University of Maryland College Park
Dept of Computer Science
CMSC132, Midterm #1 Key
Summer 2008

First Name (PRINT): _____

Last Name (PRINT): _____

University ID: _____

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, 50 minutes exam.*
- *Please use a pencil to complete the exam.*
- ***PUNT RULE:** For any question, you may write PUNT, and you will get ¼ of the points for the question (rounded down). If you feel totally lost on a question, you are encouraged to punt rather than write down an incorrect answer in hopes of getting some partial credit.*
- ***WRITE NEATLY**. If we cannot understand your answer, we will not grade it (i.e., 0 credit).*

Grader Use Only

#1	Algorithmic Complexity		(18)
#2	Program Correctness		(10)
#3	Hashing		(8)
#4	Language Features		(24)
#5	Sets and Maps		(20)
#6	Linear Data Structures		(20)
Total			(100)

Problem 1 (18 pts) Algorithmic Complexity

a. (6 pts) Calculate the asymptotic complexity of the code snippets below (using big-O notation) with respect to the problem size **n**.

1.
```
for (int i=0; i<=n-3; i++) {
        System.out.println("Hello");
}
```
f(**n**) = O(**n**)

2.
```
for (i=1; i<=n; i++) {
        for (k= 0; k<=n/2; k++) {
                System.out.println("Hello");
        }
}
```
f(**n**) = O($\mathbf{n^2}$)

3.
```
for (int i=n-1; i<=n; i++) {
        for (int j=1; j<=n; j=j*2) {
                System.out.println("Hello");
        }
}
```
f(**n**) = O(**log(n)**)

b. (4 points) Give the asymptotic bound of the following functions:

1. $\mathbf{n}^3 + \mathbf{n}\log(\mathbf{n})$ f(**n**) = O($\mathbf{n^3}$)

2. $2\mathbf{n}^4 - 500\mathbf{n} - \mathbf{n}^2$ f(**n**) = O($\mathbf{n^4}$)

c. (2 pts) List the following big-O expressions in order of asymptotic complexity (lowest complexity first) where k represents a constant.

O(nlog(n)) $O(k^{\mathbf{n}})$ $O(n^n)$ O(log(n))

Answer: O(log(n)) O(nlog(n)) $O(k^{\mathbf{n}})$ $O(n^n)$

d. (2 pts) What is the complexity of computing a[i] for an array of length n?

Answer: O(1)

e. (2 pts) What is the asymptotic complexity of finding in an element in a sorted array if we use binary search?

Answer: O(log(n))

f. (2 pts) Give the complexity of an algorithm for problem size **n** whose running time:

i. Increases by 2 when **n** doubles O(**log(n)**)
ii. Is unchanged when **n** triples O(**1**)

Problem 2 (10 pts) Program Correctness and Exceptions

1. (2 pts) **F** → 60% code coverage implies 40% of the code is incorrect.
2. (2 pts) **F** → Catching checked exceptions is optional.
3. (2 pts) **F** → Checked exceptions represent typical errors a program can ignore.
4. (2 pts) **T** → Exceptions are implicitly propagated to the caller.
5. (2 pts) **F** → Catching unchecked exceptions is mandatory.

Problem 3 (8 pts) Hashing

1. (1 pt) **F** → The default Object class **hashCode** and **equals** methods violate the Java Hash code contract.
2. (1 pt) **T** → If the hashCode() values of two objects are different then the objects cannot be equal.
3. (1 pt) **F** →The hashCode() method is incorrect if two objects have the same hashCode().
4. (1 pt) **T** →Through hashing it is possible to access elements in O(1) given the appropriate hash function.

5. (4 pts) The Person class is defined as follows:

```
public class Person {
    String name;
    int age;
    int numberOfCars;
}
```

The maximum number of cars a person can have is two. Age values range from 1 up to 100.

a. **F** →A hashCode() method that just returns numberOfCars will be considered an excellent method.
b. **T** →A hashCode() method that just returns age will be considered an reasonable method.
c. **F** →A hashCode() method that returns 0 will be better than returning numberOfCars.
d. **T** →A hashCode() method that returns the sum of the ascii character values in name will be better than returning age.

