Exercises related to Java programming

- When you work with JCreator, it is not necessary to create projects. You can simply open the **.java** files to the editor and compile and execute them.

Kari Laitinen
http://www.naturalprogramming.com
2005-06-07 File created.
2015-11-16 Last modification

These exercises are written for the students that I teach. The exercises are also suitable for the readers of *A Natural Introduction to Computer Programming with Java.*

EXERCISES WITH PROGRAM Game.java

(Note that these are slightly different exercises than those provided in the book.)

Exercise 1:

Now the program presents an integer that is only minimally larger than the integer written by the user of the program. Modify the program so that it outputs an integer that is twice as large as the integer typed by the user. (Here you should also invent a better name for the variable `one_larger_integer`. The character * is the multiplication operator in Java.)

Exercise 2:

Improve the program further so that it prints three integers to the user. The first number is the twice-as-large integer that is calculated in the previous exercise. The other two integers are numbers that follow the twice-as-large integer. For example, if the user types in the integer 144, your program must print numbers 288, 289, and 290. You should also modify the texts printed by the program. If the user types in the integer value 17, the output of the program should look like

```
This program is a computer game. Please, type in
an integer in the range  1 ... 2147483646 : 17

You typed in 17.
My numbers are 34, 35, and 36.
Sorry, you lost the game.
I have more and larger numbers.
```

Exercise 3:

Continue the program so that after the game is played, it lets the user play another game. In the new game the program will again ask an integer from the user. In the second game the program should lose the game so that it presents three integers that are much smaller than the integer provided by the user. The first integer should be half of the integer of the user, the second integer should be half of the first integer, and the third integer should be half of the second integer. The last part of the program should produce the following output when the user types in the value 144.

```
Let's play another game. Please, type in
an integer in the range  1 ... 2147483646 : 144

You typed in 144.
My numbers are 72, 36, and 18.
You won the game!
I have only small numbers.
```

Note that it will be quite easy to make this second game if you copy the statements of the first game and paste the copied program lines to the end of the program. You must, of course, modify the copied program lines so that they work as specified above.

The character / is the division operator which you will need to get a half of an integer.

EXERCISES RELATED TO CALCULATIONS AND DECISIONS

There are different units to express lengths and heights in the world. For example, in the U.S. it is common to use feet and inches to express human heights, whereas in continental Europe meters and centimeters are in use. These units relate to each other so that 1 foot is 30.48 centimeters, 1 inch is 2.54 centimeters, and 1 foot is 12 inches.

Exercise 1:

Write a program with which you can convert a human height given in feet and inches to centimeters. The program should ask the user to type in his or her height in two parts: first the height in feet and then the inches part for the height. (A person can say that his or her height is, for example, 5 feet and 9 inches. That would be 30.48 * 5 + 2.54 * 9 centimeters.

There can be thus two separate input statements in the program. After the program has received the feet and the inches, it should calculate the corresponding value in centimeters and print it to the screen.

To start making this program, you can take, for example, the **Game.java** program and rename it to **Height.java**. Remember also to change the name of the class to `Height`.

You can input the values to `int` variables but the calculation results could be stored in variables of type `double`. (See programs **Miles.java** and **Distance.java**.)

Exercise 2:

Improve your program so that its user can give her or his height either in feet and inches or in centimeters. Before any height values shall be given, the program should ask what kinds of units the user wants to give. The program should have an `if` construct which decides what kinds of calculations will be made. Your program could have the following structure.

```
System.out.print( " This program makes calculations related to your height."
                +  "\n Type 1 to give your height in feet and inches or"
                +  "\n Type 2 to give your height in centimeters. " ) ;

int unit_selection = keyboard.nextInt() ;

if ( unit_selection == 1 )
{

   // Here you can make the calculations already done in the
   // previous exercise.

}
else if ( unit_selection == 2 )
{
   // Here you can ask the height in centimeters and do the necessary
   // calculations
}
```

Conversion from centimeters to feet and inches can be done with the division operator / and with the remainder operator %. When calculating with the division operator / and `int` values, computers always round numbers downwards. The following text clarifies these operators.

