



# University of Maryland College Park

## Department of Computer Science

### CMSC132 Fall 2019

#### Exam #2

FIRSTNAME, LASTNAME (PRINT IN UPPERCASE):

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 200 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 (Algorithmic Complexity)	18	
#2	Problem #2 (Nested Types)	20	
#3	Problem #3 (Miscellaneous)	46	
#4	Problem #4 (Class Implementation)	116	
<b>Total</b>	<b>Total</b>	<b>200</b>	



## Problem #1 (Algorithmic Complexity)

1. (3 pts) List the following Big O expressions in order of asymptotic complexity (lowest complexity first).

$O(n \log(n))$

$O(n^n)$

$O(n!)$

$O(n^2)$

$O(1)$

2. (3 pts) Indicate the complexity (Big O) for an algorithm whose running time increases by a constant when input size doubles.

3. (3 pts) Indicate the complexity (Big O) for an algorithm whose running time doubles as input size doubles.

4. (3 pts) Indicate the complexity (Big O) for an algorithm whose running time quadruples as input size doubles.

5. (3 pts) Big O results are only valid for: (Circle your answer)

- a. Small values of  $n$
- b. Big values of  $n$
- c. Any value of  $n$  between 1 (inclusive) and a 1000 (inclusive)
- d. For any value of  $n$

6. (3 pts) Indicate the algorithm complexity of the following expression:  $7n^2 + n \log(n) + 8n$

## Problem #2 (Nested Types)

1. (2 pts) For the compiler to understand lambda expressions, the interface used for the lambda expression must have:

- a. Only one abstract method.
- b. Only two abstract methods.
- c. At least one abstract method.
- d. None of the above.

2. (18 pts) The **EvalExpr** interface is defined below.

```
public interface EvalExpr {  
    public int eval(int a, int b);  
}
```

- a. (6 pts) Using a **lambda** expression, initialize the variable **prod** with an object that implements the **EvalExpr** interface. The **eval** method returns the product of the parameter values. For example, calling **prod.eval(3, 4)** will return **12**.

**EvalExpr prod =**

DirectoryId:

- b. (12 pts) Using **anonymous** class syntax, initialize the variable **sum** with an object that implements the **EvalExpr** interface. The **eval** method returns the sum of the parameter values. For example, calling **sum.eval(3, 4)** will return 7.

**EvalExpr sum =**

### **Problem #3 (Miscellaneous)**

1. (3 pts) Define an interface called **Transferable** that can be use to implement the marker design pattern.

2. (18 pts) The **Car** class extends the **Vehicle** class. Which of the following are valid (will compile)? Circle your answer.

- a. `ArrayList<?> a = new ArrayList<Vehicle>();` // VALID / INVALID
- b. `ArrayList<Object> b = new ArrayList<Vehicle>();` // VALID / INVALID
- c. `ArrayList<Vehicle> c = new ArrayList<Car>();` // VALID / INVALID
- d. `ArrayList<Car> d = new ArrayList<Vehicle>();` // VALID / INVALID
- e. `ArrayList<? extends Vehicle> e = new ArrayList<Vehicle>();` // VALID / INVALID
- f. `ArrayList<? extends Vehicle> f = new ArrayList<Car>();` // VALID / INVALID
- g. `Object[] g = new Vehicle[10];` // VALID / INVALID
- h. `Vehicle[] h = new Car[10];` // VALID / INVALID
- i. `ArrayList<int> i = new ArrayList<int>();` // VALID / INVALID

3. (3 pts) Which component(s) of the Model View Controller paradigm did we provide for the Clear Cell Game project?

4. (3 pts) When the following JUnit test is executed:

```
public void addTest() {  
    int x = 3;  
    System.out.println(x + 10);  
}
```

- a. It is considered successful (you will see a green bar on the JUnit area).
- b. It is considered a test that failed (you will see a brown / red bar on the JUnit area).
- c. None of the above.

5. (6 pts) What is the output of the following program?

```
public class Building {
    private int size;
    static {
        System.out.println("Cat");
    }

    public Building(int size) {
        this.size = size;
        System.out.println("Dog " + size);
    }

    {
        System.out.println("Bird");
    }

    public static void main(String[] args) {
        Building building1 = new Building(1);
        Building building2 = new Building(2);
    }
}
```

6. (13 pts) Transform the following class into a generic class so the **Box** class can store and retrieve any kind of objects from the **items** array instead of just **String** objects. Feel free to edit / cross out the code.

```
public class Box    {
    private String[] items;
    private int pos = 0;
    public Box()    {
        items = new String[4];
    }

    public String getFirst()    {
        return items[0];
    }

    public void add(String elem)    {
        items[pos++] = elem;
    }
}
```

DirectoryId:

## Problem #4 (Class Implementation)

This problem relies on the partial implementation of the **App** and **Phone** classes below. A **Phone** keeps track of how many apps have been installed by using the array **apps** and the instance variable **installedApps**. We have provided a driver and the corresponding output that illustrates the functionality of the code you are expected to implement. Feel free to ignore it if you know what to implement. **You may not add any instance variables, static variables nor methods to the App or Phone classes.**

```
public class App {
    private String name;
    private int cost;

    public App(String name, int cost) {
        this.name = name;
        this.cost = cost;
    }

    public int getCost() { return cost; }

    public void setCost(int cost) { this.cost = cost; }

    public String toString() {
        return "App [name=" + name + ", cost=" + cost + "]";
    }
}

public class Phone {
    private App[] apps;
    private int installedApps;
    private boolean reversed;

    public Phone(int maxApps) {
        apps = new App[maxApps];
        installedApps = 0;
        reversed = false;
    }

    public void flipReversed() {
        reversed = !reversed;
    }

    public String toString() {
        String answer = "";

        for (int i = 0; i < installedApps; i++) {
            answer += apps[i] + "\n";
        }
        return answer;
    }
}
```

```
/* Driver */
Phone iphone = new Phone(10);
App app1 = new App("Map", 2);
App app2 = new App("Tetris", 3);
App app3 = new App("Calendar", 1);
iphone.add(app1).add(app2).add(app3);

Iterator<App> it = iphone.iterator();
while (it.hasNext()) {
    System.out.println(it.next());
}

AppComparator appComparator = new AppComparator();
System.out.println(appComparator.compare(app1, app2) < 0 ? "Yes" : "No");
iphone.flipReversed();

Iterator<App> it2 = iphone.iterator();
while (it2.hasNext()) {
    System.out.println(it2.next());
}
```

```
/* Driver Output */
App [name=Map, cost=2]
App [name=Tetris, cost=3]
App [name=Calendar, cost=1]
Yes
App [name=Calendar, cost=1]
App [name=Tetris, cost=3]
App [name=Map, cost=2]
```

1. (2 pts) Complete the first line of the **App** class declaration so the class can have a **clone()** method.

**public class App**

{

2. (10 pts) **App class clone method** – Implement a public **clone()** method for the **App** class. Modifications to the object returned by the **clone()** method will not affect the original (this will define the kind of copy you need to make). Declare (instead of using try / catch block) the exception that might be thrown by this method.

3. (18 pts) **AppComparator class** – Implement a class called **AppComparator** that implements the **Comparator** interface. Apps will be compared based on cost. If we were to sort apps using this comparator, they will be sorted in increasing cost value.

4. (2 pts) Complete the first line of the **Phone** class declaration so we can use the *for each* (enhanced for loop) with an instance of this class.

**public class Phone**

{

**DirectoryId:**

5. **(16 pts) Phone class add method** – Define an **add** method that adds a **copy** of an **App** parameter (called **app**) to the **apps** array (if there is space in the array). Use the **clone()** method to generate the copy. Declare (instead of using try / catch block) the exception that might be thrown by this method. The method will return a reference to the current object. Make sure you update the **installedApps** instance variable, accordingly. No processing will take place if there is no space in the array (just return a reference to the current object).

6. **(58 pts) MyIterator inner class**– Define an **inner** class (present inside of the **Phone** class) named **MyIterator**. This class will implement the **Iterator** interface. If **reversed** (Phone instance variable) is false, apps from the **apps** array will be returned by the iterator in the order in which they were added to the array; otherwise, they will be returned in reversed order (last one added will be returned first). You do not need to implement the iterator **remove** method. **Do not use an anonymous inner class and do not use any additional data structure (e.g., stack, ArrayList, etc.) to implement the iterator functionality.**



7. (10 pts) **Phone class iterator() method** – Define an **iterator()** method that returns an instance of the **MyIterator** class.

DirectoryId:

**EXTRA PAGE IN CASE YOU NEED IT**

**LAST PAGE**