





****6.30** (*Game: craps*) Craps is a popular dice game played in casinos. Write a program to play a variation of the game, as follows:
 Roll two dice. Each die has six faces representing values 1, 2, ..., and 6, respectively.
 Check the sum of the two dice. If the sum is 2, 3, or 12 (called *craps*), you lose;
 if the sum is 7 or 11 (called *natural*), you win; if the sum is another value
 (i.e., 4, 5, 6, 8, 9, or 10), a point is established. Continue to roll the dice until either
 A 7 or the same point value is rolled. If 7 is rolled, you lose. Otherwise, you win.
 Your program acts as a single player. Here are some sample runs.

<pre>You rolled 5 + 6 = 11 You win</pre>	
<pre>You rolled 1 + 2 = 3 You lose</pre>	
<pre>You rolled 4 + 4 = 8 point is 8 You rolled 6 + 2 = 8 You win</pre>	
<pre>You rolled 3 + 2 = 5 point is 5 You rolled 2 + 5 = 7 You lose</pre>	

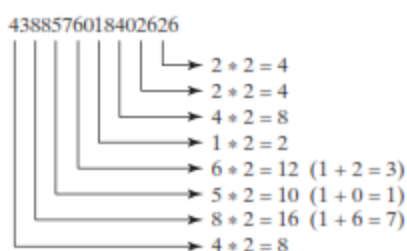
****6.31** (*Financial: credit card number validation*) Credit card numbers follow certain patterns.

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 4388576018402626):

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 4388576018402626 is invalid, but the number 4388576018410707 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 digit 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)
```

Here are sample runs of the program: (You may also implement this program by reading the input as a string and processing the string to validate the credit card.)



```
Enter a credit card number as a long integer:
4388576018410707 Enter
4388576018410707 is valid
```



```
Enter a credit card number as a long integer:
4388576018402626 Enter
4388576018402626 is invalid
```

****6.32** (Game: chance of winning at craps) Revise Exercise 6.30 to run it 10,000 times and display the number of winning games.