



# University of Maryland College Park

## Department of Computer Science

### CMSC132 Fall 2022

### 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)	40	
#2	Problem #2 (Class Implementation)	60	
<b>Total</b>	Total	100	

## **Problem #1 (Short Answer)**

1. (2 pts) A class can be declared both `final` and `abstract`. True or False? **False**
2. (2 pts) An `enum` type can have a public constructor. True or False? **False**
3. (2 pts) In Java, you can have an `ArrayList` of primitive `int`. True or False? **False**
4. (2 pts) If your customized exception has `Exception` as its parent class, it will be an unchecked exception. True or False? **False**
5. (2 pts) In Java programming, one uses composition to create a “has-a” relationship. True or False? **True**
6. (2 pts) What is the visibility modifier one uses to make a method accessible to subclasses outside of the package of the parent class while preventing access to non-subclasses? **protected**
7. (2 pts) Code in this block will execute regardless of whether the `try` block completes or if an exception is thrown passing control to a `catch` clause? **finally**
8. (2 pts) In the code, below:

```
ArrayList<Double> example = new ArrayList<Double>();  
example.add(7.3);
```

The fact that the `add` method call compiles without an error, demonstrates this feature of Java introduced in version 5.

**autoboxing**

9. (3 pts) Assume you are working with a version of Java prior to the introduction of generics. Re-write the third line of code so it will compile and allow the assignment to take place.

```
ArrayList example = new ArrayList();  
example.add("hi");  
String s = example.get(0);
```

**String s = (String) example.get(0);**

10. (3 pts) In no more than 2 sentences, explain what is meant by late (dynamic) binding.

**System waits until runtime to decide which version of an overridden method to invoke based on the actual dynamic type of the object at runtime (not the static type of the variable).**

11. (3 pts) Assume a `csStudent` class implements a `Person` Interface and extends a `Student` class. What will take place when the following code fragment is executed?

```
Student s = new csStudent();  
Person p = new csStudent();  
boolean b = s.getClass() != p.getClass();
```

- a. An exception will be generated.
- b. **false will be assigned to b.**
- c. true will be assigned to b
- d. It will not even compile.

12. (3 pts) In no more than 2 sentences, explain what is the purpose of using the `final` keyword when declaring a method?

It prevents the method being overridden by subclasses.

13. (3 pts) Fix the code below so it will compile **without** a try/catch clause. Just draw an arrow to where the extra code need to be inserted and write the code at the start of the arrow.

```
public int myMethod() throws Exception {
    int x =0;
    if (x==0){
        throw new Exception ("don't / by zero");
    }
    else {
        System.out.println(5/x);
    }
    return 0;
}
```

14. (3 pts) In no more than 2 sentences, explain what is the general purpose of the call `super.toString()` when you override the `toString` of a class.

It allows you to call the base class version of `toString` from the body of the overridden version.

15. (3 pts) Assume `Student` inherits from `Person` and both classes have default constructors. Write one assignment statement that shows your understanding of what *upcasting* means.

Person p = new Student(); //should have no explicit cast

16. (3 pts) Assume you want to create more than one “is a” relationship for a `Student` class you are writing. In no more than 2 sentences, explain the strategy discussed in class to overcome the lack of multiple inheritance in Java.

Inherit from one parent class and implement as many interfaces as you want to create multiple is a relationships.

DirectoryId:

## Problem #2 (Class Implementation)

Assume the following 2 classes that you should not change.

<pre>public abstract class Project {      private StringBuffer name;      public Project(String name) {         this.name = new StringBuffer (name);     }      public String getName() {         return name.toString();     }      @Override     public String toString() {         return getName();     }  }</pre>	<pre>public class NonCSProj extends Project{      public NonCSProj(String name) {         super(name);     }  }</pre>
--	---

For this problem, you will complete the implementation of the `CSProject` and `Student` classes (whose partial definitions are provided below). A `CSProject` object has a `tests` field. This field will reference an integer with 3 elements. The first element is the number of public tests associated with the CS project, the second element is the number of release tests, and the third element is the number of secret tests. Assume when the object is created valid data will be passed in (i.e. an array that has 3 non-negative integers). The `Student` object has a `proj` field. This is all the projects that the student is working on. Assume all students have at least one project. All 5 classes (counting the driver below) are in the same package. **You may not add any instance nor static fields and you may not add any auxiliary methods to the classes.**

<pre>import java.util.Arrays; public class CSProject extends Project{      private int []tests;      public CSProject(String name, int[] tests) {         super(name);         this.tests = Arrays.copyOf(tests, 3);     }      public int getPublic(){         return tests[0];     }     public int getRelease(){         return tests[1];     }     public int getSecret(){         return tests[2];     }      public void clearTests(){         tests[0]=tests[1]=tests[2]=0;     }      @Override     public String toString() {         return getName()+ Arrays.toString(tests);     }      /* INCOMPLETE CLASS */  }</pre>	<pre>import java.util.ArrayList; public class Student implements Comparable&lt;Student&gt;{      private ArrayList &lt;Project&gt; proj;      public Student(ArrayList&lt;Project&gt; proj) {         super();         this.proj = new ArrayList&lt;Project&gt;(proj);     }      @Override     public String toString() {         return proj.toString();     }      /* INCOMPLETE CLASS */  }</pre>
---	---

Below you will see a sample driver and expected output that illustrates the functionality of the classes you need to implement.