Problem 4 (24 pts) Java Language Features

1. (2 pts) **F** → Java methods are examples of data abstractions.
2. (2 pts) **T** → Generic classes help detect errors in Java programs.
3. (2 pts) **T** → A private visibility modifier allows us to enforce encapsulation.
4. (2 pts) **F** → A Constructor cannot be defined for an abstract class.
5. (2 pts) When we call the method Collections.sort(L) on the Collection object L the data in L is sorted. Which interface (Comparable or Comparator) is implemented by elements of object L?

   Answer: Comparable

6. (2 pts) Complete the following variable declaration so we have an ArrayList with objects that implement the Comparable interface.

   Answer: ArrayList<Comparable> aList; or ArrayList<**?** extends Comparable>

7. (2 pts) Complete the following variable declaration so we have an ArrayList with objects that belong to the **Reptile** class or are subtypes of this class.

   Answer: ArrayList<**?** extends Reptile> bList;

8. (2 pts) Complete the following variable declaration so we have an ArrayList with objects that can be of any type.

   Answer: ArrayList<?> cList;

9. (8 pts) Make the following class generic so that it can deal with an arbitrary class rather than only Boolean. Feel free to cross out parts of the following code.

   Answer(8 pts):

```
public class MyDS<T> {

    private T [] data;

    private int curr = 0;

    public MyDS(int size) {
        data = (T[]) new Object[size];
    }

    public T first() {
        return data[0];
    }

    public void append(T val) {
        data[curr++] = val;
    }
}
```

Problem 5 (20 pts) Sets and Maps

The **Assistants** class maintains, for a set of courses, the set of TAs for each course.

```
public class Assistants {
        Map<String, Set<String>> map;

        public Assistants() {
                // YOU MUST IMPLEMENT THIS METHOD
        }

        public void addTA(String course, String taName) {
                // YOU MUST IMPLEMENT THIS METHOD
        }

        public void displayTAsPerCourse() {
                // YOU MUST IMPLEMENT THIS METHOD
        }
}
```

1. **(3 pts)** Implement a constructor for Assistants that creates an empty map.

   Answer:

   ```
   map = new HashMap<String, Set<String>>();
   ```

2. **(10 pts)** Implement the **addTA** method that adds a teaching assistant to a specific course. A map entry for the course must be created if one does not exist.

   NOTE: If punting 2 pts credit

   Answer:

   ```
   Set<String> tas = map.get(course);
   if (tas == null) {
         tas = new TreeSet<String>();
         map.put(course, tas);
   }
   tas.add(taName);
   ```

3. **(7 pts)** Implement the **displayTAsPerCourse** method that prints (using System.out.println) the name of a course followed by the TAs for the course.

   Answer:

   ```
   for (String c : map.keySet()) {
         System.out.println(c);
         for (String taName: map.get(c))
               System.out.println(taName);
   }
   ```

Problem 6 (20 pts) Linear Data Structures

Implement the methods below based on the following Java class definitions. You may not add any instance variables, static variables or auxiliary methods to the LinkedList class. In addition, you may not use the Java API LinkedList class.

```
public class LinkedList<T> {
        private class Node {
                private T data;
                private Node next;
                public Node(T data) {
                        this.data = data;
                        next = null;
                }
        }

        private Node head;

        public LinkedList() { /* YOU MUST IMPLEMENT THIS METHOD */ }
        public boolean isEmpty() { /* YOU MUST IMPLEMENT THIS METHOD */ }
        public T removeFirst() { /* YOU MUST IMPLEMENT THIS METHOD */ }
        public ArrayList<T> getList() { /* YOU MUST IMPLEMENT THIS METHOD */ }
}
```

1. (2 pts) Implement a constructor that defines an empty list.
2. (2 pts) Implement the method **isEmpty** that returns true if the list is empty.
3. (4 pts)) Implement the method **removeFirst** that removes the first element from the list and returns that element.
4. (12 pts) Implement the method **getList** that returns an ArrayList with all the elements in the list.

(2 pts)

```
public LinkedList() {
        head = null;
}
```

(2 pts)

```
public boolean isEmpty() {
        return head == null ? true: false;
}
```

(4 pts)

```
public T removeFirst() {
        T temp;
        if (head == null)
                return null;
        else {
                temp = head.data;
                head = head.next;
                return temp;
        }
}
```

(12 pts)

```
public ArrayList<T> getList() {
        ArrayList<T> result = new ArrayList<T>();
        Node temp = head;
        while (temp != null) {
                result.add(temp.data);
                temp = temp.next;
        }
        return result;
}
```