

# Computer Project

(2017-2019)

Satvik Saha

Class: XI B

Roll number: 24

*“Writing code a computer can understand is science. Writing code  
other programmers can understand is an art.”*

— **Jason Gorman**

“When in doubt, use brute force.”

— Ken Thompson

**Problem 1** An  $n$  digit integer  $(a_1a_2 \dots a_n)$ , where each digit  $a_i \in \{0, 1, \dots, 9\}$ , is said to have *unique digits* if no digits are repeated, i.e., there is no  $i, j$  such that  $a_i = a_j$  ( $i \neq j$ ).

Verify whether an inputted number has *unique digits*.

## Solution

## Source Code

```
1 public class Unique {
2     public static void main (String[] args) {
3         try {
4             long number = Long.parseLong(args[0]);
5             if (isUnique(number)) {
6                 System.out.println("Unique Number!");
7             } else {
8                 System.out.println("Not a Unique Number!");
9             }
10        } catch (NumberFormatException e) {
11            System.out.println("Enter an integer as the first argument!");
12        } catch (IndexOutOfBoundsException e) {
13            System.out.println("Enter 1 argument (integer)!");
14        }
15    }
16
17    public static boolean isUnique (long number) {
18        int[] count = new int[10];
19        for (long n = Math.abs(number); n > 0; n /= 10) {
20            int digit = (int) n % 10;
21            count[digit]++;
22            if (count[digit] > 1){
23                return false;
24            }
25        }
26        return true;
27    }
28 }
```

*“Elegance is not a dispensable luxury but a factor that decides between success and failure.”*

— Edsger W. Dijkstra

**Problem 2** A *partition* of a positive integer  $n$  is defined as a collection of other positive integers such that their sum is equal to  $n$ . Thus, if  $(a_1, a_2, \dots, a_k)$  is a partition of  $n$ ,

$$n = a_1 + a_2 + \dots + a_k \quad (a_i \in \mathbb{Z}^+)$$

Display every *unique partition* of an inputted number.

## Solution

## Source Code

```
1
2 public class Partition {
3     public static void main (String[] args) {
4         try {
5             int target = Integer.parseInt(args[0]);
6             if (target < 1) {
7                 throw new NumberFormatException();
8             }
9             partition(target);
10        } catch (NumberFormatException e) {
11            System.out.println("Enter a natural number as the first
12                argument!");
13        } catch (IndexOutOfBoundsException e) {
14            System.out.println("Enter 1 argument (natural number)!");
15        }
16    }
17    public static void partition (int target) {
18        partition(target, target, "");
19    }
20    public static void partition (int target, int previousTerm, String suffix) {
21        if (target == 0)
22            System.out.println(suffix);
23        for (int i = 1; i <= target && i <= previousTerm; i++)
24            partition(target - i, i, suffix + " " + i);
25    }
26 }
```

*“Simplicity is the ultimate sophistication.”*

— Leonardo da Vinci

**Problem 3** A *Caesar cipher* is a type of monoalphabetic substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. The positions are circular, i.e., after reaching *Z*, the position wraps around to *A*. For example, following is some encrypted text, using a right shift of 5.

Plain: ABCDEFGHIJKLMNOPQRSTUVWXYZ  
Cipher: FGHIJKLMNOPQRSTUVWXYZABCDE

Thus, after mapping the alphabet according to the scheme  $A \mapsto 0, B \mapsto 1, \dots, Z \mapsto 23$ , we can define an encryption function  $E_n$ , in which a letter  $x$  is shifted rightwards by  $n$  as follows.

$$E_n(x) = (x + n) \mod 26$$

The corresponding decryption function  $D_n$  is simply

$$D_n(x) = (x - n) \mod 26$$

Implement a simple version of a *Caesar cipher*, encrypting capitalized plaintext by shifting it by a given value. Interpret positive shifts as rightwards, negative as leftwards.

