This work complies with the JMU Honor Code.

Name:_____ Signature:_____ Section:_____

***Instructions***. Answer all the following questions. This is a "closed book" examination, and you must work entirely on your own.

1. (10 points) Given the following declarations:

```
double                          weight;
int[]                           values;
String[]                        brands;
HashMap<Integer, String>        hmap;
```

Indicate whether each of the following is a *primitive value* (V), a *reference* (R), or *syntactically invalid* (I). Assume all variables are properly initialized.

1.1 ___V__ `weight`

1.2 ___R__ `values`

1.3 ___R__ `brands`

1.4 __I___ `brands.get(0)`

1.5 ___V__ `values[1]`

1.6 ___V__ `hmap.size()`

1.7 ___R__ `hmap`

1.8 ___V__ `brands[0].charAt(1)`

1.9 ___I___ `hmap[0]`

1.10 __V__ `values.length`

2. Write a single line of Java code to accomplish each of the following:

2.1. Create a variable called `birthYears` with a new HashMap that maps individual names to birth years:

```
HashMap<String, Integer> birthYears = new HashMap<String, Integer>();
```

2.2. Using the `birthYears` map, add an entry with the key “Kevin” and the value of 2002:

```
birthYears.put(“Kevin”, 2002);
```

2.3. Create a variable called `gpas` with a new ArrayList to store GPAs (e.g., the decimal number `3.5`):

```
ArrayList<Double> gpas = new ArrayList<Double>();
```

2.4. Add the value `4.0` to the `gpas` list:

```
gpas.add(4.0);
```

2.5. Assume you have a variable named `uniqueWords` that holds a reference to a `Set` containing `Strings`. Print out the number of items stored in `uniqueWords`.

```
System.out.println(uniqueWords.size());
```

3. Consider the following method.

```
public static double magicFun(String a, String b) {
    int index;
    double val;
    double[] values;

    try {
        index = Integer.parseInt(a);
    } catch (NumberFormatException e) {
        index = 0;
    }
    try {
        if (index < 0) {
            index = 1;
            throw new IndexOutOfBoundsException();
        }
        val = Double.parseDouble(b);
        values = new double[] {val, val};

    } catch (NumberFormatException e) {
        values = new double[] {0.1, 0.1};
    } catch (IndexOutOfBoundsException e) {
        values = new double[] {0.0, 0.0};
    }

    return values[0] + values[1];
}
```

What would each of the following statements return?

| # | Statements | Output |
|---|---|---|
| 3.1 | `System.out.println(magicFun("-1", "1.0"));` | 0.0 |
| 3.2 | `System.out.println(magicFun("0", "2.0"));` | 4.0 |
| 3.3 | `System.out.println(magicFun("1", "double"));` | 0.2 |

4.1. Given the attached implementaton of the classes and interfaces, what will be printed by the following program? ***Note: The class does NOT contain any syntax errors.***

```
public class Driver1 {
    public static void main(String[] args) {
        Extra extra;
        extra = new Extra("Carl");
        extra = new Extra("Celine");
        CrewMember crew = new CrewMember("Bob", Job.WRITER, "WGA");
        CastMember cast = new CastMember("Diego", "Max");

        System.out.println(crew.toString());
        System.out.println(extra.toString());
        System.out.println(cast.toString());
    }
}
```

```
Bob #3 (Crew)
Celine #2
Diego #4
```

