



# University of Maryland College Park

## Department of Computer Science

### CMSC132 Summer 2023

### Exam #1

FIRSTNAME, LASTNAME (PRINT IN UPPERCASE):

**KEY**

STUDENT ID (e.g. 123456789):

#### Instructions

- Please print your answers and use a pencil.
- This exam is a closed-book, closed-notes exam with a duration of 50 minutes and 100 total points.
- **Do not remove the exam's staple.** Removing it will interfere with the scanning process (even if you staple the exam again).
- Write your directory id (e.g., terps1, not UID) at the bottom of pages with **DirectoryId**.
- Provide answers in the rectangular areas.
- Do not remove any exam pages. Even if you don't use the extra pages for scratch work, return them with the rest of the exam.
- Your code must be efficient and as short as possible.
- If you continue a problem on the extra page(s) provided, make a note on the particular problem.
- For multiple choice questions you can assume only one answer is expected, unless stated otherwise.
- You don't need to use meaningful variable names; however, we expect good indentation.
- **You must write your name and id at this point (we will not wait for you after time is up).**
- You must stop writing once time is up.

#### Grader Use Only

#1	Problem #1 (Short Answer)	28	
#2	Problem #2 (Class Implementation)	72	
<b>Total</b>	Total	100	

## Problem #1 (Short Answer)

For question 1 to 8, assume the following 3 classes all in the same package.

```
public interface Exam1Interface {
    void interfaceMethod();
}

public class Computer {
    private int modelName;
    public void myMethod(){
        System.out.println("base");
    }
}

public class Laptop extends Computer implements Exam1Interface{
    private int batteryHour;
    public void myMethod(){
        System.out.println("laptop 1");
    }
    public void myMethod(double x){
        System.out.println("laptop 2");
    }
    public void interfaceMethod() {
        System.out.println("laptop 3");
    }
    public static void main(String[] args) {
        //Code to be put here for question 1 to 6
    }
}
```

For questions 1 to 6, assume the code given is the only code placed in the main method above. Simply write **NC** if it would not compile, **CE** if it would compile but throw an exception, **C** if it would compile but no output to the console, or write the output if it will compile and produce output. 3 points each.

1. NC, C, CE, or output? **NC**

```
Computer c = new Laptop();
Exam1Interface e = c;
```

2. NC, C, CE, or output? **NC**

```
Computer c = new Laptop();
c.myMethod(7.5);
```

3. NC, C, CE, or output? **laptop 3**

```
Computer c = new Laptop();
((Laptop) c).interfaceMethod();
```

4. NC, C, CE, or output? **NC**

```
Exam1Interface e = new Laptop();
e.batteryHour = 9;
```

5. NC, C, CE, or output? **CE**

```
Computer c = new Computer();
Laptop L = (Laptop)c;
```

6. NC, C, CE, or output? **C**

```
Computer c = new Laptop ();
Laptop L = (Laptop)c;
Exam1Interface e = L;
```

7. (3 pts) In no more than 2 sentences, explain why adding the following constructor to the Laptop class would not work?

```
public Laptop(int mN, int bH){
    super();
    modelNumber = mN;
    batteryHour = bH; }

```

**modelNumber is private to base class and cannot be directly accessed in Laptop constructor.**  
**Note, there is nothing wrong with using super (i.e. that is not the problem).**

8. (3 pts) In no more than 2 sentences, explain why adding the following method to the Laptop class would not work?

```
public int myMethod(double y){
    System.out.println("laptop 4"); }

```

**This would not be a valid overload. No difference in signature from the one already in the code.**

9. (2 pts) **Throwable** is the parent class of Exception.
10. (2 pts) The class Error and its subclasses are considered **checked** exceptions.

## **Problem #2 (Class Implementation)**

Assume the following classes are all in the same package. Also assume that there is a class called MathFaculty. The only thing you need to know about the MathFaculty class to code your part is that a MathFaculty *is a* Faculty.

```
import java.util.ArrayList;

public abstract class Faculty {
    private int id;
    ArrayList<String> tasks;

    //assume non-null tasks with at least 1 string element
    public Faculty(int id, ArrayList<String> tasks) {
        this.id = id;
        this.tasks = tasks; }

    public int getId() {
        return id; }

    public static String allTasks(Faculty f) {
        //you will code this } }

-----
import java.util.ArrayList;
import java.util.Collections;

public class CSFaculty extends Faculty implements Comparable<CSFaculty>{
    private ArrayList<String> csTasks;

    //assume non-null tasks & csTasks with at least 1 string element each
    public CSFaculty(int id, ArrayList<String> tasks, ArrayList<String> csTasks) {
        //you will code this }

    public ArrayList<String> getCsTasks() {
        //you will code this }

    @Override
    public String toString() {
        return getId()+ " " + csTasks + " " + tasks ; }

    @Override
    public int compareTo(CSFaculty o) {
        //you will code this } }