## Solution

### Source Code

```
1 public class CaesarShift {
2     public static void main (String[] args) {
3         try {
4             int shift = Integer.parseInt(args[0]) % 26;
5             String plaintext = args[1].toUpperCase();
6             String ciphertext = "";
7             for (int i = 0; i < plaintext.length(); i++) {
8                 char plain = plaintext.charAt(i);
9                 char crypt = ' ';
10                if ('A' <= plain && plain <= 'Z') {
11                    crypt = numToChar(charToNum(plain) + shift);
12                } else {
13                    crypt = plain;
14                }
15                ciphertext += crypt;
16            }
17        }
18    }
19 }
```

```

16         }
17         System.out.println(ciphertext);
18     } catch (NumberFormatException e) {
19         System.out.println("Enter an integer as the first argument!");
20     } catch (IndexOutOfBoundsException e) {
21         System.out.println("Enter 2 arguments (shift, plaintext)!");
22     }
23 }
24
25 public static int charToNum (char letter) {
26     return Character.toUpperCase(letter) - 'A';
27 }
28
29 public static char numToChar (int number) {
30     return (char) ('A' + Math.floorMod(number, 26));
31 }
32 }

```

*“There are 2 hard problems in computer science: cache invalidation,  
naming things, and off-by-1 errors.”*

— Leon Bambrick

**Problem 4** A *palindrome* is a sequence of characters which reads the same backwards as well as forwards. For example, *madam*, *racecar* and *kayak* are words which are palindromes. Similarly, the sentence “A man, a plan, a canal -- Panama!” is also a palindrome.

Analyze a sentence of input and display all *words* which are palindromes. If the entire *sentence* is also a palindrome, display it as well.

*(A word is an unbroken sequence of characters, separated from other words by whitespace. Ignore single letter words such as I and a. Ignore punctuation, numeric digits, whitespace and case while analyzing the entire sentence.)*

## Solution

### Source Code

```
1 import java.util.Scanner;
2
3 public class Palindrome {
4     public static void main (String[] args) {
5         System.out.print("Enter your sentence : ");
6         String sentence = (new Scanner(System.in)).nextLine().trim();
7         boolean foundPalindrome = false;
8         System.out.println("Palindromes : ");
9         foundPalindrome |= checkWords(sentence);
10        foundPalindrome |= checkSentence(sentence);
11        if (!foundPalindrome) {
12            System.out.println("No palindromes found!");
13        }
14    }
15
16    public static boolean checkWords (String sentence) {
17        boolean foundPalindrome = true;
18        int start = -1;
19        int end = 0;
20        while (end < sentence.length()) {
21            while (Character.isWhitespace(sentence.charAt(++start)));
22            end = start;
```

```

23         while (end < sentence.length() &&
24                !Character.isWhitespace(sentence.charAt(end++)));
25         String word = sentence.substring(start, end).trim();
26         if (isPalindrome(word)) {
27             foundPalindrome = true;
28             System.out.println(getAlphabets(word));
29         }
30         start = end - 1;
31     }
32     return foundPalindrome;
33 }
34
35 public static boolean checkSentence (String sentence) {
36     if (isPalindrome(sentence)) {
37         System.out.println(sentence);
38         return true;
39     }
40     return false;
41 }
42
43 public static boolean isPalindrome (String text) {
44     String rawText = getAlphabets(text).toUpperCase();
45     for (int i = 0, j = rawText.length() - 1; i < j; i++, j--) {
46         if (rawText.charAt(i) != rawText.charAt(j)) {
47             return false;
48         }
49     }
50     return (rawText.length() > 1);
51 }
52
53 public static String getAlphabets (String text) {
54     String rawText = "";
55     for (int i = 0; i < text.length(); i++) {
56         if (Character.isAlphabetic(text.charAt(i))) {
57             rawText += text.charAt(i);
58         }
59     }
60     return rawText;
61 }

```



*“Any fool can use a computer. Many do.”*

— Ted Nelson

**Problem 5** Design a simple interface for an examiner which can format and display marks scored by a group of students in a particular examination. Calculate the percentage scored by each candidate and display the list of students and percentages in an ASCII bar chart, arranged alphabetically.