## Sample Driver / Output

```
import java.util.ArrayList;
import java.util.Collections;

public class SampleDriver {

    public static void main(String[] args) {

        String result = "";

        CSProject cs1 = new CSProject("Blackjack", new int[] {5,3,4});
        CSProject cs2 = new CSProject("Webpage", new int[] {11,12,0});

        CSProject cs3 = new CSProject (cs1);
        result+= cs1+"\n"+cs3+"\n";

        cs1.clearTests();
        result+= cs1+"\n"+cs3+"\n";

        CSProject cs4 = new CSProject("someCSProj1", new int[] {11,12,0});
        CSProject cs5 = new CSProject("someCSProj2", new int[] {10,10,3});

        result+= cs2.equals(cs4)+"\n";
        result+= cs2.equals(cs5)+"\n";

        ArrayList <Project> s1Proj = new ArrayList <Project>();
        s1Proj.add(cs2);
        s1Proj.add(new NonCSProj("BioProj"));
        s1Proj.add(cs5);
        s1Proj.add(new NonCSProj("MathProj"));

        ArrayList <Project> s2Proj = new ArrayList <Project>();
        s2Proj.add(new CSProject("P1", new int[] {5,5,5}));
        s2Proj.add(new NonCSProj("HisProj"));
        s2Proj.add(cs5);

        ArrayList <Project> s3Proj = new ArrayList <Project>();
        s3Proj.add(new CSProject("P2", new int[] {7,2,3}));
        s3Proj.add(new CSProject("P3", new int[] {4,5,8}));

        ArrayList <Student> sList = new ArrayList <Student> ();
        sList.add(new Student(s1Proj));
        sList.add(new Student(s2Proj));
        sList.add(new Student(s3Proj));

        result+= sList+"\n";
        Collections.sort(sList);
        result+= sList+"\n";

        result+= Student.makeList(sList)+"\n";

        System.out.println(result);

    }

}
```

```
Blackjack[5, 3, 4]
Blackjack[5, 3, 4]
Blackjack[0, 0, 0]
Blackjack[5, 3, 4]
true
false
[[Webpage[11, 12, 0], BioProj, someCSProj2[10, 10, 3], MathProj], [P1[5, 5, 5], HisProj, someCSProj2[10, 10, 3]],
[P2[7, 2, 3], P3[4, 5, 8]]]
[[P2[7, 2, 3], P3[4, 5, 8]], [P1[5, 5, 5], HisProj, someCSProj2[10, 10, 3]], [Webpage[11, 12, 0], BioProj,
someCSProj2[10, 10, 3], MathProj]]
[29, 38, 46]
```

The bold line starting with **[[Webpage** and ending with **P3[4, 5, 8]]]** will be on one line in the console.

The bold and italicized line starting with **[[P2[7, 2, 3],** and ending with **MathProj]]** will be on one line in the console.

Due to space constraints on the exam, each line was split into 2 lines.

## 1. CSPProject Class Methods

**NOTE:** You cannot use methods from `java.util.Arrays` or any other library methods when you write the code for the `CSPProject` class.

- a. **Copy Constructor** – The resulting object should be totally independent of the object passed in as the argument. Notice in the sample driver, there is a call to the `clearTests` method to see if the copy constructor is implemented correctly. You will never call `clearTests` in the code you write. It is only used for testing out the code in the driver.

```
public CSPProject(CSPProject csProj) {  
    super(csProj.getName());  
    tests = new int[3];  
    for (int i=0; i<3;i++)  
        tests[i]=csProj.tests[i];  
}
```

- b. **equals**– Two `CSPProject` objects are equal if they have the same number of public tests, same number of release tests, and the same number of secret tests. Implement according to our CMSC 132 convention of using `instanceof`.

```
public boolean equals(Object obj) {  
    if (obj == this)  
        return true;  
    if (!(obj instanceof CSPProject))  
        return false;  
    CSPProject proj = (CSPProject) obj;  
    return tests[0]==proj.tests[0]&&  
        tests[1]==proj.tests[1]&&  
        tests[2]==proj.tests[2];  
}
```

## 2. Student Class Methods

**NOTE:** When writing your code, the only library methods you may use are the default constructor of the `ArrayList` class, the `ArrayList` `add` method, and the `ArrayList` `size` method.

- a. **compareTo** – If the current object has less project than the argument return -1, if more return a 1, and if they both have an equal number of projects, return a 0.

```
public int compareTo(Student s) {
    if (proj.size() < s.proj.size()) {
        return -1;
    } else if (proj.size() == s.proj.size()) {
        return 0;
    } else {
        return 1;
    }
}
```



- b. **makeList** – This method takes in an `ArrayList` of `Students` called `s`. You can assume it has at least one student. The method returns the number of total CS tests (i.e. the sum of all 3 test type) per student in an `ArrayList` of `Integers`. For example, looking at the last line of the driver output, you will see the last element in the return value is a 46. That is because the last student in the sorted `ArrayList` (with `s1Proj`) had 2 CS projects. The first had 11 public and 12 release, and the second had 10 public, 10 release, and 3 secret. So the sum is  $11+12+10+10+3=46$ .

```
public static ArrayList<Integer> makeList(ArrayList<Student> s) {  
    ArrayList<Integer> myList = new ArrayList<Integer>();  
    for (Student st: s) //for each student in argument  
    {  
        int sum =0;  
        for (Project p: st.proj) //for each proj of current student  
        {  
            if(p instanceof CSProject) //is it a cs proj?  
            {  
                CSProject currentCSproj = (CSProject)p;  
                sum += currentCSproj.getPublic() +  
                    currentCSproj.getRelease() +  
                    currentCSproj.getSecret();  
                //sum of 3 test types  
            }  
        }  
        myList.add(sum);  
    }  
    return myList;  
}
```

**Extra Page If You Need It**

**Last Page**