

B.Sc. In Software Development. Year 3.

Applications Programming.

Control Structures.



**LIMERICK INSTITUTE
OF TECHNOLOGY**
**SCHOOL OF SCIENCE,
ENGINEERING & I.T.**

Department of Information Technology

Control Statements

- Selection Statements
 - Using if and if...else
 - Nested if Statements
 - Using switch Statements
 - Conditional Operator
- Repetition Statements
 - Looping: while, do, for and for each
 - Nested loops
 - Using break and continue

Selection Statements



- `if` Statements
- `switch` Statements
- Conditional Operators

if Statements

```
if (booleanExpression) {  
    statement(s);  
}
```

Example 1:

```
if (i <= 10) {  
    System.out.print("i is <= 10");  
}
```

Example 2:

```
if ((i >= 0) && (i <= 10)) {  
    System.out.print("i is >=1 but <= 10");  
}
```

The `if...else` Statement

```
if (booleanExpression) {  
    statement(s)-for-the-true-case;  
}  
else {  
    statement(s)-for-the-false-case;  
}
```

if...else Example

```
if (radius >= 0) {  
    area = radius*radius*PI;  
}  
else {  
    System.out.println("Negative input");  
}
```

NOTE: parenthesis are not required when only one statement forms either the if block or the else block.

```
if (radius >= 0)  
    area = radius*radius*PI;  
else  
    System.out.println("Negative input");
```

Nested `if` Statements

The program `TestIfElse.java` on [GitHub](#) (in a NetBeans project called `Source`) demonstrates using nested `if` Statements.

This program reads in number of years and loan amount and computes the monthly payment and total payment. The interest rate is determined by number of years.