## Solution

### Source Code

```
1 public class Marksheet {
2     public static final int SCREEN_WIDTH = 100;
3     private final double maxMarks;
4     private final int numberOfStudents;
5     private int lastStudent;
6     private String[] names;
7     private double[] marks;
8
9     public Marksheet (double maxMarks, int numberOfStudents) {
10         this.maxMarks = maxMarks;
11         this.numberOfStudents = numberOfStudents;
12         this.names = new String[numberOfStudents];
13         this.marks = new double[numberOfStudents];
14         this.lastStudent = -1;
15     }
16
17     public boolean addMarks (String name, double score) {
18         try {
19             names[++lastStudent] = name;
20             marks[lastStudent] = score;
21             return true;
22         } catch (IndexOutOfBoundsException e) {
23             return false;
24         }
25     }
26
27     public void displayChart () {
28         System.out.println(Marksheet.multiplyString("-",
29             Marksheet.SCREEN_WIDTH));
30         for (int i = 0; i <= lastStudent; i++) {
31             double fraction = marks[i] / maxMarks;
```

```

31         String name = (names[i].length() < 16)
32             ? names[i]
33             : (names[i].substring(0,13) + "...");
34         int points = (int) (fraction * (SCREEN_WIDTH - 34));
35         String bar = multiplyString("*", points)
36             + multiplyString(" ", SCREEN_WIDTH - 34 - points);
37         System.out.printf("| %16s | %s | %6.2f %% |\n"
38             , name
39             , bar
40             , fraction * 100);
41     }
42     System.out.println(Marksheet.multiplyString("-",
43         Marksheet.SCREEN_WIDTH));
44
45     public void displayMaxScorers () {
46         String maxScorers = "";
47         double maxScore = getMaxScore();
48         for (int i = 0; i <= lastStudent; i++) {
49             if (marks[i] == maxScore) {
50                 maxScorers += ", " + names[i];
51             }
52         }
53         System.out.println(maxScorers.substring(1)
54             + " scored the highest ("
55             + maxScore + "/"
56             + maxMarks + ")");
57     }
58
59     public void sortByName () {
60         for (int right = lastStudent; right > 0; right--)
61             for (int i = 1; i <= right; i++)
62                 if (names[i-1].compareToIgnoreCase(names[i]) > 0)
63                     swapRecords(i, i - 1);
64     }
65
66
67     public double getMaxScore () {
68         double max = Integer.MIN_VALUE;
69         for (int i = 0; i <= lastStudent; i++) {
70             max = Math.max(max, marks[i]);
71         }
72         return max;
73     }
74
75     private void swapRecords (int x, int y) {

```

```

76         String tempName = names[x];
77         double tempMark = marks[x];
78         names[x] = names[y];
79         marks[x] = marks[y];
80         names[y] = tempName;
81         marks[y] = tempMark;
82     }
83
84     public static String multiplyString (String s, int n) {
85         String out = "";
86         while (n --> 0)
87             out += s;
88         return out;
89     }
90 }

1  import java.util.Scanner;
2  import java.util.InputMismatchException;
3
4  public class ScoreRecorder {
5      public static void main (String[] args) {
6          Scanner inp = new Scanner(System.in);
7          double maxMarks = 0.0;
8          int numberOfStudents = 0;
9          try {
10             System.out.print("Enter the maximum marks allotted for each
11                 student : ");
12             maxMarks = inp.nextDouble();
13             System.out.print("Enter the total number of students : ");
14             numberOfStudents = inp.nextInt();
15             if (maxMarks <= 0) {
16                 System.out.println("Maximum marks must be positive!");
17                 System.exit(0);
18             }
19             if (numberOfStudents <= 0) {
20                 System.out.println("Number of students must be
21                     positive!");
22                 System.exit(0);
23             }
24             Marksheet sheet = new Marksheet(maxMarks, numberOfStudents);
25             System.out.println("Enter " + numberOfStudents + " students'
26                 names and marks : ");
27             for (int i = 0; i < numberOfStudents; i++) {
28                 String name = "";
29                 while (!inp.hasNextDouble()) {

