



University of Maryland College Park

Dept of Computer Science

CMSC131 Fall 2017

Midterm II Key

Last Name (PRINT): _____

First Name (PRINT): _____

University Directory ID (e.g., umcpturtle)_____

I pledge on my honor that I have not given or received any unauthorized assistance on this examination.

Your signature: _____

Grader Use Only

#1	Problem #1 (General Questions)	(20)	
#2	Problem #2 (Memory Map)	(30)	
#3	Problem #3 (Class Definition)	(95)	
#4	Problem #4 (String Manipulation)	(55)	
Total	Total	(200)	

Problem #1 (General Questions)

1. (3 pts) If the garbage collector stops working which of the following can eventually be exhausted if we continue creating objects. Circle all that apply.
- a. Stack
 - b. Heap
 - c. Area where static variable resides.
 - d. None of the above.

Answer: b.

2. (3 pts) A method should be defined as static if (circle all that apply):
- a. Makes no reference to instance variables.
 - b. Makes a reference to at least one instance variable.
 - c. Makes a reference to static method.
 - d. None of the above.

Answer: a.

3. (3 pts) Which of the following represents “no address”? Circle all that apply.
- a. 0
 - b. null
 - c. false
 - d. None of the above.

Answer: b.

4. (3 pts) How many objects are present in the following code fragment?

```
StringBuffer b;  
int x = 10;
```

Number of Objects: ____

Answer: 0

5. (3 pts) When is the code associated with a finally block executed?
- a. Only when the exception occurs.
 - b. Always
 - c. Only if no exception occurs.
 - d. None of the above.

Answer: b.

6. (3 pts) What is the actual task a constructor method performs?

- a. Creating the object in the stack.
- b. Initializing the object.
- c. Moving the object from the stack to the heap.
- d. None of the above.

Answer: b.

7. (2 pts) Which of the following applies to the “this” reference?

- a. It is a reference to the current object.
- b. Can be used by both static and non-static methods.
- c. It is initialized for you.
- d. None of the above.

Answer: a. and c.

Problem #2 (Memory Map)

On the next page draw a memory diagram showing both the stack and the heap at the moment this program reaches the point identified by **/* HERE */**

```
public class Fruit {
    private String name;
    private int size;

    public Fruit(String nameIn, int sizeIn) {
        name = new String(nameIn);
        size = sizeIn;
    }

    public void increaseSize(int x) {
        size += x;
    }

    public String toString() {
        return name + " " + size;
    }
}

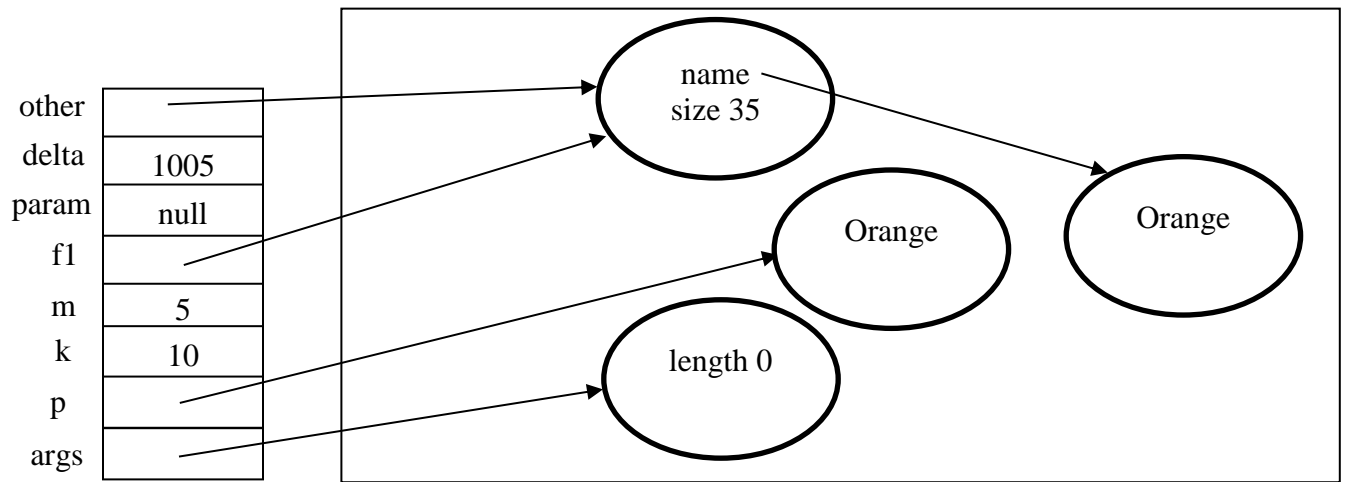
public class Driver {
    public static void task(Fruit param, int delta) {
        param.increaseSize(delta);
        delta += 1000;
        Fruit other = param;
        other.increaseSize(20);
        param = null;
        /* HERE */
    }

    public static void main(String[] args) {
        String p = "orange";
        int k = 10, m = 5;

        Fruit f1 = new Fruit(p, k);
        task(f1, m);
    }
}
```

Stack

Heap



Stack Bottom

Problem #3 (Class Definition)

Implement a class named **Party** according to the specifications below. The class allow us to define a party with a maximum number of guests and to add guests to the party. Below we have provided a driver that illustrates the functionality associated with the **Party** class. Feel free to ignore it if you know what to implement. Make sure you avoid code duplication.