Conditional Operator

```
if (x > 0)
    y = 1;
else
    y = -1;
```

is equivalent to

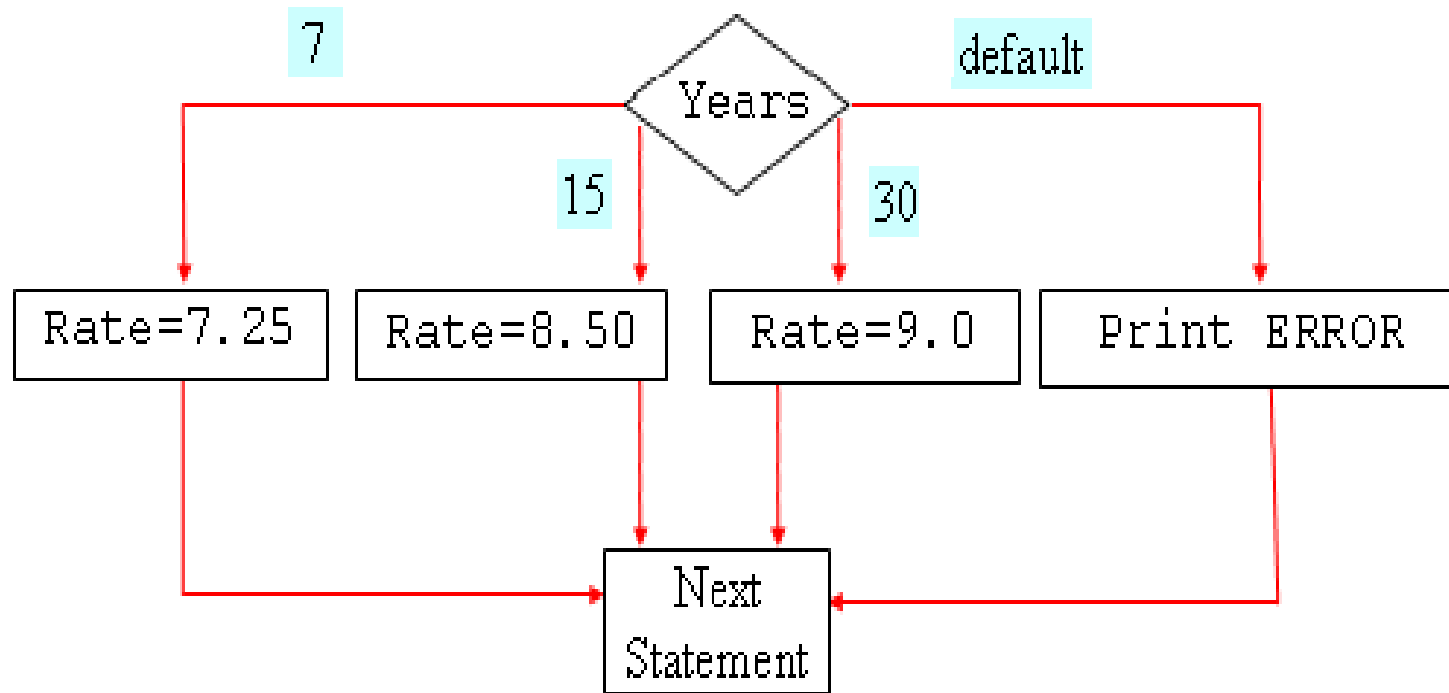
```
y = (x > 0) ? 1 : -1;
```


switch Statements

```
switch (year)
{
    case 7:  annualInterestRate = 7.25;
            break;
    case 15: annualInterestRate = 8.50;
            break;
    case 30: annualInterestRate = 9.0;
            break;
    default: System.out.println(
        "Wrong number of years, enter 7, 15, or 30");
}
```

Consult TestSwitch.java on [GitHub](#) (in a NetBeans project called Source) for the full example.