4.2. What is the output of the following code? Note: the class does NOT contain any syntax errors.

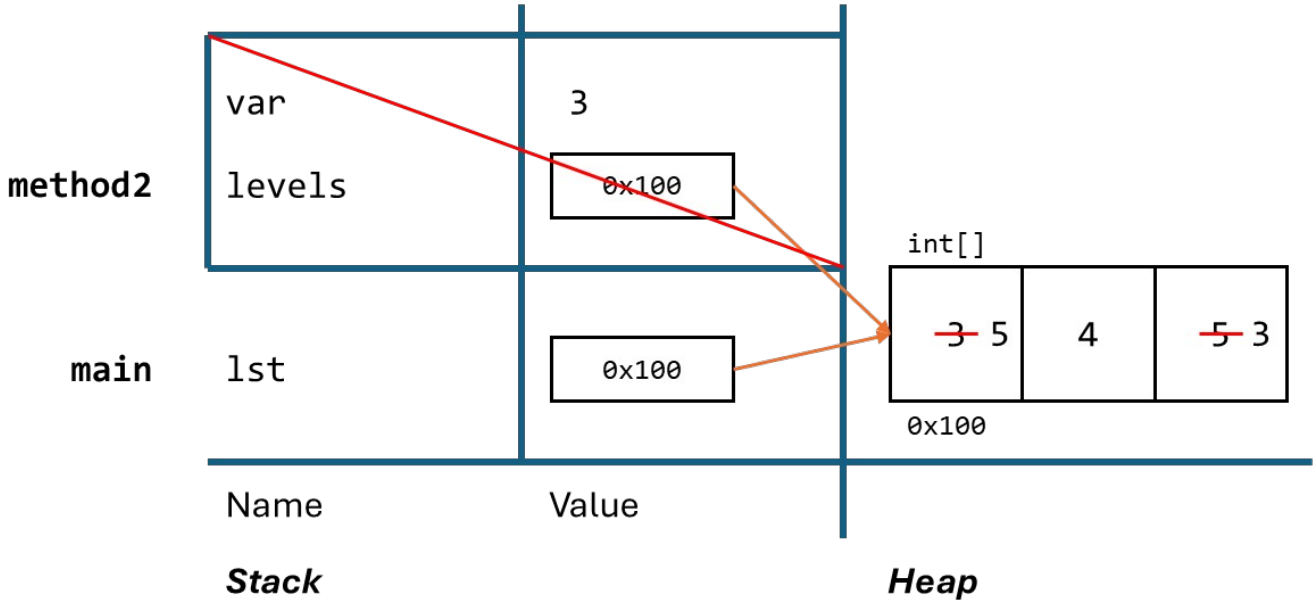
```
public class ChallengeClass{
    public static void method2(int[] levels) {
        int var = levels[0];
        levels[0] = levels[levels.length - 1];
        levels[levels.length - 1] = var;
    }

    public static void main(String[] args) {
        int[] lst = {3, 4, 5};

        method2(lst);
        System.out.println(lst[0] + " " + lst[lst.length - 1]);
    }
}
```

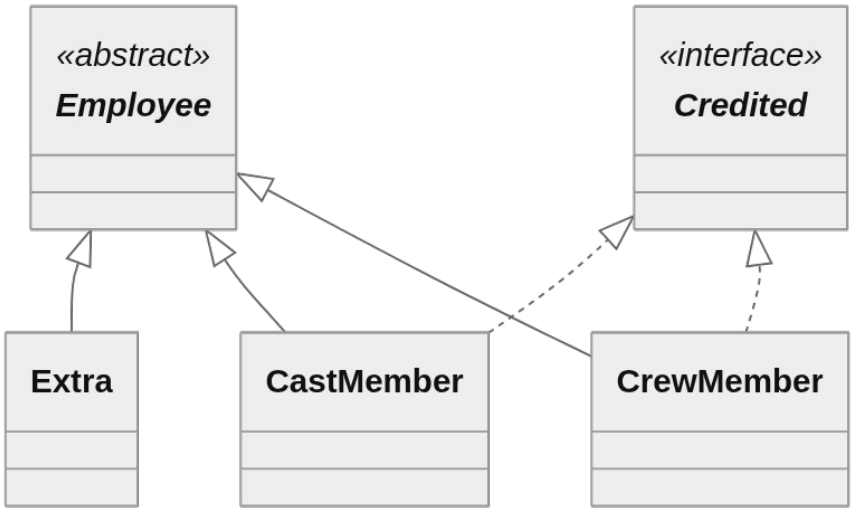
```
5 3
```

For full credit, draw a memory diagram below:



5. Draw a UML class diagram consisting of `Credited`, `Employee`, `Extra`, `CrewMember`, and `CastMember`. For full credit, you must use correct UML syntax, including arrow types. Clearly label any italicized text. DO NOT INCLUDE ATTRIBUTES AND METHODS IN YOUR DIAGRAM.

