Question 1

[25 marks]

1 A palindromic number reads the same both ways. The largest palindrome made from the product of two 2-digit numbers is 9009 = 91 × 99.

Write a Java program which prints out the largest palindrome made from the product of two 3-digit numbers.

```
public class Q1 {
    public static void main (String args[]) {
        int largestPalindrome = 0;
        int n1 = 0;
        int n2 = 0;
        for (int i = 100; i < 1000; i++) {
            for (int j = 100; j < 1000; j++) {
                if (isPalindrome(i*j)) {
                    if(largestPalindrome < i * j) {
                        largestPalindrome = i * j;
                        n1 = i;
                        n2 = j;
                    }
                }
            }
        }
        System.out.println("Largest Palindrome of 3-digit numbers is "
                + n1 + " * " + n2 + " = "  + largestPalindrome);
    }

    public static boolean isPalindrome(int num) {
        String numStr = String.valueOf(num);
        StringBuffer sb = new StringBuffer(numStr);
        String reverseNumStr = sb.reverse().toString();
        if (numStr.equals(reverseNumStr)) {
            return true;
        } else {
            return false;
        }
    }
}
```

Question 2

[25 marks]

2 Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

Write a Java program which prints out the 1,000th term of the above Fibonacci sequence.

```java
public class Q2 {
    public static void main (String args[]) {
        //The acutual Fibonacci sequence is 1, 1, 2, 3, 5, 8...
        //Is this case, the exam paper said is 1, 2, 3 ,5 ,8
        //It missed a '1', so if you want to get 1000th of sequence,
        //You need to input 1001.
        System.out.println(fibonacci(1001));
    }

    public static int fibonacci(int n) {
        if (n <= 1) {
            return n;
        }
        else {
            return fibonacci(n-1) + fibonacci(n-2);
        }
    }

}
```

Question 3

[25 marks]

3 Given *n* students in a class, how likely is it that some of them share the same birthday?

Write a Java program which uses a Monte Carlo simulation to determine to the nearest percent the probability that at least *x* number of students in a class of size *n* have the same birthday. The program should take in *x* and *n* as inputs.

```
import java.util.Scanner;

public class Q3 {
    public static void main (String args[]) {
        Scanner sc = new Scanner(System.in);
        int x = sc.nextInt();
        int n = sc.nextInt();
        sc.close();
        System.out.println(getProbability(x,n));
    }

    public static double getProbability(int x, int n) {
        int N = 1000000;
        int classSize = n;
        int sameBirthdayNum = x;
        int count = 0;

        for(int i=0; i< N; i++) {
            int[] birthdays = new int[365];

            for (int j=0; j < classSize ; j++) {
                int randomBirthday = (int) (365 * Math.random());
                if(birthdays[randomBirthday] >= sameBirthdayNum - 1) {
                    count ++;
                    break;
                }
                birthdays[randomBirthday]++;
            }
        }
        return (double) 100 * count / (double) N;
    }
}
```

Question 4

[25 marks]

4 Write a Java program which takes in an array of Strings and prints them out ordered according to their size, with the shortest Strings being printed first. If two Strings have the same size, then they should be printed out in alphabetical order.

```
import java.util.Scanner;
import java.util.LinkedList;
import java.util.Collections;
import java.util.Comparator;

public class Q4 {
    public static void main (String args[]) {
        Scanner sc = new Scanner (System.in);
        LinkedList <String> input = new LinkedList <String>();
        int SIZE = sc.nextInt();
        sc.nextLine();

        for(int i = 0; i < SIZE; i++) {
            input.add(sc.nextLine());
        }

        Collections.sort(input, new Comparator<String>() {
            @Override
            public int compare(String o1, String o2) {
                if(o1.length() != o2.length()) {
                    return o1.length() - o2.length();
                }
                else {
                    return o1.compareTo(o2);
                }
            }
        });

        for(String element : input ) {
            System.out.println(element);
        }
    }

}
```