Problem-1 The last character of ID

[Problem Description]
Assume the ID of your student card is composed of a sequence of characters as following:

- The ID begins with 8 digits.
- Then, ID is followed by two or three English alphabets.
- The last character is a special character, which is calculated using the following algorithm:

    - Replace the English alphabet in the ID with a numeric value:
      'A' is mapped to value 20; 'B' is mapped to 21; 'C' is mapped to 22; and so on, until 'Z' is mapped to 45.

    - If the ID has 2 alphabet and 8 digits, do the following:
      Let the ID with 10 characters as $C_9C_8C_7C_6C_5C_4C_3C_2C_1C_0$, and calculate the weighted sum as:
        - Sum = $30C_9-29C_8+28C_7+27C_6+26C_5+25C_4+24C_3+23C_2+22C_1+C_0$

    - If the ID has 3 alphabets and 8 digits, do the following:
      Let the ID with 8 characters as $C_{10}C_9C_8C_7C_6C_5C_4C_3C_2C_1C_0$, and calculate the weighted sum as:
        - Sum = $250+21C_{10}+20C_9-19C_8+18C_7+17C_6+16C_5+15C_4+14C_3+13C_2+12C_1+11C_0$

    - Use remainder operator to perform Sum%20.

      The last character is '&' if the result is 17; '@' if the result is 1; otherwise

      the last character is '!'.

[Your task]
Write a Java program to ask user input the first ten or eleven characters of an ID, and then output the last character of ID.
Remark: NO NEED to do error checking.

[Sample outputs]
The underline characters are input by the user.

Sample output 1:

```
Input an ID: 12345678AB
The last character of ID is &
```

Sample output 2:

```
Input an ID: 78945612YZ
The last character of ID is @
```

Sample output 3:

```
Input an ID: 65432178ABC
The last character of ID is !
```

Problem-2 Number Scanner

[Problem Description]
A "number scanner" reads a 4-digit input. The number can be positive or negative. The scanner will display either a 2-digit number or "Invalid input" using the following rule:

- If the input is a positive value, display **the smallest two digits in ascending order**. For example, '12' will be displayed if the input is '3291'.
- If the input is a negative value, display **the largest two digits in descending order**. For example, '93' will be displayed if the input is '-3291'.
- If any of the digits in the 4-digit input is 0, the scanner will display "Invalid input".

[Your task]
Write a program that prompts the user to enter a 4-digit number and then shows the display.

Remark: 1) No error checking is required; 2) **NOT ALLOW to use Java built-in sorting algorithm**.

[Sample output]
The underline characters are input by the user.

```
Enter a 4-digit number: 9910
Invalid input!
```

```
Enter a 4-digit number: 0991
Invalid input!
```

```
Enter a 4-digit number: 1342
The smallest two-digit number is 12.
```

```
Enter a 4-digit number: 7895
The smallest two-digit number is 57.
```

```
Enter a 4-digit number: -1342

The largest two-digit number is 43.
```

```
Enter a 4-digit number: -7895

The largest two-digit number is 98.
```

Problem 1 – Self-dividing and Palindrome Numbers

[Problem Description]
A Self-dividing number is a number that is divisible by every digit it contains and not allowed to contain the digit zero. For example, 128 is a self-dividing number because 128%1=0, 128%2=0, 128%8=0.

A Palindrome number is a number that is same after reverse. For example, 121, 343, 131 are the palindrome numbers.

[Your task]
Write a program that prints the list of Self-dividing and Palindrome numbers within a specified range. The program will first prompt the user to input the upper limit and lower limit. Error message will be displayed if the lower limit is larger than or equal to the upper limit. After that, the program will check if there are any Self-dividing or Palindrome numbers between the specified range. If there is no, the program will display that there is no such number. Otherwise, the program will return all the relevant numbers between the range inclusively.

Note: Your program must use methods as appropriate.
Problem 2 – Pattern matching with regular expression (50%)

[Problem Description]
Given an input string and a pattern, we want to determine if this pattern exists in the string. For example, if the pattern is “apple” and the string is “I want to eat an apple”, there is a match, because “apple” can be found in both the pattern and sequence.

We need to support pattern matching with regular expression which has dot (.) or asterisk (*).

A ‘.’ in a regular expression means that it can be replaced by any single character. For example, the regular expression “a.ple” is matched to the sequence, because ‘.’ can replace ‘p’. The regular expression “ap..e” is also matched to the sequence, because the

first '.' can replace 'p' and the second '.' can replace 'l'. However, the regular expression "app..e" will not match to the sequence, because there is no pattern in the sequence that starts with "app" and end with "e" with two characters in between.

A '*' in a regular expression means that it can be replaced by zero or more of the preceding character. For example, the regular expression "ap*le" is matched to the sequence, because * can be replaced once of preceding characters "p". Similarly, the regular expression "appl*e" is also matched to the list, because * can be replaced zero of the preceding character "l". However, the regular expression "app*e" will not match to the sequence, because there is no pattern in the sequence that starts with "app" and end with "e" with zero or more of the preceding character "p" in between.

[Your task]
Write a program that prompts the user to enter a sequence and a pattern. Display whether there is a match. Your program must use methods as appropriate.

[Sample run]

Easy examples

```
Sequence: Whatever is worth doing is worth doing well
Pattern: well
```

```
Sequence: Whatever is worth doing is worth doing well
Pattern: Well
```

(Note: the match is case sensitive)

```
Sequence: Whatever is worth doing is worth doing well
Pattern: w.ll
```

```
Sequence: Whatever is worth doing is worth doing well
Pattern: w.l
```

Examples with moderate difficulty

```
Sequence: Whatever is worth doing is worth doing well

Pattern: we*ll
```

```
Sequence: Whatever is worth doing is worth doing well

Pattern: w*ll
```

Difficult examples

Remark: Try if you have time.

```
Sequence: Java programming is fun!

Pattern: progra.*ing
```