CSE143 Summer 2008 Final Exam — Part A KEY
August 21, 2008

Name : ______________________________

Section (eg. AA) : ________ TA : ______________________________

This is an open-book/open-note exam. Space is provided for your answers. Use the backs of pages if necessary. The exam is divided into 4 questions with the following points:

Question	Points	Score
Linked List	10	
Comparable class	20	
Binary Trees	10	
Inheritance/Casting Question	10	
Total:	50	

Do not begin work on this exam until instructed to do so. Any student who starts early or who continues to work after time is called will receive a 10 point penalty.

The exam is not, in general, graded on style and you do not need to include comments.

Please turn off an cell phones or other devices that might disturb others during the exam. You are NOT to use any electronic devices while taking the test, including calculators. Anyone caught using an electronic device will receive a 10 point penalty.

If you finish the exam early, please hand your exam to the instructor and sit quietly until the exam time is finished.

1. (10 points) **Linked List**

   Write a method removeEveryIthElement for the LinkedIntList we discussed in class. This method takes an integer and modifies the list by removing every ith element, but otherwise keeps the list unchanged. So removeEveryIthElement(3) removes every third element (the 3rd 6th 9th etc.).

   Your method should throw an IllegalArgumentException if the integer is less than 1.

   You may not call any other methods of the LinkedIntList class to solve this problem You may write helper methods to help you solve this problem but other than those you should not call any methods in LinkedIntList. Do not create any additional structured objects (arrays, LinkedIntList, etc) in your solution.

   Examples

   removeEveryIthElement(4) to a list [1 2 3 4 5 6 7 8 9] changes it to [1 2 3 5 6 7 9]
   removeEveryIthElement(2) to a list [77 9 1 2] changes it to [77 1]

Solution:

```
public void removeEveryIthElement(int i) {
  if(i < 1) throw new IllegalArgumentException("i too small");
  front = removeEveryIthElement(1, i, front);
}

public ListNode removeEveryIthElement(int pos,
                                     int i,
                                     ListNode currentNode) {
  if(currentNode == null) return null;
  if(pos == i)
    return removeEveryIthElement(1,i,currentNode.next);
  currentNode.next = removeEveryIthElement(pos+1, i, currentNode.next);
  return currentNode;
}
```

```
//an alternate recursive solution that does not use x = change(x)
public void removeEveryIthElement(int i) {
  if(i < 1) throw new IllegalArgumentException("i too low");
  if(i == 1) {
    front = null;
  } else {
    removeEveryIthElement(i,1,front);
  }
}
public void removeEveryIthElement(int i, int pos, ListNode node) {
  if(node == null) return;
  if(pos + 1 == i && node.next != null) {
    node.next = node.next.next;
    pos = 0;
  }
  removeEveryIthElement(i, pos + 1, node.next);
}

//an alternate iterative solution;  You can see it is less elegant.
public void removeEveryIthElement(int i) {
  ListNode current = front;
  int position = 1;
  if(i < 1) {
    throw new IllegalArgumentException("i too low");
  }
  if(i == 1) {
    front = null;
  } else {
    while(current != null && current.next != null) {
      if(position + 1 == i) {
        current.next = current.next.next;
        position = 0;
      }
      current = current.next;
      position++;
    }
  }
}
```

2. (20 points) **Comparable class**
Imagine a video game that produces scores in the range -5 to 5 (5 is the best possible score, -5 is the worst). Players are ranked according to their all-time highest score. If two players have the same all-time high score, the player who has scored the high score the most times is the higher ranked. Two players who have scored the high score an equal number of times are ranked equally.

For example a player scoring (-5, -4, 4, 4) is ranked higher than a player who scores (4, 3, 3, 3) because player 1 has scored 4 two times.

Write a class ScoreHistory that keeps track of player scores in this game. Include the following functionality:

constructor	ScoreHistory should be constructed with a String representing the player name
addScore(int)	takes an integer parameter representing the score (it should throw an IllegalArgumentException if the score is not in the range -5 to 5)
numberOfPlays()	returns the number of scores that have been added
name()	returns the player name
Comparable	Your class should also implement the Comparable interface. A ScoreHistory is greater than another if the high score is greater. If two ScoreHistory objects have the same high score, ScoreHistory with the greatest number of that score is greater. If both ScoreHistory objects have the same number of the high score, they are equal.

You can use any fields you wish in the ScoreHistory. You can also write any helper methods you need to accompish the described functionality.