```

```

import java.util.ArrayList;

public class Driver {
    public static void main(String[] args) {
        ArrayList <String> task =new ArrayList <String> ();
        task.add("teach"); task.add("meeting");
        ArrayList <String> CStask =new ArrayList <String> ();
        CStask.add("program"); CStask.add("task1");
        CStask.add("read"); CStask.add("quiz");
        CStask.add("task2");CStask.add("exam");

        CSFaculty csF = new CSFaculty (123, task, CStask);
        MathFaculty mathF= new MathFaculty (456, task);

        System.out.println(csF); //task1 is first cs task

        System.out.println(Faculty.allTasks(csF)); //tasks followed by csTasks
        System.out.println(Faculty.allTasks(mathF)); //just tasks

        System.out.println(csF.compareTo(csF)); //everything the same, so 0

        CSFaculty csF1 = new CSFaculty (789, task, CStask);
        System.out.println(csF.compareTo(csF1)); //same # of tasks & csTasks but 123<789, so -1

        task.remove(1);
        CSFaculty csF2 = new CSFaculty (231, task, CStask);
        System.out.println(csF.compareTo(csF2)); //1 less task than csF, so 1

        CStask.add("task3");
        CSFaculty csF3 = new CSFaculty (456, task, CStask);
        System.out.println(csF.compareTo(csF3));//1 less task than csF,but 1 more csTask, so -1

        task.clear(); //does not clear the tasks of the csF, only local copy
        CStask.clear(); //does not clear the csTasks of the csF, only local copy

        System.out.println(csF);

        ArrayList <String> localCStask= csF.getCsTasks();
        localCStask.clear(); //no privacy leak
        System.out.println(csF);
    }
}

```

### Output

```

123 [task1, exam, program, quiz, read, task2] [teach, meeting]
teach meeting task1 exam program quiz read task2
teach meeting
0
-1
1
-1
123 [task1, exam, program, quiz, read, task2] [teach, meeting]
123 [task1, exam, program, quiz, read, task2] [teach, meeting]

```

1. **Constructor** – The resulting object should be totally *independent* of the 2 ArrayList objects passed in as the argument. You can assume both ArrayList have at least 1 string element and that any additional elements will be String (not null). Assign the arguments to the corresponding fields. The strings in csTasks must be sorted in alphabetical order. However, if the String literal task1 shows up as an element in csTasks it must be the first element after sorting. You can assume the String task1 is either in csTasks once or not at all. The field tasks should not be sorted. As for library methods, you may only use: ArrayList constructors, Collections.sort, ArrayList methods: size, get, and set, and the equals method of String to check for task1.

```

public CSFaculty(int id, ArrayList<String> tasks, ArrayList<String> csTasks) {

    super(id, new ArrayList<String>(tasks));
    this.csTasks = new ArrayList<String>(csTasks);

    Collections.sort(this.csTasks);
    //see if it has "task1" - if there assume just one time
    boolean found = false;
    for (int i =this.csTasks.size()-1; i > 0; i--)
    {
        if (this.csTasks.get(i).equals("task1"))
        {
            found =true;

        }
        if (found)
        {
            this.csTasks.set(i, this.csTasks.get(i-1)); //shift right
        }
    }
    if (found)
    {
        this.csTasks.set(0, "task1");
    }
}

```

2. `getcsTasks()` – A getter avoiding a privacy leak. If you have more than one statement, you are doing something wrong. You can use the one library method needed to get this done.

```
public ArrayList<String> getcsTasks() {  
    return new ArrayList<String>(csTasks);  
}
```

3. `compareTo` – The current object is considered larger if it has more elements in its `csTasks` and smaller if it has less elements in its `csTasks`. If the number of elements in `csTasks` is the same for both objects, look at the number of elements in the `tasks` field. At this point, the current object is considered larger if it has more elements in its `tasks` and smaller if it has less elements in its `tasks`. If the number elements are the same in `tasks` too, look at `id`. The current object will be considered larger if it has a larger id number, and smaller if it has the smaller id. If the ids are the same, the two objects are equal. You can only return a 1, 0, or -1. You should know what to do based on the standard convention associated with `compareTo`. Only library method allowed is `size` of `ArrayList` class.

```
public int compareTo(CSFaculty o) {  
  
    if (this.csTasks.size() > o.csTasks.size())  
        return 1;  
    else if (this.csTasks.size() < o.csTasks.size())  
        return -1;  
    else if (this.tasks.size() > o.tasks.size()) //same # of csTasks, check tasks  
        return 1;  
    else if (this.tasks.size() < o.tasks.size())  
        return -1;  
    else if (this.getId() > o.getId()) //same # of both tasks, check id  
        return 1;  
    else if (this.getId() < o.getId())  
        return -1;  
    else  
        return 0;  
}
```

4. **allTasks** – Simply return a `String` with all elements in `tasks` concatenated to each other with a space between them. If a `CSFaculty` is passed in, it also needs to concatenate to the `String` all elements in `csTasks` with a space between them. For looping constructs, you can only use a for each loop and no library methods are allowed (but all operators....hint hint...are allowed).

```
public static String allTasks(Faculty f) {  
  
    String s = "";  
  
    for (String t: f.tasks)  
    {  
        s+=t + " ";  
    }  
  
    if(f instanceof CSFaculty)  
    {  
        for (String t: ((CSFaculty)f).getcsTasks())  
        {  
            s+=t + " ";  
        }  
    }  
  
    return s;  
  
}
```

**Extra Page If You Need It**

**Last Page**