

Warm up recursions:

```
/*
 * Exercise number   : 8.1
 * File Name        : PalindromeDetection.java
 * Name (First Last) : Idan Felix
 * Group number     : 1
 */
public class PalindromeDetection {
    public static boolean testPalindrome(String suspectedPalindrome){
        int length = suspectedPalindrome.length();
        if (length <= 1){
            return true;
        }

        char firstChar = suspectedPalindrome.charAt(0);
        int lastCharIndex = length - 1;
        char lastChar = suspectedPalindrome.charAt(lastCharIndex);
        if (!Character.isLetter(firstChar)){
            String firstCharRemoved = suspectedPalindrome.substring(1);
            return testPalindrome(firstCharRemoved);
        }

        if (!Character.isLetter(lastChar)){
            String lastCharRemoved =
                suspectedPalindrome.substring(0, lastCharIndex);
            return testPalindrome(lastCharRemoved);
        }

        String innerString =
            suspectedPalindrome.substring(1, lastCharIndex);
        if (firstChar - lastChar == 0){
            return testPalindrome(innerString);
        }

        if (firstChar - lastChar == -32){
            // First letter is capital, and the last is not,
            // but the letters are equal.
            return testPalindrome(innerString);
        }

        if (firstChar - lastChar == 32){
            // Other way around.
            return testPalindrome(innerString);
        }

        return false;
    }
}
```

Mathematical proofs:

Formula:

I used "floor(x)" to denote $\lfloor x \rfloor$

- $a*b = (2a)(b/2)$ for even b, $(2a)(\text{floor}(b/2))+a$ for odd b.

Induction:

Note: you can base it on the commutativity property of multiplication, but the method is not commutative.

BASE:

- $B=0$ (b is even) then $\text{mult}(a,b)=(2a)(b/2)=(2a*0/2)=0=a*0$.
- $B=1$ (b is odd) then $\text{mult}(a,b)=(2a*\text{floor}(1/2))+a=(2a*0)+a=a=a*1$.
- $A=0$ then $\text{mult}(a,b)=2a*(b/2)=2*0*(b/2)=0*b$ or $2a*\text{floor}(b/2)+a=0+0=0=0*b$.

ASSUMPTION:

- $\text{Mult}(a,b)=a*b$ for any a, $b \geq 0$

STEP:

- Even b:
 - $\text{Mult}(a,b+1)$
 - $=\text{Mult}(a,C)$ // let's assign $C=b+1$ odd
 - $=a*C$ // from the induction assumption
 - $=a(b+1)$
- Odd b:
 - $\text{Mult}(a,b+1)$
 - $=\text{mult}(a,C)$ // let's assign $C=b+1$ even
 - $=a*C$ // induction assumption
 - $=a(b+1)$
- Even b:
 - $\text{Mult}(a+1,b)$
 - $=2(a+1)*b/2$ // replacing with formula
 - $=(2a+2)(b/2)$ // simple math
 - $=2a*b/2+2*b/2$ // simple math
 - $=\text{mult}(a,b)+b$ // replacing back from formula
 - $=a*b+b$ // induction assumption
 - $=b(a+1)$.
- Odd b:
 - $\text{Mult}(a+1,b)$

- `=2(a+1)*floor(b/2)+(a+1)` // replacing with formula
- `=(2a*floor(b/2))+2*floor(b/2)+a+1` // simple math
- `=mult(a,b)+2*floor(b/2)+1` // replacing back from formula
- `=mult(a,b)+(b-1)+1` // because b is odd
- `=a*b+b`
- `=b(a+1)`

8.3 call stack

1. Recursively concatenating the first character to the end of the reversed string (like in the method given), or recursively concatenating the reversed string to the last character.
2. Iterative. The following is using the second approach (=bad solution). **The second solution is the right one:**

```
public static String reverseIterativeWrong(String s) {
    StringBuilder reversed = new StringBuilder();
    int currChar = s.length();
    while (--currChar >= 0) {
        reversed.append(s.charAt(currChar));
    }
    return reversed.toString();
}

public static String reverseIterative(String s) {
    StringBuilder reversed = new StringBuilder();
    int currChar = 0;
    while (currChar < s.length()) {
        reversed.insert(0, s.charAt(currChar++));
    }
    return reversed.toString();
}
```

3. The length, plus 1 for the base case – Total of 44 for the given string.
4. 1 – Just the method call. No additional frames were added.
5. The recursive method caused a Stack overflow (“StackOverflowError”). The iterative version worked just fine.

8.4 Tail recursions

1. In regular recursions the stack is stacked up, and when it shrinks (on return calls), the return value is computed. On tail recursions the final return value is already computed when the stack is at its max. In order for this to happen, the base case, which is called when the call-stack is at its max, has to return the final result, as opposed to regular recursions, where it returns a base result on which more computations are executed.

2.

```
static int factorial(int number) {
    if (number == 0 || number == 1) {
        return 1;
    }
    return factorialTail(number, 1);
}

static int factorialTail(int num, int mult) {
    if(num == 1) {
        return mult;
    } else {
        return factorialTail(num - 1, mult * num);
    }
}
```

3. The compiler can easily turn a tail recursion to a loop. This improves runtime performance, and enables the handling of more items (as the Stack is quite limited).