***(You may also omit the `Job` enum in your diagram.)***



6. Given the attached classes and interfaces, and the following statements indicate whether each of the following fragments will:

- N – not compile
- X – compile but generate an exception at run-time, or
- R – compile and run without generating an exception

For each snippet that does not compile or compiles but generates an exception, briefly explain.

```
Credited[] credits   = new Credited[2];
Employee[] employees = new Employee[3];
```

| # | Independent Code Snippet | N, X, R | For N or X – why? |
|---|---|---|---|
| 6.1 | `employees[0] = new CastMember("Edgar", "Waiter");` | R | |
| 6.2 | `employees[1] = new CrewMember();` | N | Missing arguments |
| 6.3 | `employees[2] = new Extra("Francine");` | R | |
| 6.4 | `credits[0] = new CastMember("Georgia", "Child #2");` | R | |
| 6.5 | `employees = new Employee[1];`<br>`employees[1] = new Extra("Francine");` | X | Out of bounds exception |
| 6.6 | `Employee e = new Employee("John", "SAG");` | N | Employee is abstract |
| 6.7 | `CrewMember crew = new CrewMember("Alice");` | N | Constructor needs more arguments |
| 6.8 | `CastMember cast = new CastMember("Charlie", "Director");`<br>`String role = cast.role;` | N | role is private |
| 6.9 | `employees[0] = credits[0];` | N | Incompatible types |
| 6.10 | `Extra extra = new Extra("Diane");`<br>`extra.getJob();` | N | No method getJob() |

7.Given the attached classes and interfaces, and the following methods:

```
public static void printType(CastMember v)
{
  System.out.println("CastMember");
}

public static void printType(Credited v)
{
  System.out.println("Credited");
}

public static void printType(Employee v)
{
  System.out.println("Employee");
}

public static void printType(Object v)
{
  System.out.println("Object");
}
```

indicate what will be printed by each of the following fragments (**which are in the same class as the methods above**). Note: Each fragment must be considered independently.

7.1

```
Employee v = new CastMember("Robert DeNiro", "Vito Corleone");
printType(v);
```

Employee

7.2

```
Credited v = new CastMember("Robert DeNiro", "Vito Corleone");
printType(v);
```

Credited

7.3

```
Extra v = new Extra("John Brown");
printType(v);
```

Employee

7.4

```
Object v = new CastMember("Robert DeNiro", "Vito Corleone");
printType(v);
```

Object

8. Create a new class named `Administrator` that extends `Employee`.

| Administrator |
| --- |
| - employeeList: List<CastMember><br>- employeeCount: int |
| + Administrator(name: String, union: String)<br>+ addEmployee(person: CastMember)<br>+ removeEmployee(employeeId: int)<br>+ loadEmployeeRecords(filename: String)<br>+ saveEmployeeRecords(filename: String):boolean |

It contains 2 additional attributes:

- `employeeList`: List containing all employees currently involved in the project.
- `employeeCount`: Integer representing the total number of employees currently involved.

Implement the following:

- Constructor: Accepts name and union as parameters. Initialize the superclass attributes with name and union. Initialize `employeeCount` to 0 and `employeeList` to an empty list.
- `addEmployee(CastMember person)`: Adds a person to `employeeList` and increments employeeCount.
- `removeEmployee(int employeeId)`: Removes a person by `employeeId` from `employeeList` if they exist and decrements `employeeCount`.

```
public class Administrator extends Employee {
    public List<CastMember> employeeList;
    private int employeeCount;

    public Administrator(String name, String union) {
        super(name, union);
        this.employeeList = new ArrayList<>();
        this.employeeCount = 0;
    }

    public void addEmployee(CastMember person) {
        employeeList.add(person);
        employeeCount++;
    }

    public void removeEmployee(int employeeId) {
        Iterator<CastMember> iterator = employeeList.iterator();
        while (iterator.hasNext()) {
            CastMember emp = iterator.next();
            if (emp.getID() == employeeId) {
                iterator.remove();
                employeeCount--;
                break;
            }
        }
    }
```

9. Implement the method `loadEmployeeRecords()` in the `Administrator` class.
Reads `CastMember` records from a file specified by filename.

Each line in the file should follow the format of: the name of the `CastMember` with a space and then followed by the `role`.

Example:

```
John Advisor
Tim IT-staff
Lucy Professor
```

You may assume that names and roles do not contain spaces.

Throws `FileNotFoundException` if the file is not found.

```
public void loadEmployeeRecords(String filename) throws FileNotFoundException {
      File file = new File(filename);
      Scanner in = new Scanner(file);

      while (in.hasNextLine()) {
          String line = in.nextLine();
          String[] parts = line.split(" ");
          String name = parts[0];
          String role = parts[1];

          CastMember employee = new CastMember(name, role);
          employeeList.add(employee);
      }
      in.close();
  }
```

10. Implement the method `saveEmployeeRecords()` in the `Administrator` class.

The method writes each employee's details to a file in the format: `name role`
Assume names do not contain spaces. For an example of the output format, see the previous problem.