switch Statement Flow Chart

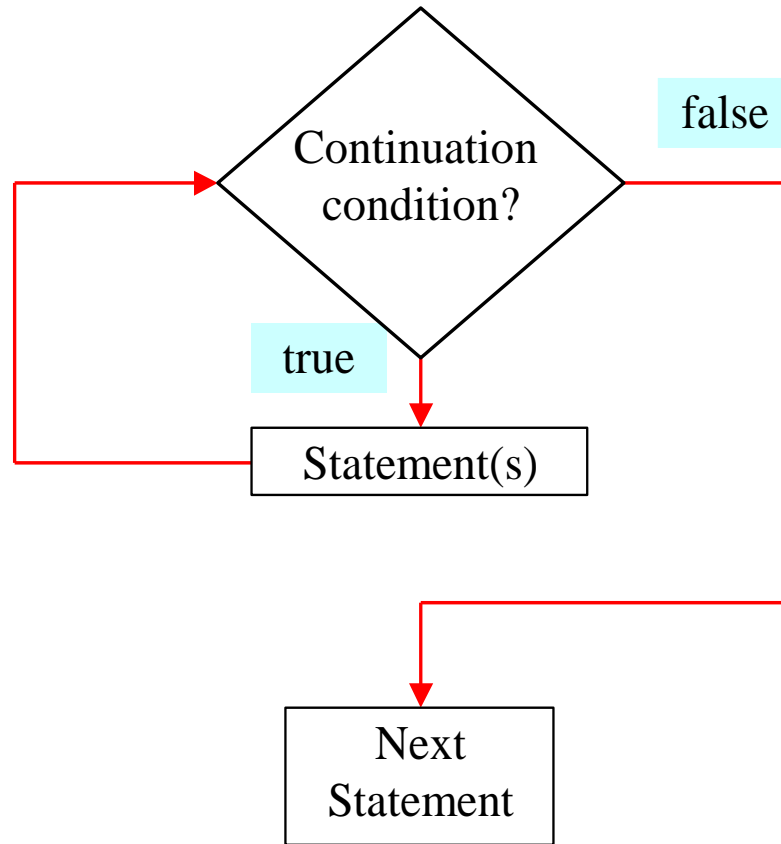


Repetitions

- while **Loops**
- do **Loops**
- for **Loops**
- break **and** continue



while Loop Flow Chart



while Loops

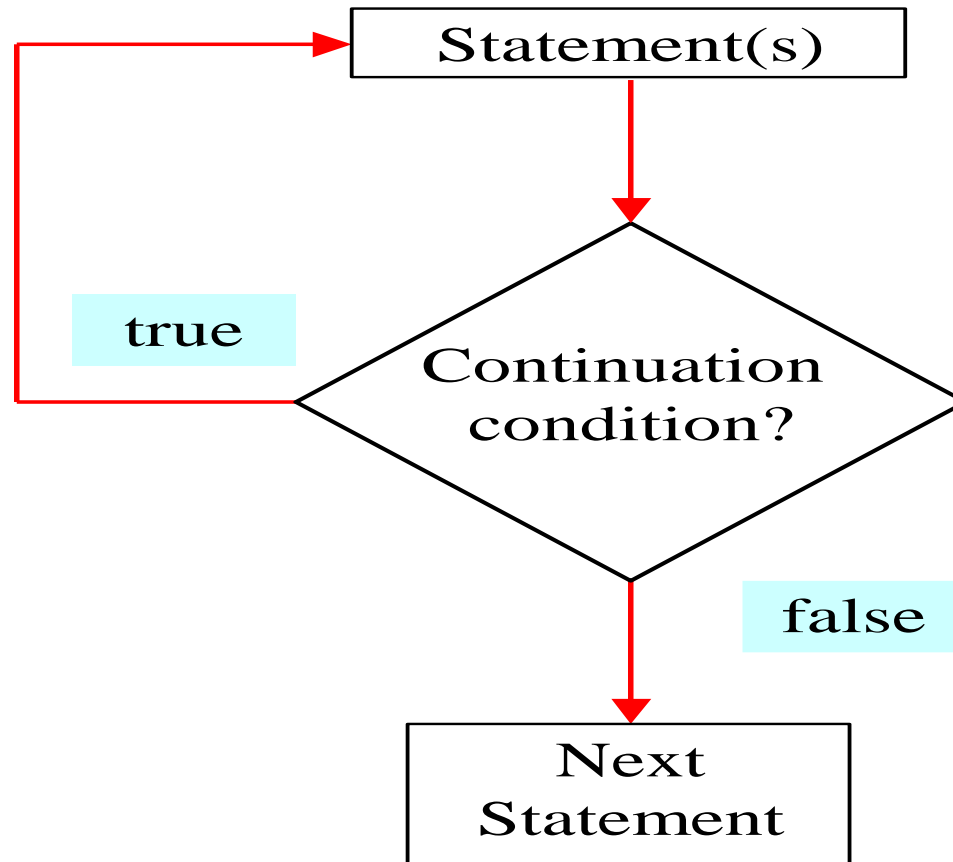
```
while (continue-condition)
{
    // loop-body;
}
```

Consult TestWhile.java on [GitHub](#) (in a NetBeans project called Source) for an example of using while loops.

