University of Maryland College Park

Dept of Computer Science

CMSC131 Spring 2011

Midterm II Key

First Name (PRINT): ______________________________

Last Name (PRINT): ______________________________

University ID: ______________________________

Section/TAName: ______________________________

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 100 points.*
- *The exam is a 50 minutes exam.*
- *Please use a pencil to complete the exam.*
- ***WRITE NEATLY**. If we cannot understand your answer, we will not grade it (i.e., 0 credit).*

Grader Use Only

#1	Problem 1 (General Questions)	(16)	
#2	Problem 2 (Memory Map)	(8)	
#3	Problem 3 (Parsing/Exception)	(20)	
#4	Problem 4 (Class Definition)	(56)	
Total	Total (100)	(100)	

Problem 1 (16 pts)

1. (1 pt) Name one class discussed in class that is immutable.

   Answer: String

2. (1 pt) When should we define a method as static?

   Answer: If it makes no reference to instance variables.

3. (1 pt) What is the default value of reference instance variables of a class?

   Answer: null

4. (2 pts) When is space for a local integer variable allocated and when is it recovered?

   Answer: Allocated: when the method is called; recovered: when the method finishes

5. (2 pts) Why do we never use == to compare floating point numbers?

   Answer: Because floating point numbers are approximations

6. (1 pt) When is the finally block associated with exceptions executed?

   Answer: Always

7. (1 pt) In which area of memory are objects created?

   Answer: heap

8. (1 pt) When should we define a constant as a static constant? In other words, when should we use **static final** vs. **final** while defining a constant?

   Answer: We should use static when all instances of the class will use the same constant value

9. (6 pts) Based on the following class, indicated whether the statements below are valid or invalid. Circle your answer.

```
public class Computer {
        private String make;

        public Computer(String makeIn) { make = makeIn; }
        public void setMake(String makeIn) { make = makeIn; }
        public static void info() { System.out.println("Computer Sys"); }
}
```

   Answer/Grading: (1 pt) each

   a. Computer c1 = new Computer("sun");   VALID
      c1.setMake("mars");

b.	Computer c2 = null;
	c2.setMake("moon");
		INVALID

c.	Computer c3 = new Computer("earth");
	Computer.setMake("saturn");
		INVALID

d.	Computer c4 = new Computer("venus");
	c4.info();
		VALID

e.	Computer c5 = new Computer("jupiter");
	c5.make = "uranus";
		INVALID

f.	Computer.info();	VALID

Problem 2 (8 pts)

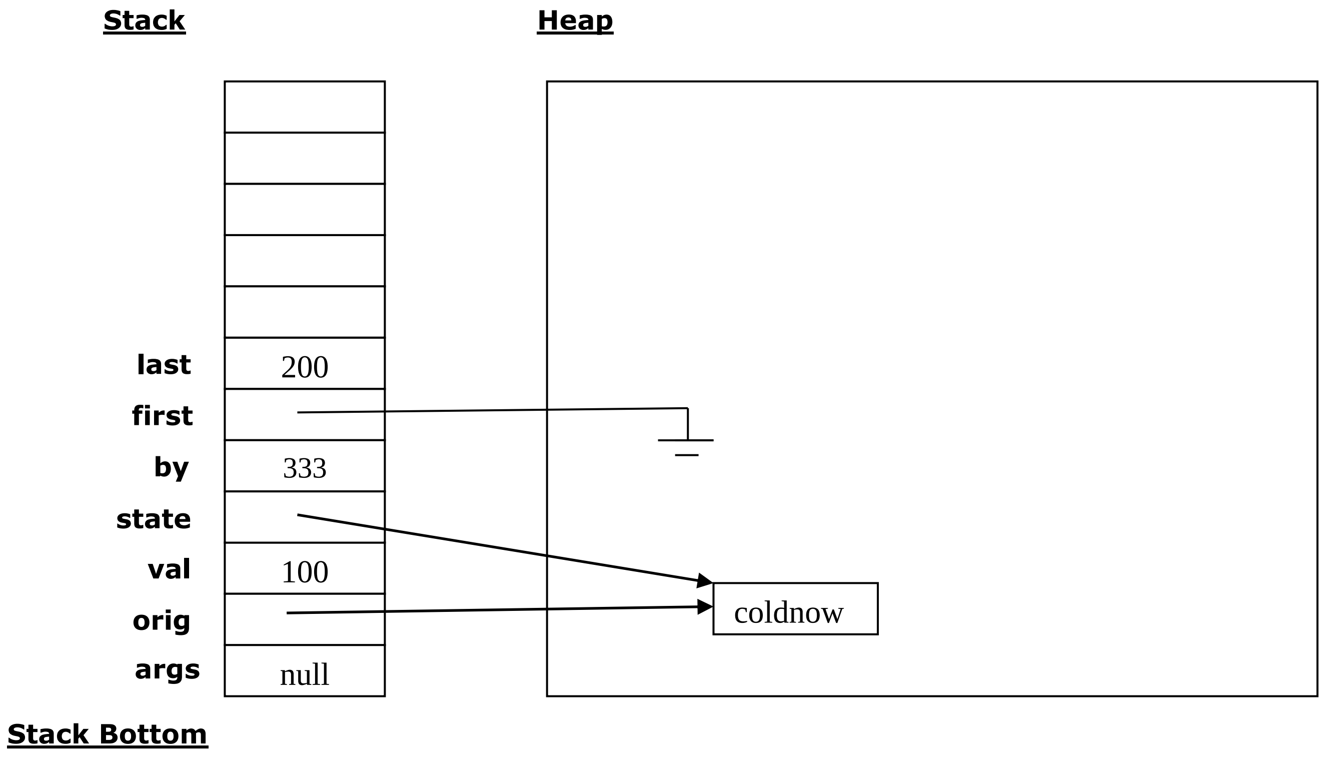
Draw a memory diagram showing both the stack and the heap at the moment this program reaches the point marked **/* HERE */**

```
public class MemoryMap {
      public static void filter(StringBuffer state, int by) {
            state.append("now");
            by = 333;
            process(state, by);
      }

      public static void process(StringBuffer first, int last) {
            first = null;
            last = 200;
            /* HERE */
      }

      public static void main(String[] args) {
            StringBuffer orig = new StringBuffer();
            orig.append("cold");

            int val = 100;
            filter(orig, val);
      }
}
```



