Problem 1

Write a program which defines an array of arrays of characters and checks (in a loop), if they could form a valid password. Arrays of characters can be conveniently created from objects of class **String** using the **toCharArray** method, for example:

```
char[][] passwords = {
    "AbcDe93".toCharArray(),
    "A1b:A1b>".toCharArray(),
    "Ab:Acb<".toCharArray(),
    "abc123><".toCharArray(),
    "Zorro@123".toCharArray()
};
```

We assume that a valid password contains

1. at least 8 characters;
2. at least 6 different characters;
3. at least 1 digit;
4. at least 1 uppercase letter;
5. at least 1 lowercase letter;
6. at least 1 non-alphanumeric character (which is neither a letter nor a digit).

For each candidate password, the program displays an appropriate message every time when any of the conditions mentioned above is *not* met. You can assume that all characters are plain ASCII characters with codes from the range '!' (which is 33) through '~' (which is 126; using explicitly ASCII/UNICODE codes shouldn't be necessary though).
For the data as above the result should be something like

```
checking AbcDe93
Too short
No non-alphanumeric character

checking A1b:A1b>
Too few different characters

checking Ab:Acb<
Too short
Too few different characters
No digit

checking abc123><
No uppercase letter
```

```
checking Zorro@123
OK
```

Problem 2

Write a function which takes a position of a chess knight on the chessboard. Position is specified as a string of length 2: the first character denotes the column, or file (from 'a' through 'h') and the second the row, or rank (from '1' through '8'). The function returns a string containing, space separated, positions that are available for the knight in one move.

For example, the following program

download *ChessKnight.java*

```java
public class ChessKnight {
    public static String knightMoves(String pos) {
        // ...
    }

    public static void main (String[] args) {
        for (String s : new String[]{"A1","d5","g6","C8"})
            System.out.println(s + " -> " + knightMoves(s));
    }
}
```

should print

```
A1 -> b3 c2
d5 -> b4 b6 c3 c7 e3 e7 f4 f6
g6 -> e5 e7 f4 f8 h4 h8
C8 -> a7 b6 d6 e7
```

Problem 3

A nucleic acid sequence is a succession of letters describing the order of nucleotides within a DNA molecule; the letters are A, C, G and T (abbreviations come from *adenine*, *cytosine*, *guanine* and *thymine*). A very primitive method of measuring the degree of similarity between two sequences can be described as follows:

- For sequences *of the same length*, for any identical subsequence of length d we add d^2 points; the sum calculated in this way will be the coefficient of similarity. For example, for sequences `ACGTC` and `AGGTG` it will be 5: one point for `A` at the first position and four for a two-character subsequence `GT` at positions 3-4.
- For sequences of unequal lengths, we calculate, as described above, coefficients of similarity between the shorter sequence, of, say, length d, and all subsequences of this length of consecutive elements from the longer sequence. The resulting coefficient is the largest of these partial ones. For example, for sequences `AGG` and `CGGAT`, we calculate the similarity between `AGG` and, one by one, `CGG` (4 points), `GGA` (1 point) and `GAT` (0 point). Hence, the resulting coefficient will be the largest of these values, i.e., 4.

Write function

```
    public static int simil(String a, String b)
```

which takes two sequences (as **Strings**) and returns their similarity coefficient. You may, if you find it useful, define also an auxiliary function. Do not create any arrays or collections and do not use any classes other than **String**. For example, the program

download *DNA.java*

```
public class DNA {
    public static int simil(String a, String b) {
        // ...
    }

    public static void main (String[] args) {
        String a = "AACTACGTC";
        String b =     "ACGTA";
        System.out.println(a + " and " + b +
                           " -> " + simil(a, b));
        String c =   "GCGC";
        String d = "AGGGCA";
        System.out.println(c + " and " + d +
                           " -> " + simil(c, d));
    }
}
```

should print

```
AACTACGTC and ACGTA -> 16
GCGC and AGGGCA -> 5
```

Methods **charAt** and **substring** from class **String** may be helpful.

Problem 4

Define a class **Frac** describing fractions (rational numbers). Objects of the class should have fields num (numerator) and denom (denominator). Write

- Constructor taking numerator n and denominator d and creating an object representing the fraction n/d;
- Constructor taking only one **int** and creating the fraction representing a whole number (with denominator equal to 1);
- Default constructor (with no parameters) creating the fraction corresponding to 0;

Representation of a fraction should be unique. Therefore, independently of arguments passed to the constructor, (private) fields denoting the numerator and denominator must not have any common factors, denominator should always be positive, and if the numerator is 0, the denominator should be 1, to ensure unique representation of zero.

A private member function calculating the greatest common divisor will probably be useful. . .

The class should also define

- static function
  `public static Frac add(Frac a, Frac b)`
  returning object representing the sum of a and b;
- method
  `public Frac add(Frac other)`
  modifying object **this** so it represents the sum of its previous value and other. The method should return object this so their invocations can be „chained";
- analogous static functions and methods **sub** (for subtraction), **mul** (for multiplication) and **div** (for division). Division by zero should provoke an exception of type **ArithmeticException**;

Methods **equals**, **hashCode** and **toString** from class **Object** should be appropriately overriden. The last one shoud return a string containg the numerator and denominator separated with a slash; for whole numbers the slash and denominator should be omitted.

The following **main** function

download *Frac.java*

```
public static void main (String[] args) {
    Frac a = new Frac(20,4),   b = new Frac(1,-2),
         c = new Frac(-14,-4), d = new Frac(-8,-4),
         f = new Frac(4),      m = new Frac(-1);
    Frac f2 = Frac.add(Frac.mul(d,c),
                       new Frac(10).mul(b));
    Frac f4 = Frac.add(Frac.mul(Frac.mul(a,b),
                       new Frac(-2)),m);
    Frac zz = d.add(a).add(f.mul(Frac.mul(b,c)));
    Frac ww = Frac.div(f2,f4).sub(b).sub(c);
    System.out.println(f2 + " " + f4 + " " +
                       zz + " " + ww);
}
```

should print

```
2 4 0 -5/2
```

We used here the fact that *methods* **add**, **sub**, **mul** and **div** return **this** object, what allows us to chain their invocations, e.g.,
`frac.add(f1).mul(f2).sub(f3)`

Remark: generally, objects of a class representing numbers should be immutable (and have its fields declared as **final**). In this problem objects of type **Frac** *are* modifiable mainly for dydactic reasons. . .