do Loops

```
do  
{  
    // Loop body;  
} while (continue-condition)
```

do Loop Flow Chart



for Loops

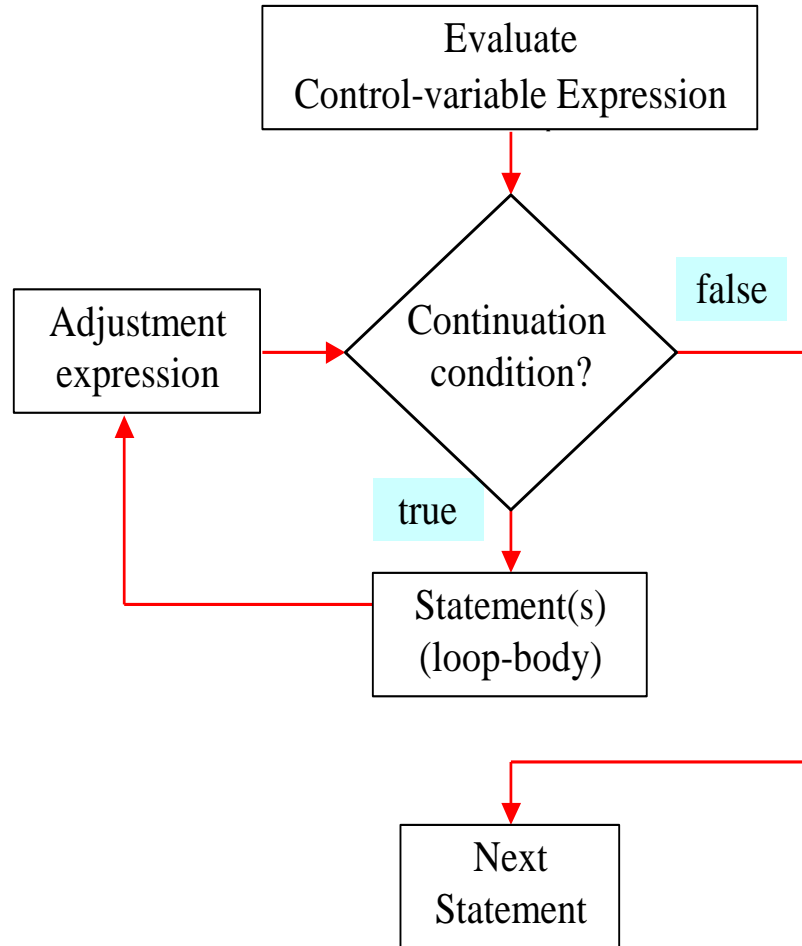
```
for (control-variable-initializer;  
    continue-condition; adjustment-statement)  
{  
    //loop body;  
}
```

```
int i = 0;  
while (i < 100) {  
    System.out.println("Welcome to Java! " + i);  
    i++;  
}
```

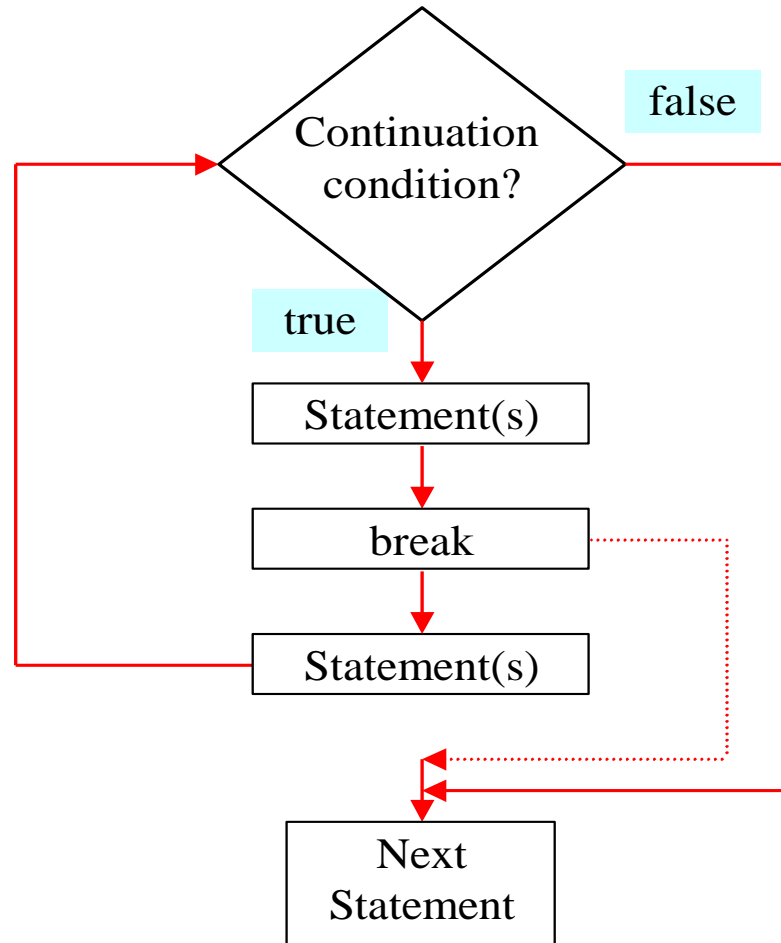
Example:

```
int i;  
for (i = 0; i<100; i++) {  
    System.out.println("Welcome to Java! " + i);  
}
```