Solution:

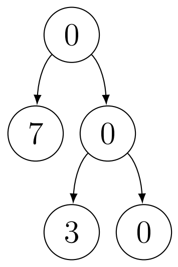
```
public class ScoreHistory implements Comparable<ScoreHistory> {
  int[] scoreCounts;
  String name;
  public ScoreHistory(String name) {
    scoreCounts = new int[11];
    this.name = name;
  }

  public void addScore(int score) {
    if(score > 5 || score < -5)
      throw new IllegalArgumentException("score out of range");
    int arrayPos = score + 5;
    scoreCounts[arrayPos] = scoreCounts[arrayPos] + 1;
  }

  public int numberOfPlays() {
    int total = 0;
    for(int count : scoreCounts)
      total += count;
    return total;
  }

  private int bestScore() {
    for(int i = 5; i >= -5; i--)
      if(scoreCounts[i + 5] > 0)
        return i;
    return -6;
  }

  public int compareTo(ScoreHistory other) {
    int best = bestScore();
    int scoreDiff = best - other.bestScore();
    if(scoreDiff != 0) return scoreDiff;
    return scoreCounts[best + 5] - other.scoreCounts[best + 5];
  }

}
```

3. (10 points) **Binary Trees**

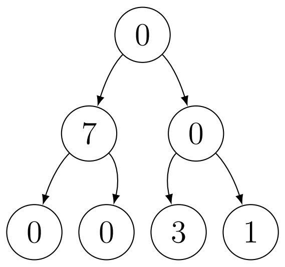
Write a method hasZeroPath for the IntTree class we have discussed in lecture. This method should return true if there is a path from the root of the tree to a leaf consisting only of elements with zero value. An empty tree is considered to have a 0-length path and should therefore return true.

You may write helper methods to help you solve this problem, but you should not call any other methods of IntTree. Do not create any additional structured objects (Stacks, arrays, etc.) to solve this problem.

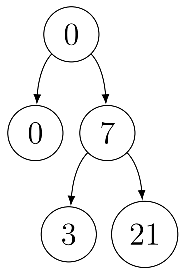
Examples



hasZeroPath() returns true



hasZeroPath() returns false



hasZeroPath() returns true

Solution:

```
public boolean hasZeroPath() {
  return hasZeroPath(overallRoot);
}
private boolean hasZeroPath(IntTreeNode node) {
  if(node == null) return true;
  if(node.data != 0) return false;
  return hasZeroPath(node.left) || hasZeroPath(node.right);
}
```

4. (10 points) **Inheritance/Casting Question**
   Assume the follow classes have been defined:

```
public class Bird
{
    public void method1() {
        System.out.println("Bird 1");
        method2();
    }

    public void method2() {
        System.out.println("Bird 2");
    }
}

public class Dog extends Bird
{
    public void method2() {
       System.out.println("Dog 2");
    }

    public void method3() {
       System.out.println("Dog 3");
    }
}

public class Cat extends Bird
{
    public void method1() {
       System.out.println("Cat 1");
    }

    public void method2() {
       super.method2();
       System.out.println("Cat 2");
    }

    public void method3() {
       System.out.println("Cat 3");
    }
}
```

   You may find it handy to carefully tear this page out of your test packet so you can refer to it as you answer the questions on the following page.

Assume the following variables have been defined:

```
Bird birdDog = new Dog();
Dog dogDog = new Dog();
Cat catCat = new Cat();
```

On the lines below, indicate the output produced by each statement. If the statement produces more than one line of output, indicate the line breaks with slashes as in "a/b/c" to indicate three lines of output with "a" followed by "b" followed by "c". If the statement causes an error, write either the phrase "compiler error" or "runtime error" to indicate when the error would be detected (you may abbreviate these as "ce" and "re" or "c.e." and "r.e.").

`birdDog.method3();` $\boxed{\texttt{c.e.}}$

`dogDog.method1();` $\boxed{\texttt{Bird 1/Dog 2}}$

`catCat.method2();` $\boxed{\texttt{Bird 2/Cat 2}}$

`((Bird) catCat).method1();` $\boxed{\texttt{Cat 1}}$

`((Bird) dogDog).method1();` $\boxed{\texttt{Bird 1/Dog 2}}$

`((Bird) catCat).method2();` $\boxed{\texttt{Bird 2/Cat 2}}$

`((Cat) birdDog).method2();` $\boxed{\texttt{r.e.}}$

`((Cat) dogDog).method2();` $\boxed{\texttt{c.e.}}$

`((Dog) birdDog).method3();` $\boxed{\texttt{Dog 3}}$

`((Cat) ((Bird) catCat)).method1();` $\boxed{\texttt{Cat 1}}$