Does **not** throw `FileNotFoundException`. Instead, it returns `false` if the file cannot be opened.

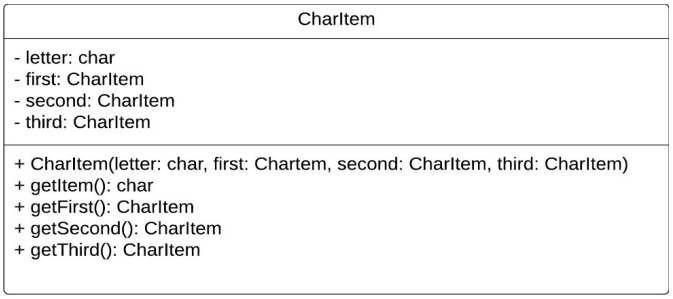
```
public boolean saveEmployeeRecords(String filename) {
    File file;
    PrintWriter out;

    try {
        file = new File(filename);
        out = new PrintWriter(file);
    } catch (FileNotFoundException e) {
        return false;
    }

    for (CastMember emp : employeeList) {
        out.printf("%s %s\n", emp.getName(), emp.getCredit());
    }

    out.close();
    return true;
}
```

11. Write a recursive method, `countChar(CharItem item, char target)`, to find the number of items with `letter` of `target` value in an arbitrary linked "tree" of `CharItems` where each item has the form indicated in the UML below.




Assume the method is called with the `top` (the top node of the tree) and the char `c`.
Example: `System.out.println(CountChar(top, c));`

```
public static int countChar(CharItem item, char target) {

    if (item == null) {
        return 0;
    }

    int count = 0;
    if (item.getItem() == target) {
        count = 1;
    }

    return count
            + countChar(item.getFirst(), target)
            + countChar(item.getSecond(), target)
            + countChar(item.getThird(), target);
}
```

12 Consider the recursive method below.

```
public static int magicFun(String[] words, int left, int right, String target) {
        if (left > right) {
            return 0;
        }

        int center = (left + right) / 2;

        int count = 0;
        if (words[center].equals(target)) {
            count = 1;
        }

        int leftVal = magicFun(words, left, center - 1, target);
        int rightVal = magicFun(words, center + 1, right, target);

        return count + leftVal + rightVal;
 }
```

Assuming that `String[] words = {"cs", "it", "cs", "cis", "math", "cs"};`

You may use the next page to trace the call `magicFun(words, 1, 5, "cs")`
The diagram has been started for you. Make sure to clearly show each call and its return value.

Answer the following questions.

12.1 What is the return value of `magicFun(words, 1, 5, "cs")`?

2

12.2 How many times is magicFun called with an initial call of `magicFun(words, 1, 5, "cs")`, including the first call (in other words how many stack frames/activation records are generated)?