```

```

27         name += inp.next() + " ";
28     }
29     double marks = inp.nextDouble();
30     if (marks <= 0 || marks > maxMarks) {
31         System.out.println("Marks must be within 0.0 and
32             " + maxMarks + "!");
33         System.exit(0);
34     }
35     sheet.addMarks(name.trim(), marks);
36 }
37 sheet.sortByName();
38 sheet.displayChart();
39 sheet.displayMaxScorers();
40 } catch (InputMismatchException e) {
41     System.out.println("Invalid Input!");
42     System.exit(0);
43 }
44 }

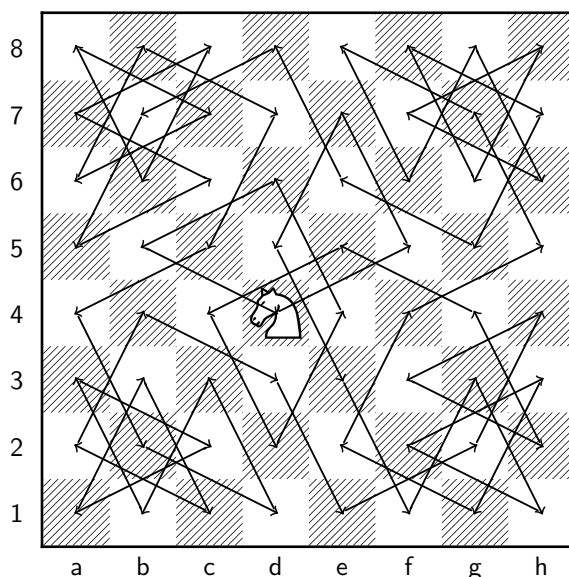
```

*“My project is 90% done. I hope the second half goes as well.”*

— Scott W. Ambler

**Problem 6** A *Knight’s Tour* is a sequence of moves of a knight on a chessboard such that the *knight* visits every square only once. If the knight ends on a square that is one knight’s move from the beginning square, the tour is *closed* forming a closed loop, otherwise it is *open*.

There are many ways of constructing such paths on an empty board. On an  $8 \times 8$  board, there are no less than 26,534,728,821,064 *directed*<sup>1</sup> *closed* tours. Below is one of them.



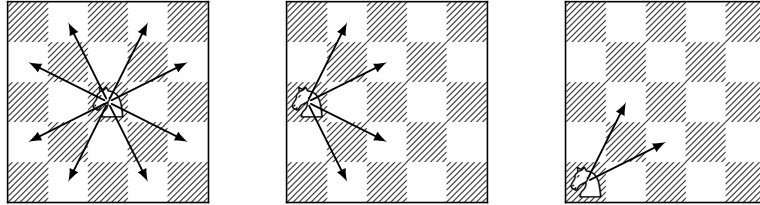
Construct a *Knight’s Tour* (*open* or *closed*) on an  $n \times n$  board, starting from a given square.

(Mark each square with the move number on which the knight landed on it. Mark the starting square 1.)

---

<sup>1</sup>Two tours along the same path that travel in opposite directions are counted separately, as are rotations and reflections.

**Solution** A knight on a chessboard can move to a square that is two squares away horizontally and one square vertically, or two squares vertically and one square horizontally.



## Source Code

```

1 public class TourSolver {
2     private final int size;
3     private Position[] path;
4     private int numberOfMoves;
5     private int[] [] board;
6     private int[] [] degreesOfFreedom;
7     private Position initPosition;
8
9     private static final int[] [] KNIGHT_MOVES = {
10         {-1, -2}, {-1, 2}, {1, -2}, {1, 2},
11         {-2, -1}, {-2, 1}, {2, -1}, {2, 1}
12     };
13
14     public TourSolver (int size, Position initPosition) {
15         this.size = size;
16         this.initPosition = initPosition;
17         this.path = new Position[size * size];
18         this.numberOfMoves = 0;
19         this.initBoard();
20         this.initDegreesOfFreedom();
21     }
22
23     private void initBoard () {
24         board = new int[size][size];
25         for (int i = 0; i < size; i++)
26             for (int j = 0; j < size; j++)
27                 board[i][j] = 0;
28     }
29
30     private void initDegreesOfFreedom () {
31         degreesOfFreedom = new int[size][size];

