-------------------------
null
null
[(Easy Project: 0)]
null
[(Changed: 3), null, (B: 5), (C: 5)]

-------------------------
id=123
[(P1: 3), (Easy Project: 0), (P2: 3)]
[(Easy Project: 0)]
[(Easy Project: 0)]
[(p1: 4), (p2: 5), (p3: 4)]
[(Easy Project: 0), (A: 5), (B: 5), (C: 5)]

-------------------------
[(P2: 3), (Easy Project: 0), (Easy Project: 0), (p2: 5), (C: 5)]
```

1. The purpose of the `arrayCleaner` method is to potentially modify the passed in 2D array so that each row has at least one element. First, check to see if the parameter is null or if the 2D array has no rows. If that is the case, simply throw the `IllegalArgumentException` with the message "`ERROR - Parameter is null or has no rows`". At this point, you have established that the array has at least one row. If a row is null or an empty array, make that row reference a 1-D array of `Project` with one element, where the element references a `Project` with the name "`Easy Project`" and `difficultyLevel` of 0. If a row references an array with an element that is null, make the element reference a `Project` with the name "`Easy Project`" and `difficultyLevel` of 0. If an element in a row is not referencing null, make no changes to it. Notice that even though the method is void, the changes made to the parameter will persist after the method returns. You can see this by looking at the output of the variable `p` in the sample `main` before and after the constructor call (the constructor will call `arrayCleaner`).

Directory id:

```
public static void arrayCleaner(Project[][] p)
      {
            if (p==null || p.length ==0 ){   //if null or empty
        throw new IllegalArgumentException("ERROR - Parameter is null or has no rows");
            }


            for(int i =0; i<p.length; i++)
            {
                  if(p[i]==null || p[i].length ==0)  //if null or empty row
                  {
                        p[i]= new Project[1];
                        p[i][0] = new Project ("Easy Project", 0);
                  }
                  else {
                        for(int j =0; j<p[i].length; j++)
                        {

                              if(p[i][j]==null)
                              {
                                    p[i][j] = new Project ("Easy Project", 0);
                              }

                        }
                  }

            }


      }
```

2. Next, code the `csStudent` constructor. First, use a try/catch clause to call `arrayCleaner` on the parameter `p`. If the exception is caught, simply use `System.out.println` to print the message and return from the constructor. Assuming the exception is not thrown by `arrayCleaner`, finish up the code in the constructor by assigning the `id` parameter to the `id` field (no error checking needed) and make an **independent** duplicate 2D array structure (i.e. same number of rows and columns, and same `Project` references) of the parameter `p` and assign the copy to the field `proj`. Notice that since `Project` is an immutable class, assigning the references found in the elements of `p` to the corresponding elements of `proj` is fine, but the rows and columns of the `proj` field have to be separate from `p`. If done correctly, changes to `p` after the constructor call will not modify the `proj` field as seen in the sample `main`. You must write **all the code** that makes the duplicate array without resorting to any library calls that help in duplicating an array.

```
public csStudent(int id, Project[][] p) {
            try {
                  arrayCleaner(p);
            }
            catch (IllegalArgumentException e) {
                  System.out.println(e.getMessage());
                  return;
            }
            //at least one row with one cell at this point

            this.id = id;

            proj = new Project[p.length][];  //same # of rows as p

            for (int i =0; i<proj.length; i++)
                  proj[i] = new Project[p[i].length];  //same # of cells for each row

            for (int i =0; i<proj.length; i++)
                  for (int j =0; j<proj[i].length; j++)
                        proj[i][j] = p[i][j];


      }
```

Directory id:

3. The `makePortfolio` method returns a 1D array of `Project` where the element at index *i* references the most difficult project in row *i* of the `proj` field. Since this instance method can only be called by an existing `csStudent` (i.e. the constructor and therefore `arrayCleaner` has already been called), you know that each row has at least one `Project`. If more than one project in a row *i* have the same highest difficulty level, the **last one** should be assigned to the array element *i*.

```
public Project[] makePortfolio() {
            Project[] portfolio = new Project[proj.length];
            for (int i =0; i<proj.length; i++)
            {     int index = 0;  //index of hardest
                  int hard =0;  //hardest project
                  for (int j =0; j<proj[i].length; j++)
                  {
                        if (proj[i][j].getDifficultyLevel()>=hard)
                        {
                              hard =proj[i][j].getDifficultyLevel();
                              index=j;
                        }

                  }
                  portfolio[i]= proj[i][index];
            }

            return portfolio;


      }
```

EXTRA PAGE IN CASE YOU NEED IT

LAST PAGE

Directory id: