

*“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 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    public static boolean isUnique (long number) {
17        int[] count = new int[10];
18        String rawNumber = Long.toString(Math.abs(number));
19        for (int i = 0; i < rawNumber.length(); i++) {
20            int digit = rawNumber.charAt(i) - '0';
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 class Partition {
3     public static void main (String[] args) {
4         try {
5             int n = Integer.parseInt(args[0]);
6             if (n < 1) {
7                 throw new NumberFormatException();
8             }
9             partition(n);
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 n) {
18        partition(n, n, "");
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 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
8             for (int i = 0; i < plaintext.length(); i++) {
9                 char plain = plaintext.charAt(i);
10                char crypt = ' ';
11                if ('A' <= plain && plain <= 'Z') {
12                    crypt = numToChar(charToNum(plain) + shift);
13                } else {
14                    crypt = plain;
15                }
16            }
17        }
18    }
19 }
```

```

16         ciphertext += crypt;
17     }
18     System.out.println(ciphertext);
19 } catch (NumberFormatException e) {
20     System.out.println("Enter an integer "
21         + "as the first argument!");
22 } catch (IndexOutOfBoundsException e) {
23     System.out.println("Enter 2 arguments "
24         + "(shift, plaintext)!");
25 }
26 }
27
28 public static int charToNum (char letter) {
29     return Character.toUpperCase(letter) - 'A';
30 }
31
32 public static char numToChar (int number) {
33     return (char) ('A' + Math.floorMod(number, 26));
34 }
35 }

```

*“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 plaindrome.

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 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 noMatch = true;
8         System.out.println("Palindromes : ");
9         noMatch &= checkWords(sentence);
10        noMatch &= checkSentence(sentence);
11        if (noMatch) {
12            System.out.println("(None found!)");
13        }
14    }
15
16    public static boolean checkWords (String sentence) {
17        boolean noMatch = 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
26         String word = sentence.substring(start, end).trim();
27         if (isPalindrome(word)) {
28             noMatch = false;
29             System.out.println(getAlphabets(word));
30         }
31         start = end - 1;
32     }
33     return noMatch;
34 }
35
36 public static boolean checkSentence (String sentence) {
37     if (isPalindrome(sentence)) {
38         System.out.println(sentence);
39         return false;
40     }
41     return true;
42 }
43
44 public static boolean isPalindrome (String text) {
45     String rawText = getAlphabets(text).toUpperCase();
46     for (int i = 0, j = rawText.length() - 1; i < j; i++, j--) {
47         if (rawText.charAt(i) != rawText.charAt(j)) {
48             return false;
49         }
50     }
51     return (rawText.length() > 1);
52 }
53
54 public static String getAlphabets (String text) {
55     String rawText = "";
56     for (int i = 0; i < text.length(); i++) {
57         if (Character.isAlphabetic(text.charAt(i))) {
58             rawText += text.charAt(i);
59         }
60     }
61     return rawText;
62 }
63 }

```

“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  class Marksheet {
2
3      public static final int SCREEN_WIDTH = 80;
4
5      double maxMarks;
6      int numberOfStudents;
7      int lastStudent;
8
9      String[] names;
10     double[] marks;
11
12     Marksheet (double maxMarks, int numberOfStudents) {
13         this.maxMarks = maxMarks;
14         this.numberOfStudents = numberOfStudents;
15         names = new String[numberOfStudents];
16         marks = new double[numberOfStudents];
17         lastStudent = -1;
18     }
19
20     boolean addMarks (String name, double score) {
21         try {
22             names[++lastStudent] = name;
23             marks[lastStudent] = score;
24             return true;
25         } catch (IndexOutOfBoundsException e) {
26             return false;
27         }
28     }
29
30     void displayChart () {
```



```

31      System.out.println(Marksheet.multiplyString("-",
32                          Marksheet.SCREEN_WIDTH));
33      for (int i = 0; i <= lastStudent; i++) {
34          double fraction = marks[i] / maxMarks;
35          String name = (names[i].length() < 16)
36                        ? names[i]
37                        : (names[i].substring(0,13) + "...");
38          int points = (int) (fraction * (SCREEN_WIDTH - 32));
39          String bar = multiplyString("*", points)
40                    + multiplyString(" ", SCREEN_WIDTH - 32 - points);
41          System.out.printf("| %16s | %s | %5.2f %%%n"
42                          , name
43                          , bar
44                          , fraction * 100);
45      }
46      System.out.println(Marksheet.multiplyString("-",
47                          Marksheet.SCREEN_WIDTH));
48  }
49
50  void displayMaxScorers () {
51      String maxScorers = "";
52      double maxScore = getMaxScore();
53      for (int i = 0; i <= lastStudent; i++) {
54          if (marks[i] == maxScore) {
55              maxScorers += ", " + names[i];
56          }
57      }
58      System.out.println(maxScorers.substring(1)
59                      + " scored the highest ("
60                      + maxScore + "/"
61                      + maxMarks + ")");
62  }
63
64  void sortByName () {
65      for (int right = lastStudent; right > 0; right--)
66          for (int i = 1; i <= right; i++)
67              if (names[i-1].compareTo(names[i]) > 0)
68                  swapRecords(i, i - 1);
69  }
70
71  double getMaxScore () {
72      double max = Integer.MIN_VALUE;
73      for (int i = 0; i <= lastStudent; i++) {
74          max = Math.max(max, marks[i]);
75      }

```

```

75         return max;
76     }
77
78     void swapRecords (int x, int y) {
79         String tempName = names[x];
80         double tempMark = marks[x];
81         names[x] = names[y];
82         marks[x] = marks[y];
83         names[y] = tempName;
84         marks[y] = tempMark;
85     }
86
87     public static String multiplyString (String s, int n) {
88         String out = "";
89         for (int i = 0; i < n; i++)
90             out += s;
91         return out;
92     }
93 }

1  import java.util.Scanner;
2  import java.util.InputMismatchException;
3
4  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
                student : ");
11             maxMarks = inp.nextDouble();
12             System.out.print("Enter the total number of students : ");
13             numberOfStudents = inp.nextInt();
14             if (maxMarks <= 0) {
15                 System.out.println("Maximum marks must be positive!");
16                 System.exit(0);
17             }
18             if (numberOfStudents <= 0) {
19                 System.out.println("Number of students must be
                    positive!");
20                 System.exit(0);
21             }
22             Marksheet sheet = new Marksheet(maxMarks, numberOfStudents);
23             System.out.println("Enter " + numberOfStudents

```

```

24         + " students' names and marks : ");
25     for (int i = 0; i < numberOfStudents; i++) {
26         String name = "";
27         while (!inp.hasNextDouble()) {
28             name += inp.next() + " ";
29         }
30         double marks = inp.nextDouble();
31         if (marks <= 0 || marks > maxMarks) {
32             System.out.println("Marks must be within 0.0
33                                 and " + maxMarks + "!");
34             System.exit(0);
35         }
36         sheet.addMarks(name.trim(), marks);
37
38         sheet.sortByName();
39         sheet.displayChart();
40         sheet.displayMaxScorers();
41     } catch (InputMismatchException e) {
42         System.out.println("Invalid Input!");
43         System.exit(0);
44     }
45 }
46 }

```

“To iterate is human, to recurse divine”

— L. Peter Deutsch

Problem 6 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 (2×2) matrix is simply given by

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

For example, the determinant of a (3×3) 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