```

```

32         for (int i = 0; i < size; i++)
33             for (int j = 0; j < size; j++)
34                 degreesOfFreedom[i][j] = getPossibleMovesCount(new
                    Position(i, j));
35     }
36
37     public boolean addMove (Position p) {
38         if (numberOfMoves < (size * size)) {
39             path[numberOfMoves++] = p;
40             board[p.getX()][p.getY()] = numberOfMoves;
41             return true;
42         }
43         return false;
44     }
45
46     public boolean removeMove () {
47         if (numberOfMoves > 0) {
48             Position p = path[numberOfMoves - 1];
49             board[p.getX()][p.getY()] = 0;
50             path[--numberOfMoves] = null;
51             return true;
52         }
53         return false;
54     }
55
56     public int[][] getBoard () {
57         return board;
58     }
59
60     public Position[] getSolution () {
61         if (size < 5)
62             return null;
63         addMove(initPosition);
64         if(solve(initPosition))
65             return path;
66         return null;
67     }
68
69     public boolean solve (Position p) {
70         if (numberOfMoves == (size * size))
71             return true;
72         Position[] possibleMoves = getPossibleMoves(p);
73         if (possibleMoves[0] == null)
74             return false;
75         sortMoves(possibleMoves);
76         for (Position move : possibleMoves) {

```

```

77         if (move != null) {
78             addMove(move);
79             if (solve(move))
80                 return true;
81             removeMove();
82         }
83     }
84     return false;
85 }
86
87 public void sortMoves (Position[] moves) {
88     int count = 0;
89     for (Position p : moves)
90         if (p != null)
91             count++;
92     for (int right = count; right > 0; right--)
93         for (int i = 1; i < right; i++)
94             if (compareMoves(moves[i-1], moves[i]) > 0)
95                 swapMoves(i-1, i, moves);
96 }
97
98 public int compareMoves (Position a, Position b) {
99     int aCount = getPossibleMovesCount(a);
100    int bCount = getPossibleMovesCount(b);
101    if (aCount != bCount)
102        return aCount - bCount;
103    int aFree = degreesOfFreedom[a.getX()][a.getY()];
104    int bFree = degreesOfFreedom[b.getX()][b.getY()];
105    if (aFree != bFree)
106        return aFree - bFree;
107    return (Math.random() < 0.4)? 1 : -1;
108 }
109
110 private static void swapMoves (int x, int y, Position[] moves) {
111     Position t = moves[x];
112     moves[x] = moves[y];
113     moves[y] = t;
114 }
115
116 public Position[] getPossibleMoves (Position start) {
117     Position[] possibleMoves = new Position[KNIGHT_MOVES.length];
118     int i = 0;
119     for (int[] move : KNIGHT_MOVES) {
120         int x = start.getX() + move[0];
121         int y = start.getY() + move[1];
122         if (isWithinBoard(x, y) && board[x][y] == 0) {

```



```

123             possibleMoves[i++] = new Position(x, y);
124         }
125     }
126     return possibleMoves;
127 }
128
129 public int getPossibleMovesCount (Position start) {
130     int i = 0;
131     for (Position p : getPossibleMoves(start))
132         if (p != null)
133             i++;
134     return i;
135 }
136
137 public boolean isWithinBoard (int x, int y) {
138     return (x >= 0 && x < size && y >= 0 && y < size);
139 }
140 }

1 public class Position {
2     private final int x;
3     private final int y;
4
5     public Position (int x, int y) {
6         this.x = x;
7         this.y = y;
8     }
9
10    public Position (String s) {
11        int x = 0;
12        int i = 0;
13        while (i < s.length() && Character.isAlphabetic(s.charAt(i))) {
14            x = (x * 26) + Character.toLowerCase(s.charAt(i)) - 'a' + 1;
15            i++;
16        }
17        int y = Integer.parseInt(s.substring(i));
18        this.x = x - 1;
19        this.y = y - 1;
20    }
21
22    public int getX () {
23        return x;
24    }
25
26    public int getY () {

