Problem 1: [22 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 four-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.)

```java
DieNode temp = head.next.next;
temp.next.die = head.die;
temp.next = head;
```
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.

```java
public class Base {
    protected String name = "Marvin";

    public void report() {
        System.out.println("Hi, I'm "+name);
    }
}

public class Thingy extends Base {
    public Thingy(String who) {
        name = who;
    }

    public void report() {
        System.out.println("Reporting:");
        super.report();
    }

    public void rename(String who) {
        name = who;
    }
}

public class Other extends Thingy {
    public Other() {
        super("Eleanor");
    }

    public void report() {
        System.out.println("I think my name is "+name);
    }

    public void nameCheck(String who) {
        if (who.equals(name)) {
            System.out.println("Yep, I'm "+who);
        } else {
            System.out.println("Nope, that's not me");
        }
    }
}
```
Problem 2: [24 points]

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

```java
Base x = new Thingy("Brad");
x.report();
```

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

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

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

```java
Base x = new Other();
x.report();
```

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

```java
Base x = new Other();
x.nameCheck("Brad");
```
The questions below refer to this method, which makes use of the code on page 2:

```java
public void renameSome(ArrayList<Thingy> things) {
    Thingy item;
    for(int i=0; i<things.size(); i++) {
        if (i % 2 == 0) { // That’s the mod operator
            item = things.get(i);
            item.rename("Brad");
        }
    }
    System.out.println("Done");
}
```

**Problem 3: [26 points]**

a) Is the `renameSome` method polymorphic? Explain.

b) Can the input list contain instances of `Base`? Explain why or why not.

c) Write a $T(n)$ function that represents the worst-case number of computational steps required to execute `renameSome` on a list of length $n$. Explain your answer.

d) What’s $O(n)$ for `renameSome` when passed a list of length $n$? Justify your answer by providing appropriate values for $c$ and $n_0$ given your $T(n)$ from above.
Problem 4: [28 points]

Below, finish the definition of a Tic-Tac-Toe playing class called CenterPlayer that could be pitted against the RandomPlayer, CornerPlayer, or NextOpenPlayer classes from the second assignment. Its makeMove method should attempt to take the center square if it’s open. If the center square is taken it should take a corner, otherwise select an open spot randomly. (Feel free to add additional methods or instance variables as you see fit.) For full credit, use inheritance to simplify the makeMove method as much as possible. Documentation for the Board class can be found on the next page in case that’s helpful.

```java
public class CenterPlayer
{
    public CenterPlayer(int symbol, String name) {

    }

    public void makeMove(Board theBoard) {

    }
}
```
public class Board extends java.lang.Object

The Board class models a 3 by 3 Tic-Tac-Toe board. There are methods for filling board positions with X or O symbols, checking for a winner, and inspecting the contents of the board.

Field Summary

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Field and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static int</td>
<td>BLANK</td>
</tr>
<tr>
<td></td>
<td>A constant corresponding to blank spaces.</td>
</tr>
<tr>
<td>static int</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>A constant corresponding to player/symbol O.</td>
</tr>
<tr>
<td>static int</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>A constant corresponding to player/symbol X.</td>
</tr>
</tbody>
</table>

Constructor Summary

<table>
<thead>
<tr>
<th>Constructor and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Board()</td>
</tr>
<tr>
<td>The no-argument constructor creates a new blank board, using the characters 'X', 'O', and '.' to represent X, O, and open spaces.</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>boardFilled()</td>
</tr>
<tr>
<td></td>
<td>This method returns true if all spots on the board are filled.</td>
</tr>
<tr>
<td>void</td>
<td>fillPosition(int col, int row, int player)</td>
</tr>
<tr>
<td></td>
<td>Fills a position on the board with the specified symbol (Board.X or Board.O).</td>
</tr>
<tr>
<td>int</td>
<td>getContents(int col, int row)</td>
</tr>
<tr>
<td></td>
<td>This method returns the contents of the specified board position using one of the three constants defined by the Board class (Board.X, Board.O, or Board.BLANK).</td>
</tr>
<tr>
<td>int</td>
<td>getWinner()</td>
</tr>
<tr>
<td></td>
<td>Checks to see if a player has won the game.</td>
</tr>
<tr>
<td>boolean</td>
<td>isOpen(int col, int row)</td>
</tr>
<tr>
<td></td>
<td>Inspects the board to see if the specified position is open.</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>toString()</td>
</tr>
</tbody>
</table>