1. The class has the following private instance variables:
 - a. **name** → String variable representing a party's name.
 - b. **maxGuests** → Maximum number of guests allowed in the party.
 - c. **guests** → StringBuffer used to store the names of guests to the party.
 - d. Feel free to add any other private instance variables you understand you need.
2. All the methods in the class are **public** and **non-static** unless otherwise specified.
 - a. **Constructor** → Takes a string (name) and maximum number of guests (maxGuests) as parameters. The current object is initialized based on these parameters. The parameters must be named after the instance variables. Feel free to provide any other initialization that is needed.
 - b. **Constructor** → Takes a string (name) as parameter. It initializes the party with the specified name and with 100 as the maximum number of guests.
 - c. **addGuest** → Takes a string (guestName) as parameter. If the parameter is different than null, and the current number of guests have not reached the maximum possible, then the name will be added to the StringBuffer holding the names of guests. A reference to the current object will be returned.
 - d. **getTotalParties** → **Static method** that returns the total number of **Party** objects created. Feel free to add any variable you might need and to modify any method so this method can return the appropriate count.
 - e. **toString()** → Returns a string with the following format:

Name: <NAME>, NumberGuests: <NUMBER_GUESTS>, Guests: <GUESTS>

Where <NAME>, <NUMBER_GUESTS> and <GUESTS> correspond to the party's name, the number of guests, and a string with the guests, respectively. See sample driver below for an example.

Remember that the StringBuffer **append()** method allows you to add a String and the **toString()** method returns a string with the StringBuffer contents.

Driver

```
Party party;  
party = new Party("Halloween", 50);  
System.out.println(party.addGuest("Ann").addGuest("John"));  
  
Party party2 = new Party("EndOfSemester");  
System.out.println(party2);  
System.out.println("Total: " + Party.getTotalParties());
```

Output

```
Name: Halloween, NumberGuests: 2, Guests: AnnJohn  
Name: EndOfSemester, NumberGuests: 0, Guests:  
Total: 2
```

Answer:

```
public class Party {
    private String name;
    private StringBuffer guests;
    private int maxGuests, currGuests;
    private static int totalParties = 0;

    public Party(String name, int maxGuests) {
        this.name = name;
        this.maxGuests = maxGuests;
        guests = new StringBuffer();
        totalParties++;
    }

    public Party(String name) {
        this(name, 100);
    }

    public Party addGuest(String guestName) {
        if (guestName != null && currGuests < maxGuests) {
            guests.append(guestName);
            currGuests++;
        }
        return this;
    }

    public static int getTotalParties() {
        return totalParties;
    }

    public String toString() {
        String answer = "Name: " + name + ", NumberGuests: " + currGuests;
        answer += ", Guests: " + guests.toString();

        return answer;
    }
}
```

Problem #4 (String Manipulation)

For this problem you will implement a static method called **getStrength** that evaluates how strong a password is and returns one of three strings ("Weak", "Medium", "Strong"). The following definitions are associated with this problem:

- Digit** - Character represents a digit (value between 0 and 9, including 0 and 9). The Java method `Character.isDigit()` determines whether a character is a digit (e.g., `Character.isDigit('2')` returns true).
- Alphabetic** - Character represents a letter (either uppercase or lowercase). The Java method `Character.isAlphabetic()` determines whether a character is alphabetic (e.g., `Character.isAlphabetic('@')` returns false).

The password types we will have are:

- Strong** - The password has at least one digit, at least one uppercase letter and at least one non-alphabetic character.
- Medium** - The password has at least one digit and at least one non-alphabetic character.
- Week** - The password does not have digits, uppercase letters and non-alphabetic characters.

If the **passwd** parameter value is null, the method will throw an `IllegalArgumentException` with the message "Invalid password" and no further computation will take place. Remember that you can compute the length of a string by using the **length()** method and the character at a particular index position by using the **charAt()** method.

The following driver and expected output illustrates the result of calling the method you are expected to implement. Feel free to ignore the driver if you know what to implement.

<u>Driver</u>	<u>Output</u>
<code>System.out.println(getStrength("boat"))</code>	Weak
<code>System.out.println(getStrength("safebo@t4"))</code>	Medium
<code>System.out.println(getStrength("sAfebo@t4"))</code>	Strong
<code>System.out.println(getStrength("bo@t"))</code>	Weak
<code>System.out.println(getStrength("bo4t"))</code>	Weak
<code>System.out.println(getStrength("Boat"))</code>	Weak

One Possible Answer:

```
public static String getStrength(String passwd) {
    if (passwd == null) {
        throw new IllegalArgumentException("Invalid password");
    }

    boolean foundDigit = false, foundUpper = false;
    boolean foundNonAlpha = false;

    for (int i = 0; i < passwd.length(); i++) {
        char c = passwd.charAt(i);
        if (!Character.isAlphabetic(c)) {
            if (Character.isDigit(c)) {
                foundDigit = true;
            } else {
                foundNonAlpha = true;
            }
        } else {
            if (Character.isUpperCase(c)) {
                foundUpper = true;
            }
        }
    }

    if (foundDigit && foundUpper && foundNonAlpha) {
        return "Strong";
    } else if (foundDigit && foundNonAlpha) {
        return "Medium";
    } else {
        return "Weak";
    }
}
```