Problem 1: [20 points]

a) The linked structure above uses the node and die classes from our most recent lab. Write a series of assignment statements below that will remove the eight-sided die from the list by linking around it. You may declare additional variables of type DieNode if you wish, but assignments to them must be made through head.

b) On the diagram above, draw in the changes that would result from executing the following assignment statements. (Start with the original structure, rather than the modified list produced by your changes in part a.)

```
DieNode temp = head.next;
temp.next.next.die = head.next.next.die;
temp.next = head.next;
```
These class definitions are used by the questions on the next two pages. Feel free to tear this page out of the exam so you can be looking at it as you read the questions. (You don’t need to turn this page in if you’ve torn it out. Take it home and put it on the fridge.)

```java
public class Base {
    protected int num = 5;
    
    public void yell() {
        for(int i=0; i<num; i++) {
            System.out.println("HEY!");
        }
    }
}

public class Thingy extends Base {
    public Thingy(int num) {
        this.num = num;
    }
    
    public void yell() {
        System.out.println("I don't like to yell.");
    }
    
    public void snort() {
        System.out.println("Snort!");
    }
}

public class Other extends Thingy {
    public Other() {
        super(10);
    }
    
    public void yell() {
        System.out.print("My parent says: ");
        super.yell();
    }
    
    public void mumble() {
        System.out.println("mumble...");
    }
}
```
Problem 2: [20 points]

a) Can an instance of the class Base be constructed via a no-argument constructor? (In other words, is “\texttt{new Base()}” a legal expression?) Explain.

b) Can an instance of the class Thingy be constructed via a no-argument constructor? Explain.

c) Can an instance of the class Thingy be constructed via a one-argument constructor? Explain.

d) Can an instance of the class Other be constructed via a no-argument constructor? Explain.

e) Can an instance of the class Other be constructed via a one-argument constructor? Explain.
Problem 3: [20 points]

a) Is the following code legal? If so, what output would it produce? If not, explain why not.

```java
Base x = new Thingy(2);
x.yell();
```

b) Is the following code legal? If so, what output would it produce? If not, explain why not.

```java
Thingy x = new Base();
x.yell();
```

c) Is the following code legal? If so, what output would it produce? If not, explain why not.

```java
Base x = new Other();
x.mumble();
```
Problem 4: [20 points]

a) You've been hired as a consultant by Bradco. They’re writing a program to store temperature readings for long-term archival purposes. Once per second a new temperature reading is added to the front of a list until a year’s worth of data is recorded. It’s not expected that the data values will be inspected (read) frequently, if at all. Would you suggest using an ArrayList or a LinkedList? Justify your answer.

b) Your boss asked you to double-check another software developer's work. They determined that a particular method with \( T(n) = 2n + 4n^2 + 2 \) was \( O(n^2) \). To support their claim, they noted that selecting \( c=6 \) and \( n_0=1 \) demonstrates their Big-O estimate is valid. Are they right? Explain why or why not.

c) Determine \( T(n) \) for the code below. For full credit, please circle each of the computational steps in the code that you’re counting, as well as writing out \( T(n) \).

```java
public int sumUntil(int[] nums, int limit) {
    int sum = 0;
    int i = 0;
    while(i < nums.length && sum < limit) {
        sum = sum + nums[i];
        i = i+1;
    }
    System.out.println("sum got to "+sum);
    return i;
}
```
Problem 5: [20 points]

Below, write a subclass of HistoryDie called RegularDie that always has six sides. (Users, therefore, don't need to specify the number of sides when constructing one.) When rolled, it should update the history array appropriately. Its toString() method should return the same string as its parent's, but with "RegularDie: " added to the front. Documentation for HistoryDie is on the next page. (Feel free to tear out that page if you like.)
public class HistoryDie extends BasicDie

HistoryDie is a subclass of the BasicDie. It works exactly the same way, but stores information about how many times each possible value has been rolled.

Author: Brad Richards

Field Summary

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Field and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>protected int[]</td>
<td>history</td>
</tr>
</tbody>
</table>

Methods inherited from class BasicDie

Constructor Summary

public HistoryDie(int numSides)

Our one-argument constructor explicitly calls the one-argument constructor in the parent class.

Parameters:
- numSides - The desired number of sides on the die.

Method Summary

public int roll()

We invoke the BasicDie's roll method to get a random roll value, update our history array, then return the result.

Overrides:
- roll in class BasicDie

Returns:
- A random value between 1 and numSides.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait