Solutions – Exam Introduction to Programming (Java) (X_400634) – 16/12/2019

1a)
```
1 3
5 2
4 3
9 3
11 3
13 3
13 3
```

1b)
```
double calculate(double x, int n) {
    int sign = 1;
    int factorial = 1;
    int add = 0;
    double power = x * x;
    double result = sign * (factorial + add) / power;

    for (int i = 1; i < n; i++) {
        sign *= -1;  // sign = -sign
        factorial *= (2 * i - 1) * (2 * i);
        add += 1;
        power *= x;

        result += sign * (factorial + add) / power;
    }

    return result;
}
```

1c)
```
static final int NUMBER_OF_ROWS = 5;
static final int NUMBER_OF_COLUMNS = 9;

char[][] matrix = new char[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];


int numberOfOddOdds(int[][] m) {
    int result = 0;

    for (int i = 0; i < m.length; i += 2) {
        for (int j = 0; j < m[0].length; j += 2) {
            if (m[i][j] % 2 == 1) {
                result += 1;
            }
        }
    }

    return result;
}
```

1d)
```
3 4
6 5
3 11
10 6
5 16
30 16
```

3a)

```
String convert(char[] a, int i, int j) {
    if (j < i) {
        return "";
    }

    return a[i] + convert(a, i + 1, j);
}
```

3b)

```
boolean occurrences(String s, String sub, int n) {
    if (n == 0) {
        return true;
    }

    if (s.length() < sub.length()) {
        return false;
    }

    String begin = s.substring(0, sub.length());
    String rest = s.substring(1);

    if (begin.equals(sub)) {
        n -= 1;
    }

    return occurrences(rest, sub, n);
}
```

2a)

```
class Necklace {
    static final int MAX_NUMBER_OF_GEMSTONES = 75;

    Gemstone[] gemstones;
    int numberOfGemstones;

    Necklace() {
        gemstones = new Gemstone[MAX_NUMBER_OF_GEMSTONES];
        numberOfGemstones = 0;
    }

    void add(Gemstone gemstone) {
        gemstones[numberOfGemstones] = gemstone;
        numberOfGemstones += 1;
    }
}
```

2b)

add to the class Gemstone:

```
static final int VALUEABLE_BORDER = 100;

boolean isValueable() {
    return classification.equals("precious") &&
           weight >= VALUEABLE_BORDER;
}
```

add to the class Necklace:

```
Necklace valueableGemstones() {
    Necklace result = new Necklace();

    for (int i = 0; i < numberOfGemstones; i++) {
        if (gemstones[i].isValuable()) {
            result.add(gemstones[i]);
        }
    }

    return result;
}
```

2c) add to the class Gemstone:

```
static final int SOFT_BORDER = 4;

boolean isSoft() {
    return hardness < SOFT_BORDER;
}
```

add to the class Necklace:

```
void removeSoftGemstones() {
    for (int i = 0; i < numberOfGemstones; i++) {
        if (gemstones[i].isSoft()) {
            gemstones[i] = gemstones[numberOfGemstones – 1];
            numberOfGemstones -= 1;
            i -= 1;
        }
    }
}
```

2d)

```
int valuableGemstones(String name) {
    Necklace help = valuableGemstones();
    help.removeSoftGemstones();
    return help.gemstones(name);
}
```