Problem 3 (20 pts)

The method **getMemorySize** takes a string as a parameter that represents the amount of memory present in a high definition camera. The string always starts with the letters **H** and **D,** and is followed by a number. The following are some examples of valid strings: “HD100”, “HD2000”, “HD2”, “HD20500”, etc. The method returns the number (an integer) that follows “HD” in the string. For this problem you can assume the caller will provide a valid string or null. If null is provided, the method will throw an IllegalArgumentException with the message “Invalid String”. For this problem:

1. Implement the getMemorySize method.
2. Complete the main method provided below so the exception is handled and the message “Invalid String” is printed (using System.out.println) when the exception takes place.
3. Remember that the method charAt() returns the character at a particular position in a string.

```
public class Utilities {
       static Scanner sc = new Scanner(System.in);

       public static void main(String[] args) {
              // You must complete so exception is handled

              String val = sc.nextLine();

              System.out.println(getMemorySize(val));

       }

       public static int getMemorySize(String memType) {
              // You must write
```

(16 pts) getMemorySize

Answer:

```
public static int getMemorySize(String memType) {
	if (memType == null)
		throw new IllegalArgumentException("Invalid String");
	else {
		String sizeStr = "";
		for (int idx = 2; idx < memType.length(); idx++) {
			sizeStr += memType.charAt(idx);
		}
		return Integer.parseInt(sizeStr);
	}
}
```

(4 pts) main

Answer:

```
public static void main(String[] args) {
	try {
		String val = sc.nextLine();
		System.out.println(getMemorySize(val));
	} catch (IllegalArgumentException e) {
		System.out.println("Invalid String");
	}
}
```

<u>Problem 4 (56 pts)</u>

Implement a class named **BillBoard** according to the specifications below.

1. The class has the following private instance variables:
    a. **message** → String variable
    b. **cost** → integer variable
2. All the methods in the class are public, except the method **validMessage** that is private. A description of each method follows:
    a. **validMessage** → Takes as parameter a string. The method returns true if the string parameter represents a valid message. A valid message is different from null and it has at least 3 characters.
    b. **Constructor** → Takes two parameters: a string and an integer. If the string parameter represents a validMessage, the instance variables will be initialized using the parameter values. You must use the previous **validMessage** method to validate the string. If the string parameter is invalid, the message instance variable will be initialized to "NOMessage" and the cost to 10.
    c. **Constructor** → Takes a string as a parameter. It initializes the cost to 20. You **MUST** use "this" in order to call the constructor you previously defined otherwise you will not get any credit.
    d. **Copy Constructor**
    e. **getMessage** → Get method for the message instance variable.
    f. **toString()** → Returns a string with the message followed by the cost (separated by one space).
    g. **equals()** → Two objects are equal if they have the same message. The method should return false if null is provided as a parameter value.

Answer/Grading:

```
class BillBoard { }
        private String message;
        private int cost;


        private static boolean validMessage(String message) {
                if (message == null || message.length() <= 3)
                        return false;
                return true;
        }

        public BillBoard(String message, int cost) {
                if (validMessage(message)) {
                        this.message = message;
                        this.cost = cost;
                } else {
                        this.message = "NOMessage";
                        this.cost = 10;
                }
        }

        public BillBoard(String message) {
                this(message, 20);
        }
```

```
public BillBoard(BillBoard b) {
        this.message = b.message;
        this.cost = b.cost;
}


public String getMessage() {
        return message;
}

public String toString() {
        return message + " " + cost;
}

public boolean equals(BillBoard board) {
        if (board == null)
                return false;
        return message.equals(board.message);
}
```