```

```

27         return y;
28     }
29
30     public boolean equals (Position p) {
31         return (p != null)
32             && (this.getX() == p.getX()) && (this.getY() == p.getY());
33     }
34
35     @Override
36     public String toString () {
37         return xToString(this.x) + (this.y + 1);
38     }
39
40     public static String xToString (int n) {
41         int x = n + 1;
42         String letters = "";
43         while (x > 0) {
44             letters = (char) ('a' + (--x % 26)) + letters;
45             x /= 26;
46         }
47         return letters;
48     }
49 }

1 public class KnightTour {
2     public static void main (String[] args) {
3         try {
4             int boardSize = Integer.parseInt(args[0]);
5             if (boardSize <= 0)
6                 throw new NumberFormatException();
7             String initSquare = (args.length > 1)? args[1] : "a1";
8             TourSolver t = new TourSolver(boardSize, new
9                 Position(initSquare));
10            Position[] solution = t.getSolution();
11            if (solution != null) {
12                showBoard(t.getBoard());
13                showMoves(solution);
14                if (isClosed(solution))
15                    System.out.println("\nThe tour is Closed!");
16            } else {
17                System.out.println("No Knight's Tours found!");
18            }
19        } catch (Exception e) {
20            System.out.print("Enter an integer (> 1) as the first
21                argument, ");

```

```

20         System.out.println("and a well formed chessboard coordinate as
           the second! [size, startSquare]\n");
21         System.out.println("(size      -> Solve a Tour on a (size x
           size) board)");
22         System.out.println("(startSquare -> A square in algebraic
           chess notation of the form 'fr',");
23         System.out.println("           where f = the letter
           representing the file(column)");
24         System.out.println("           and r = the number
           representing the rank(row).)");
25         System.out.println("(startSquare is set to 'a1' by default)");
26     }
27 }
28
29 public static void showBoard (int[] [] board) {
30     String hLine = " " + multiplyString("+-----", board.length) + "+";
31     System.out.println(hLine);
32     for (int column = board.length - 1; column >= 0; column--) {
33         System.out.printf(" %2d ", column + 1);
34         for (int row = 0; row < board.length; row++) {
35             System.out.printf("| %3d ", board[row][column]);
36         }
37         System.out.printf("%n%s%n", hLine);
38     }
39     System.out.print(" ");
40     for (int i = 0; i < board.length; i++) {
41         System.out.printf(" %2s ", Position.xToString(i));
42     }
43     System.out.println();
44 }
45
46 public static void showMoves (Position[] moves) {
47     System.out.print("\nMoves : ");
48     String movesOut = "";
49     for (int i = 1; i < moves.length; i++) {
50         movesOut += (moves[i-1] + "-" + moves[i] + ", ");
51     }
52     System.out.println(movesOut.substring(0, movesOut.length() - 2));
53 }
54
55 public static String multiplyString (String s, int n) {
56     String result = "";
57     while (n --> 0)
58         result += s;
59     return result;
60 }

```

```
61
62     public static boolean isClosed (Position[] path) {
63         int l = path.length - 1;
64         int dX = Math.abs(path[0].getX() - path[l].getX());
65         int dY = Math.abs(path[0].getY() - path[l].getY());
66         return (dX == 1 && dY == 2) || (dX == 2 && dY == 1);
67     }
68 }
```

“To iterate is human, to recurse divine”

— L. Peter Deutsch

**Problem 7** The *determinant* of a square matrix  $A_{n,n}$  is defined recursively as follows.

$$\det(A_{n,n}) = \begin{vmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{vmatrix} = \sum_{j=1}^n (-1)^{i+j} a_{i,j} \cdot \det(M_{i,j})$$

where  $M_{i,j}$  is defined as the minor of  $A_{n,n}$ , an  $(n-1) \times (n-1)$  matrix formed by removing the  $i$ th row and  $j$ th column from  $A_{n,n}$ .

The determinant of a  $(2 \times 2)$  matrix is simply given by

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

For example, the determinant of a  $(3 \times 3)$  matrix is given by the following expression.

$$\begin{aligned} \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} &= a \begin{vmatrix} e & f \\ h & i \end{vmatrix} - b \begin{vmatrix} d & f \\ g & i \end{vmatrix} + c \begin{vmatrix} d & e \\ g & h \end{vmatrix} \\ &= aei + bfg + cdh - ceg - bdi - afh \end{aligned}$$

Calculate the *determinant* of an inputted  $(n \times n)$  square matrix.

**Solution**