



# University of Maryland College Park

## Dept of Computer Science

### CMSC131 Fall 2015

### 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: \_\_\_\_\_

#### Instructions

- This exam is a closed-book and closed-notes exam.
- Total point value is 200 points.
- The exam is a 50 minutes exam.
- Please use a pencil to complete the exam.
- WRITE NEATLY.
- There are three problems in the exam.
- Multiple choice questions can have more than one valid/correct answers.
- You do not need any import statements for coding questions.
- You must write code that is efficient and that avoids code duplication.

#### Grader Use Only

#1	Problem #1 (General Questions)	(80)	
#2	Problem #2 (Memory Map)	(40)	
#3	Problem #3 (Class Definition)	(80)	
<b>Total</b>	Total	(200)	

## Problem #1 (General Questions)

1. (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.
2. (3 pts) Which of the following takes care of an object that is no longer referenced by any variables?
  - a. The stack
  - b. The heap
  - c. The garbage collector ✓
  - d. The constructor
3. (3 pts) What is the output of the following code fragment?

```
for (int i = 1; i <= 8; i += 3) {  
    if (i == 4) {  
        continue;  
    }  
    System.out.println(i);  
}
```

Answer:

1  
7

4. (3 pts) What is the output of the previous code fragment if we replace **continue;** with **break;** ?

Answer:

1

5. (3 pts) How many objects exist in the following code fragment?

```
String a, b;  
int c;
```

Answer: 0

6. (3 pts) The following code compiles. Do you see any problems with the code? Write NONE if no problems or invalid operations are present. You can assume the method is in a class that compiles.

```
public void check(double val, int x) {
    if (x > 0) {
        if (val == 4.5) {
            System.out.println("expected");
        }
    }
}
```

Answer: We should not be comparing floating point values.

7. (3 pts) How many default constructors do we have in the following class?

```
public class Apple {
    private String flavor;
}
```

- a. 0
- b. 1 ✓
- c. The above class does not compile.
- d. None of the above

8. (6 pts) What is the output of the following program? If an exception is thrown indicate why.

```
public class Values {
    public int a;
    public boolean b;
    public String c;

    public Values() {
        String d = c;
        System.out.println(a + ", " + b + ", " + d);
    }

    public static void main(String[] args) {
        Values v = new Values();
    }
}
```

Answer: 0, false, null

9. (8 pts) Rewrite the body of the following method so it uses a single statement and the ternary operator(? :).

```
public String original(double m) {
    String a = null;

    if (m > 4.5) {
        a = "valid";
    }

    return a;
}
```

Answer:

```
return m > 4.5 ? "valid" : null;
```

10. (18 pts) Rewrite the following code using a switch statement.

```
String p;

if (x == 'a' || x == 'A') {
    p = "Al";
} else if (x == 'f') {
    p = "Fan";
} else {
    p = "Bob";
}
```

Answer:

```
switch(x) {
    case 'a':
    case 'A':
        p = "Al";
        break;
    case 'f':
        p = "Fal";
        break;
    default:
        p = "Bob";
}
```

The **Breakfast** class will be used for the questions that follow.

```
public class Breakfast {
    private int calories;
    public static final String BEST = "Fruits and Oatmeal";

    public Breakfast(int caloriesIn) {
        calories = caloriesIn;
    }

    public static void increaseCalories(Breakfast breakfast, int delta) {
        breakfast.calories += delta;
    }

    public int getCalories() {
        return calories;
    }

    public void increaseCalories(int delta) {
        /* The body of this method is the answer to question #11 */
    }

    public static void main(String[] args) {
        /* CODE HERE */
    }
}
```

11. (6 pts) Complete the implementation of the Breakfast class non-static method `increaseCalories` by calling the static `increaseCalories` method.

```
public void increaseCalories(int delta) {
}

```

Answer: `increaseCalories(this, delta);`

12. (10 pts) Which of the following statements will compile if placed in the `/* CODE HERE */` section of the main method above? Circle those that will compile.

- a. `Breakfast.calories = 10;`
- b. `Breakfast.BEST = "cheese";`
- c. `System.out.println(Breakfast.getCalories());`
- d. `System.out.println(new Breakfast(10).getCalories());`
- e. `System.out.println(new Breakfast(10).toString());`

Answer: d. and e. should be the only ones circled.