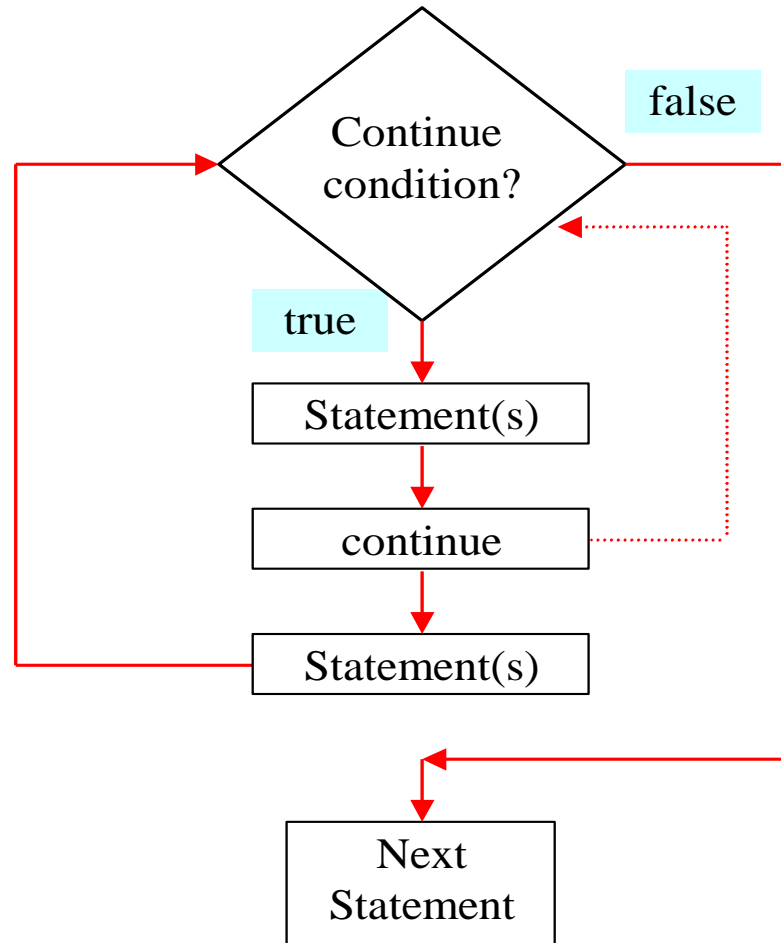

for Loop Flow Chart



The break Keyword



The `continue` Keyword



Using `break` and `continue`

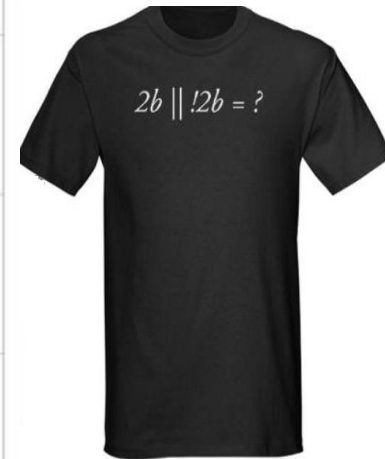
Examples for using the `break` and `continue` keywords:

Consult `TestBreak.java` on [GitHub](#) (in a NetBeans project called `Source`) for an example of using the `break` statement.

Consult `TestContinue.java` on [GitHub](#) (in a NetBeans project called `Source`) for an example of using the `continue` statement.

Full set of conditional operators

Operator	Name	Type	Description
!	Not	Unary	Returns <code>true</code> if the operand to the right evaluates to <code>false</code> . Returns <code>false</code> if the operand to the right is <code>true</code> .
&&	And	Binary	Returns <code>true</code> if both of the operands evaluate to <code>true</code> . Both operands are evaluated before the And operator is applied.
	Or	Binary	Returns <code>true</code> if at least one of the operands evaluates to <code>true</code> . Both operands are evaluated before the Or operator is applied.
^	Xor	Binary	Returns <code>true</code> if one — and only one — of the operands evaluates to <code>true</code> . Returns <code>false</code> if both operands evaluate to <code>true</code> or if both operands evaluate to <code>false</code> .
&&&	Conditional And	Binary	Same as &&, but if the operand on the left returns <code>false</code> , it returns <code>false</code> without evaluating the operand on the right.
	Conditional Or	Binary	Same as , but if the operand on the left returns <code>true</code> , it returns <code>true</code> without evaluating the operand on the right.



For each

The for-each loop was introduced in Java5.

It is mainly used to traverse array or collections.

The advantage of for-each loop is that it eliminates the possibility of bugs and makes the code more readable.

Syntax:

```
for(data_type variable : collection){}
```

For each

Example:

```
17 int values[] = {2, 4, 6};
18
19 for (int x : values) {
20     System.out.print(x + " ");
21 }
22
23 System.out.println("");
24
25 ArrayList<String> wiseMen = new ArrayList();
26 wiseMen.add("Alan");
27 wiseMen.add("Brendan");
28 wiseMen.add("Gerry");
29
30 for (String item : wiseMen) {
31     System.out.println(item);
32 }
```

Output:

```
run:
2 4 6
Alan
Brendan
Gerry
BUILD SUCCESSFUL (total time: 0s)
```

Exercise 1



Write Boolean expressions (on paper) for each of the following:

- X is a positive odd number.
- X is a positive number divisible by 10.
- X is a negative number not divisible by 5.
- month is a legitimate month number.
- X is a number in the range 1 ..1000 that is divisible by 6.
- Given 3 numbers **a**, **b**, **c** - write an expression that states the three numbers are equal.
- Given 3 numbers **a**, **b**, **c** - write an expression that states **a** is the largest of the three numbers.
- Given 3 numbers **a**, **b**, **c** - write an expression that stating that at least two of the numbers are equal.
- Given 3 numbers **a**, **b**, **c** - write an expression that stating that any two of the numbers are equal and not equal to a third.

Exercise 2



Write a program that given any valid date, will generate its successor date.


For example, the input and output would look something like this:

```
run:
Enter Day
1
Enter Month
9
Enter Year
2017
Sucessor Date is 2/9/2017
BUILD SUCCESSFUL (total time: 7 seconds)
```


Exercise 2

Test your code with the following inputs:

```
run:
Enter Day
28
Enter Month
02
Enter Year
2012
Sucessor Date is 29/2/2012
BUILD SUCCESSFUL (total time: 7 sec)
```




```
run:
Enter Day
28
Enter Month
02
Enter Year
2017
Sucessor Date is 1/3/2017
BUILD SUCCESSFUL (total time: 7 s)
```



```
run:
Enter Day
31
Enter Month
08
Enter Year
2017
Sucessor Date is 1/9/2017
BUILD SUCCESSFUL (total time: 12 s)
```



```
run:
Enter Day
31
Enter Month
12
Enter Year
2017
Sucessor Date is 1/1/2018
BUILD SUCCESSFUL (total time: 6 s)
```



Exercise 2

- By finding the maximum number of days in the given month (call it nDays) you can easily generate the successor date.
- Therefore, the task can be divided into two sub-tasks, by firstly determining the maximum number of days in the given month and then using this intermediate value to generate the successor date.

TASK ONE	
Case Analysis	Action
m = 4, 6, 9 or 11	nDays = 30
m = 2	If y is a leap year then nDays = 29 If y is not a leap year then nDays = 28
m = 1, 3, 5, 7, 8, 10, 12	nDays = 31

Exercise 2

TASK TWO (nDays to generate successor date)

Case Analysis	Action
If $d < nDays$	$d = d + 1$
If $d == nDays$	$d = 1$ (new month – and possibly new year).

TASK TWO (new month and possibly new year)

Case Analysis	Action
If $m < 12$	$m = m + 1$
If $m == 12$	$m = 1$ $y = y + 1$

Exercise 3



Accept a positive number (let's call it N) from the user and print the first N rows of the following table.

