Topic: Introduction to Java Applications: Methods, Control flow, Values, Variables, Types, Primitive types. Single Dimension Arrays, Multidimensional Arrays

Review

Problem 1. **5 point**

(Sum integers)

Write a program that passes an unspecified number of integers from command line and displays their total.

Problem 2. 5 point

(Occurrences of a specified character)

Write a method that finds the number of occurrences of a specified character in a string using the following header:

```
public static int count(String str, char a)
```

For example, count("Welcome", 'e') returns 2. Write a test program that prompts the user to enter a string followed by a character then displays the number of occurrences of the character in the string.

Problem 3. **10 point**

(Find the number of uppercase and lowercaseletters in a string)

Write a program that passes a string to the command line and displays the number of uppercase letters and lowercase letters in the string.

Problem 4. **10 point**

(Find the index of the smallest element)

Write a method that returns the index of the smallest element in an array of integers. If the number of such elements is greater than 1, return the smallest index. Use the following header:

```
public static int indexOfSmallestElement(double[] array)
```

Write a test program that prompts the user to enter 10 numbers, invokes this method to return the index of the smallest element, and displays the index.

Problem 5. 10 point

(Sort characters in a string)

Write a method that returns a sorted string using the following header:

```
public static String sort(String s)
```

For example, sort("acb") returns abc.

Write a test program that prompts the user to enter a string and displays the sorted string.

Problem 6. 10 point

(Pattern recognition: consecutive four equal numbers)
Write the following method that tests whether the array has four consecutive numbers with the same value:

```
public static boolean isConsecutiveFour(int[] values)
```

Write a test program that prompts the user to enter a series of integers and displays it if the series contains four consecutive numbers with the same value. Your program should first prompt the user to enter the input size—i.e., the number of values in the series. Here are sample runs:

```
Enter the number of values: 8
Enter the values: 3 4 5 5 5 5 4 5
The list has consecutive fours
```

Problem 7. 50 point

(Financial: credit card number validation)
Credit card numbers follow certain patterns. A credit card number must have between 13 and 16 digits. It must start with

- 4 for Visa cards
- 5 for Master cards
- 37 for American Express cards
- 6 for Discover cards

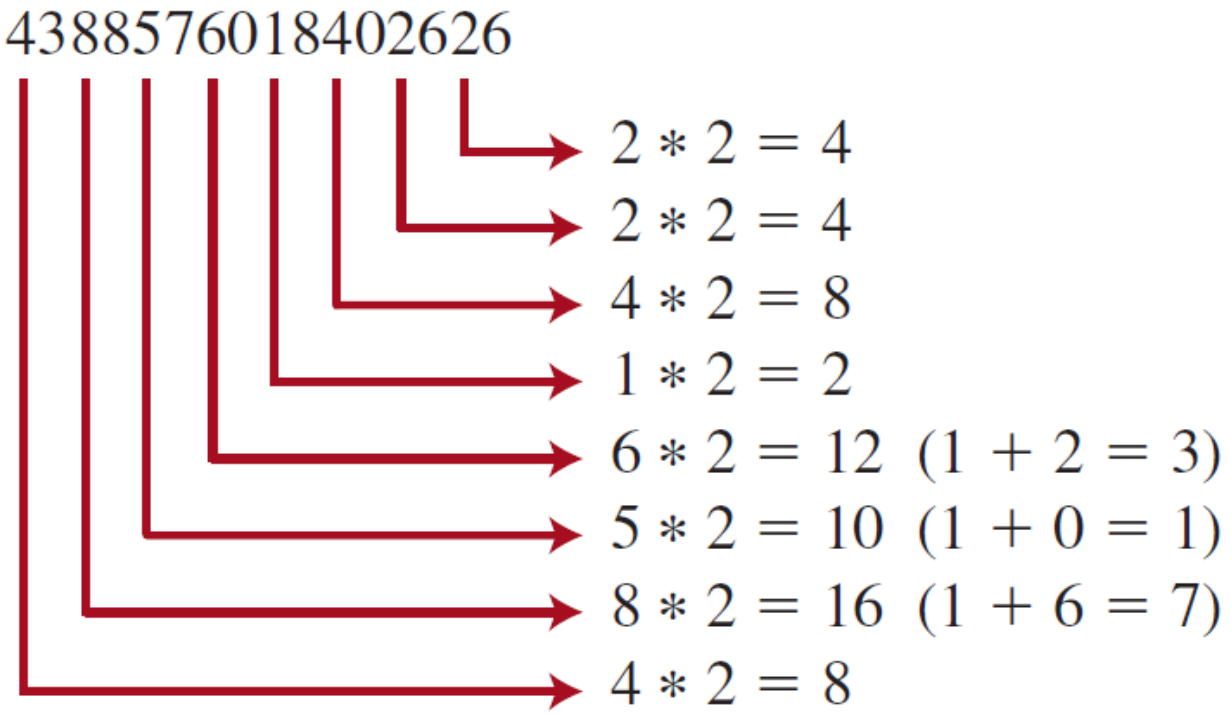
In 1954, Hans Luhn of IBM proposed an algorithm for validating credit card numbers. The algorithm is useful to determine whether a card number is entered correctly, or whether a credit card is scanned correctly by a scanner. Credit card numbers are generated following this validity check, commonly known as the *Luhn* check or the *Mod 10* check, which can be described