13. (11 pts) Define an equals method (as described in lecture) that compares two Breakfast object. Two objects are considered the same if they have the same number of calories.

Answer:

```
public boolean equals(Breakfast b) {  
    if (b == null) {  
        return false;  
    } else if (this == b) {  
        return true;  
    } else {  
        return calories == b.calories;  
    }  
}
```

## 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 Dvd {
    private String make;
    private int capacity;

    public Dvd(String makeIn, int capacityIn) { make = new String(makeIn); }
    public String toString() { return make + " " + capacity; }
}

public class Tv {
    private int serialNo;
    private Dvd dvd;

    public Tv(int serialNoIn, Dvd dvdIn) { serialNo = serialNoIn; dvd = dvdIn; }
    public void setDvd(Dvd dvdIn) { dvd = dvdIn; }
    public String toString() { return serialNo + " " + dvd; }
}

public class DriverMap {

    public static void process(Tv p, int m) {
        String arp = "Arp";
        p.setDvd(new Dvd(arp, m));
        p = null;
        m = 0;
        /* HERE */
    }

    public static void main(String[] args) {
        int cap = 50, serialNo = 123, newcap = 456;
        String make = "Ony";

        Dvd dvd1 = new Dvd(make, cap);
        Tv tv1 = new Tv(serialNo, dvd1);

        process(tv1, newcap);
    }
}
```



### Problem #3 (Class Definition)

Implement a class named **Blog** according to the specifications below. Your implementation must be efficient and must avoid code duplication. Below we have provided a driver that illustrates part of the functionality associated with the Blog class. You can ignore this driver if you know what to implement. Notice you do not need to provide a toString() method.

1. The class has the following private instance variables:
  - a. **name** → String variable representing a person's name.
  - b. **creditCard** → String variable representing a person's credit card number.
  - c. **entries** → **StringBuffer** used to store blog entries.
2. All the methods in the class are **public** and **non-static**. A description of each method follows:
  - a. **setCreditCard** → Takes as parameter a string. The method will initialize the creditCard field with the parameter value if the credit card is valid. A valid credit card is one that is different than null and that only has digits. If an invalid credit card parameter is provided the creditCard instance variable will not be modified and an IllegalArgumentException with the message "Invalid card" will be thrown. Remember you can use Character.isDigit() to verify whether a character is a digit. The string functions length() and charAt() can be helpful during the implementation of this method.

Answer:

```
public void setCreditCard(String creditCard) {
    boolean invalid = true;

    if (creditCard != null) {
        invalid = false;
        for (int i = 0; i < creditCard.length(); i++) {
            if (!Character.isDigit(creditCard.charAt(i))) {
                invalid = true;
                break;
            }
        }
    }

    if (invalid) {
        throw new IllegalArgumentException("Invalid card");
    } else {
        this.creditCard = creditCard;
    }
}
```

- b. **Constructor** → Takes two string parameters where the first one represents the person's name and the second a credit card. This method calls the **setCreditCard** method to initialize the creditCard. No instance variable will be initialized if an invalid credit card has been provided. In addition, if an invalid credit card has been provided the message associated with the exception thrown by setCreditCard must be displayed. Notice you must retrieve the error message string from the exception object. If a valid credit card has been provided the name instance variable will be initialized with the provided parameter value and a StringBuffer object will be associated with the **entries** instance variable.

Answer:

```
public Blog(String name, String creditCard) {
    try {
        setCreditCard(creditCard);
        this.name = name;
        this.entries = new StringBuffer();
    } catch (IllegalArgumentException e) {
        System.out.println(e.getMessage());
    }
}
```

- c. **Copy Constructor** → Implement it by calling the constructor defined above (i.e., using "this").

Answer:

```
public Blog(Blog blog) {
    this(blog.name, blog.creditCard);
}
```

- d. **add** → Adds an entry to the StringBuffer if the credit card instance variable has a value other than null. It returns a reference to the current object.

Answer:

```
public Blog add(String entry) {
    if (creditCard != null) {
        entries.append(entry + "\n");
    }

    return this;
}
```