The following output assumes that $N = 4$.

1			
2	4		
3	6	9	
4	8	12	16

The following output assumes that $N = 9$.

1								
2	4							
3	6	9						
4	8	12	16					
5	10	15	20	25				
6	12	18	24	30	36			
7	14	21	28	35	42	49		
8	16	24	32	40	48	56	64	
9	18	27	36	45	54	63	72	81

Exercise 4



An integer is said to be a perfect number if its factors, including 1 (but not the number itself) sum to the number.

For example:

- 6 is a perfect number because $6 = 3 + 2 + 1$
- 28 is a perfect number because $28 = 14 + 7 + 4 + 2 + 1$

Write a program that prompts the user to enter a number, your program must then determine (and display) if this number represents a perfect number or not.

Exercise 4

Assume you are trying to determine if 28 is perfect (or not).

You need to divide $28/2$. You can discard all numbers above 14 (as they are not factors of 28).

You need to systematically go through each number between 1-14 and check to see if it evenly divides into 28. If it does, then this number is a factor of 28.

You will also need to maintain a running total (of all numbers that are factors of 28). If that running total eventually becomes = 28, then you have a perfect number!

Exercise 4

TRIVIA

There are only 48 known perfect numbers!

The latest perfect number was discovered in 2013.

The following are the first 6 perfect numbers:

6
28
496
8,128
33,550,336
8,589,869,056



Scientists and mathematicians have been unable to determine if there are any odd perfect numbers (for your next assignment you must write a program to discover the first perfect odd number).

Exercise 5

A company wants to transmit sensitive data over the telephone, but they are concerned that their lines are being tapped. All of their data is transmitted as four-digit integers.

They have asked you to write a Java application that will encrypt their data so that it may be transmitted more securely. Your application should read a four-digit integer entered by the user and encrypt it as follows.

You must add 7 to each individual digit and then mod it by 10. Then swap the first digit with the third and then swap the second digit with the fourth digit.

You are then required to display the encrypted integer.

Your program should use a sentinel controlled loop to continually ask the user to enter a four digit number. As soon as the user enters -1, the program should exit.

Exercise 5

A sample run of the encryption program is as follows:

```
run:
```

```
Enter your four digit number to encrypt or -1 to QUIT?
```

```
8524
```

```
Encrypted number is 9152
```

```
Enter your four digit number? (-1 to quit)
```

```
4752
```

```
Encrypted number is 2914
```

```
Enter your four digit number? (-1 to quit)
```

```
-1
```

```
BUILD SUCCESSFUL (total time: 29 seconds)
```

Exercise 5

How to encrypt 8524...

Firstly add 7 to each digit:

$$8 + 7 = 15$$

$$5 + 7 = 12$$

$$2 + 7 = 9$$

$$4 + 7 = 11$$

The mod each new number by 10

$$15 \% 10 = 5$$

$$12 \% 10 = 2$$

$$9 \% 10 = 9$$

$$11 \% 10 = 1$$

Then swap the 1st digit with the 3rd and the 2nd with the 4th, giving us the encrypted number of 9152 which is displayed to the user.

Exercise 5

You must then write a separate program which will take an encrypted number and decrypt it back to its original form.

For example:

```
run:
Enter your four digit number? (-1 to quit)
9152
Decrypted number is 8524
Enter your four digit number? (-1 to quit)
2914
Decrypted number is 4752
Enter your four digit number? (-1 to quit)
-1
BUILD SUCCESSFUL (total time: 19 seconds)
```

References

Joel Murach 2015, *Murach's Beginning Java with NetBeans* Mike Murach & Associates.
ISBN-13 9781890774844 ([Link](#))

Paul Deitel 2017, *Java How To Program (Early Objects) (11th Edition)*. Prentice Hall.
ISBN-13 9780134800271 ([Link](#))

Daniel Liang 2017, *Introduction to Java Programming and Data Structures, Comprehensive Version (11th Edition)*. Pearson.
ISBN-13 978-0134670942 ([Link](#))