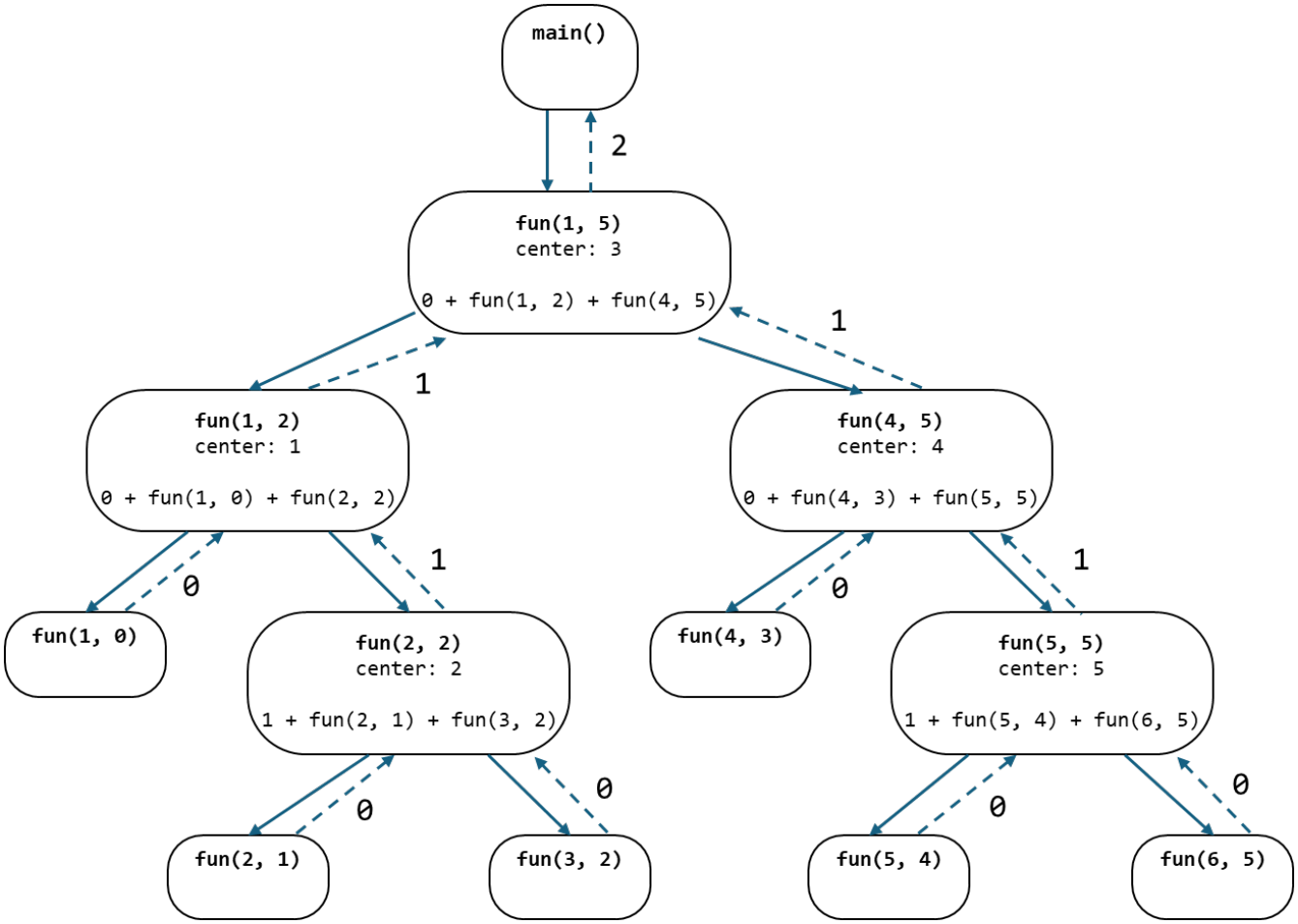
11

12.3 What is the maximum depth of the stack assuming `magicFun(words, 1, 5, "cs")` is called from main?

4

12.4 Trace the call `magicFun(words, 1, 5, "cs")`:

(Note that `words` and `target` never change, so we omit it in the diagram, only tracking `left` and `right`)



# Attachment

```
public abstract class Employee
{
    private static int nextID = 1;

    private final int    id;
    private final String name, union;

    public Employee(String name, String union){
        this.name  = name;
        this.union = union;

        id = nextID;
        ++nextID;
    }

    public String getBadge(){
        return "#" + id;
    }

    public int getID(){
        return id;
    }

    public String getName(){
        return name;
    }

    public String getUnion(){
        if (union == null) return "";
        else  return union;
    }


    public String toString(){
        return getName() + " " + getBadge();
    }
}
```

```
public interface Credited
{
    public abstract String getCredit();
    public abstract String getName();
}
```

```
public class CastMember extends Employee implements Credited
{
    private final String   role;

    public CastMember(String name, String role)
    {
        super(name, "SAG");
        this.role = role;
    }
    public String getCredit()
    {
        return role;
    }
}
```

```
public class CrewMember extends Employee implements Credited {
    private final Job job;

    public CrewMember(String name, Job job, String union) {
        super(name, union);
        this.job = job;
    }

    public String getBadge() {
        return super.getBadge() + " (Crew)";
    }

    public String getCredit() {
        if ((job == Job.PRODUCER) || (job == Job.EDITOR)) {
            return String.format("%s, %s", job.getDescription(), getUnion());
        } else {
            return job.getDescription();
        }
    }

    public Job getJob() {
        return job;
    }
}
```

```
public class Extra extends Employee {
    public Extra(String name) {
        super(name, "SAG");
    }

}
```

```
public enum Job {
    CAST("Cast Member"),
    CASTING("Casting Director"),
    COMPOSER("Music Composer"),
    COSTUMES("Costume Designer"),
    EDITOR("Editor"),
    EXECUTIVE("Executive Producer"),
    PRODUCER("Producer"),
    WRITER("Writer"),
    DIRECTOR("Director");

    private String description;

    private Job(String description) {
        this.description = description;
    }

    public String getDescription() {
        return description;
    }
}
```