8.5 Fractals

```
/*
Exercise number   : 8.5
File Name        : SnowflakeFractals.java
Name (First Last) : Idan Felix
Group number     : 1
*/

public class SnowflakeFractals {
    public static final int LENGTH = 243; // Use 729 for levels 4 and 5
    public static final int OUTER_ANGLE = 120;
    public static final int INNER_ANGLE = 60;

    public static void main(String[] args) {
        if (args.length != 1){
            printUsage();
            return;
        }
        int level = stringToInteger(args[0]);
        if (level <= 0){
            printUsage();
            return;
        }

        Turtle helgeVonKoch = new Turtle();
        drawSnowflake(level, helgeVonKoch);
        helgeVonKoch.hide();
    }

    private static void drawSnowflake(int level, Turtle helgeVonKoch) {
        // Draws the basic shape - a Equilateral triangle
        helgeVonKoch.tailDown();
        drawFractal(LENGTH, level, helgeVonKoch);
        helgeVonKoch.turnRight(OUTER_ANGLE);
    }
}
```

```

        drawFractal(LENGTH, level, helgeVonKoch);
        helgeVonKoch.turnRight(OUTER_ANGLE);
        drawFractal(LENGTH, level, helgeVonKoch);
    }

    private static int stringToInteger(String string) {
        try {
            return Integer.valueOf(string);
        } catch (NumberFormatException e) {
            return -1;
        }
    }

    private static void printUsage() {
        System.out.println("This program draws a Koch Snowflake.");
        System.out.println("Usage: java SnowflakeFractals <level>");
        System.out.println("        where <level> is an integer " +
            "greater than 0.");
    }

    public static void drawFractal(int length, int level,
        Turtle helgeVonKoch) {
        if (level==0) {
            // The Basic step
            helgeVonKoch.moveForward(length);
        }
        else
        {
            // The recursive step
            drawFractal(length/3, level-1, helgeVonKoch);
            helgeVonKoch.turnLeft(INNER_ANGLE);
            drawFractal(length/3, level-1, helgeVonKoch);
            helgeVonKoch.turnRight(OUTER_ANGLE);
            drawFractal(length/3, level-1, helgeVonKoch);
            helgeVonKoch.turnLeft(INNER_ANGLE);
            drawFractal(length/3, level-1, helgeVonKoch);
        }
    }
}

```

8.6 StudentsList Element and StudentsList

```

/*
 * Exercise number   : 8.7
 * File Name        : StudentsLinkedList.java
 * Name (First Last) : Idan Felix
 * Group number     : 1
 */

public class StudentsLinkedList {
    private static final String DELIMITER = ":";
    private static final String NEWLINE =
        System.getProperty("line.separator");

```

```

private static final String THE_LIST_IS_EMPTY = "The list is empty";
private StudentElement head = null;

public void addStudent(Student student){
    if (student == null)
        return;
    }
    if (getStudent(student.getName()) != null){
        return;
    }

    StudentElement newStudent = new StudentElement(student, null);
    if (head == null){
        head = newStudent;
    }
    else
    {
        StudentElement tail = head;
        while (tail.getNext() != null){
            tail = tail.getNext();
        }
        tail.setNext(newStudent);
    }
}

public Student getStudent(String name){
    StudentElement tempElement = head;
    while (tempElement != null){
        if (tempElement.getStudent().getName().equals(name)){
            return tempElement.getStudent();
        }

        tempElement = tempElement.getNext();
    }

    return null;
}

public void setGrade(Student student){
    if (student == null){
        return;
    }

    Student theStudent = getStudent(student.getName());
    if (theStudent != null){
        theStudent.setGrade(student.getGrade());
    }
}

public String toString(){
    if (head == null){
        return THE_LIST_IS_EMPTY;
    }
    else
    {
        StudentElement tempHead = head;
        StringBuilder list = new StringBuilder();

```

```

        while (tempHead != null) {
            Student student = tempHead.getStudent();
            list.append(student.getName());
            list.append(DELIMITER);
            list.append(student.getGrade());
            list.append(NEWLINE);

            tempHead = tempHead.getNext();
        }
        return list.toString();
    }
}

public float getAverage() {
    StudentElement tempHead = head;
    float sumOfGrades = 0;
    int countOfGrades = 0;
    while (tempHead != null) {
        sumOfGrades += tempHead.getStudent().getGrade();
        countOfGrades++;
        tempHead = tempHead.getNext();
    }

    return (sumOfGrades / countOfGrades);
}

public void removeStudent(String name) {
    if (head == null || getStudent(name) == null) {
        return;
    }

    if (head.getStudent().getName().equals(name)) {
        head = head.getNext();
        return;
    }

    // We verify that the next student is the one we need to delete.
    StudentElement current = head;
    while (current != null) {
        StudentElement next = current.getNext();
        if (next == null) {
            return; // Student wasn't found.
        }
        if (next.getStudent().getName().equals(name)) {
            {
                current.setNext(next.getNext());
            }
        }

        current = next;
    }
}
}

```

```

/*
 * Exercise number   : 8.6
 * File Name        : StudentElement.java
 * Name (First Last) : Idan Felix
 * Group number     : 1
 */

public class StudentElement {
    private static final String PARAMETER_NULL_MESSAGE =
        "parameter 'student' must not be null";
    private Student student;
    private StudentElement next;

    public Student getStudent(){
        return this.student;
    }

    public StudentElement getNext(){
        return next;
    }

    public void setNext(StudentElement next){
        this.next = next;
    }

    public void setStudent(Student student){
        if (student == null) {
            throw new IllegalArgumentException(PARAMETER_NULL_MESSAGE);
        }this.student = student;
    }

    /**
     * Constructs a new element
     */
    public StudentElement(Student student, StudentElement next){
        setStudent(student);
        setNext(next);
    }
}

```