as follows (for illustration, consider the card number 4388 5760 1840 2626):

1. Double every second digit from right to left. If doubling of a digit results in a two-digit number, add up the two digits to get a single-digit number.
2. Now add all single-digit numbers from Step 1. $4 + 4 + 8 + 2 + 3 + 1 + 7 + 8 = 37$
3. Add all digits in the odd places from right to left in the card number. $6 + 6 + 0 + 8 + 0 + 7 + 8 + 3 = 38$
4. Sum the results from Step 2 and Step 3. $37 + 38 = 75$
5. If the result from Step 4 is divisible by 10, the card number is valid; otherwise, it is invalid. For example, the number 4388 5760 1840 2626 is invalid, but the number 4388 5760 1841 0707 is valid.

Write a program that prompts the user to enter a credit card number as a long integer. Display whether the number is valid or invalid. Design your program to use the following methods:

```
/** Return true if the card number is valid */
public static boolean isValid(long number)
/** Get the result from Step 2 */
public static int sumOfDoubleEvenPlace(long number)
/** Return this number if it is a single digit, otherwise,
* return the sum of the two digits */
public static int getDigit(int number)
/** Return sum of odd-place digits in number */
public static int sumOfOddPlace(long number)
/** Return true if the number d is a prefix for number */
public static boolean prefixMatched(long number, int d)
/** Return the number of digits in d */
public static int getSize(long d)
/** Return the first k number of digits from number. If the
* number of digits in number is less than k, return number. */
public static long getPrefix(long number, int k)
```


4388576018402626
2 * 2 = 4
2 * 2 = 4
4 * 2 = 8
1 * 2 = 2
6 * 2 = 12 (1 + 2 = 3)
5 * 2 = 10 (1 + 0 = 1)
8 * 2 = 16 (1 + 6 = 7)
4 * 2 = 8

```java
/** Return true if the card number is valid */
    public static boolean isValid(long number) {
        int length = getSize(number);
        return (length >= 13 && length <= 16) &&
               (prefixMatched(number, 4) ||
                      prefixMatched(number, 5) ||
                      prefixMatched(number, 37) ||
                      prefixMatched(number, 6)) &&
               ((sumOfDoubleEvenPlace(number) +
                 sumOfOddPlace(number)) % 10 == 0);
    }

    /** Get the result from Step 2 */
    public static int sumOfDoubleEvenPlace(long number) {
        int sum = 0;
        String numStr = Long.toString(number);
        for (int i = numStr.length() - 2; i >= 0; i -= 2) {
            int digit = Character.getNumericValue(numStr.charAt(i));
            sum += getDigit(digit * 2);
        }
        return sum;
    }
```

```java
/** Return this number if it is a single digit, otherwise return
  the sum of the two digits */
public static int getDigit(int number) {
    if (number < 10) {
        return number;
    }
    return number / 10 + number % 10;
}

/** Return sum of odd-place digits in number */
public static int sumOfOddPlace(long number) {
    int sum = 0;
    String numStr = Long.toString(number);
    for (int i = numStr.length() - 1; i >= 0; i -= 2) {
        sum += Character.getNumericValue(numStr.charAt(i));
    }
    return sum;
}

/** Return true if the number d is a prefix for number */
public static boolean prefixMatched(long number, int d) {
    return getPrefix(number, getSize(d)) == d;
```

```
    }

    /** Return the number of digits in d */
    public static int getSize(long d) {
        return Long.toString(d).length();
    }

    /** Return the first k number of digits from number.
     * If the number of digits in number is less than k, return
       number.
     */
    public static long getPrefix(long number, int k) {
        if (getSize(number) > k) {
            return Long.parseLong(Long.toString(number).substring(0,
              k));
        }
        return number;
    }
```