The remainder operator %

The remainder operator %, which is sometimes called the modulus operator, is used with integers only. Division operations with integers are sometimes inaccurate because computers do not round numbers upwards. For example, the division operation 11/4, eleven divided by four, would be evaluated to 2 although 3 would be closer to the correct value. Computers do not obey human division rules. In division operations involving integers, computers always round downwards. For this reason, the remainder operator % is sometimes useful. To understand operators / and % properly, below are some correct calculations for you to study.

```
1 / 2   is  0          1 % 2   is  1
2 / 2   is  1          2 % 2   is  0
7 / 4   is  1          7 % 4   is  3
9 / 4   is  2          9 % 4   is  1
14 / 5   is  2         14 % 5   is  4
101 / 10 is 10         101 % 10 is 1
```

Exercise 3:

Improve the `if` construct of your program so that the program says "Wrong unit selection." if the user types something else than 1 or 2 in the beginning.

Exercise 4:

If you have time, search the Internet and find out how to calculate an ideal weight for a person who has a certain height. Make your program to print the user's ideal weight.

EXERCISES RELATED TO LOOPS

Exercise 1:

Write a program that prints a conversion table from miles to kilometers. The program should produce the following output to the screen

```
 miles     kilometers

 10.00      16.09
 20.00      32.19
 30.00      48.28
 40.00      64.37
 50.00      80.47
 60.00      96.56
 70.00     112.65
 80.00     128.74
 90.00     144.84
100.00     160.93
110.00     177.02
```

You can make this program by first making a copy of program **Miles.java** that we have studied earlier. You should use a `while` loop or a `for` loop in your program. If you use a `while` loop, the structure of the loop can look like the following. (In place of the four dots .... you need to put your own statements or expressions. Remember that it is good to indent the statements inside the braces { and }. Indentation means that you write the statements three character positions to the right.)

```
while (  ....  )
{
   ....

   ....

   distance_in_miles  =  distance_in_miles  +  10 ;
}
```

Exercise 2:

Improve the program so that it prints, after the table created in the previous exercise, a table that contains conversions from kilometers to miles. The table could look like the following.

```
kilometers     miles

   10.00        6.21
   20.00       12.43
   30.00       18.64
   40.00       24.86
   50.00       31.07
   60.00       37.28
   70.00       43.50
   80.00       49.71
   90.00       55.92
  100.00       62.14
  110.00       68.35
```

After this exercise is completed, your program should print two conversion tables.

Exercise 3:

Improve your program so that the user can select what kind of conversion table must be printed. In the beginning your program should print the following text.

```
This program prints conversion tables.
Type a letter to select a conversion table

 m   miles to kilometers
 k   kilometers to miles
```

After these lines are printed your program should read one character from the keyboard. According to the character the program should print the correct conversion table.

In this exercise you should add an `if-else if-else` construct to your program, and you must put the loops that were written in previous exercises inside the blocks of the `if-else if-else` construct. You should study program **Likejava.java** to find out how to organize the new version of your program.

Exercise 4:

Add a new selectable feature to your program. By pressing the letter P the user should be able to get a conversion table that contains conversions from pounds to kilograms. Pound is a unit of weight that is used in some countries. One pound is 0.4536 kilograms.

Exercise 5:

Improve the program so that it does not stop after it has printed a conversion table. Instead, it should have the exit from the program a selectable feature. The whole program that is constructed in the previous exercises should be put inside a `while` loop, and inside the `while` loop the program should print the following menu

```
This program prints conversion tables.
Type a letter to select a conversion table

 m   miles to kilometers
 k   kilometers to miles
 p   pounds to kilograms
 x   exit the program
```

One possibility is to use a `while` loop that uses a `boolean` variable in the following way

```
boolean user_wants_to_quit = false ;

while ( user_wants_to_quit == false )
{
   ....
```

Inside the loop the `boolean` variable should be set to value `true` when the user types the letter X.

EXERCISES RELATED TO ARRAYS

Exercise 1:

Program **Reverse.java** stores integers in an array and prints the given integers in reverse order. Make a copy of **Reverse.java** and modify it so that it takes exactly seven integers to the array. Also, the program must ensure that the given integers are in the range from 1 to 39.

In this exercise you need to put an `if` construct inside the `do-while` loop of the program.

Make the program print the given integers in 'normal' order instead of the reverse order.

Exercise 2:

Modify the program so that it is not possible to enter an integer if the number has already been typed in previously.

To do this, you need a loop that checks that the new given integer is not among those integers that have already been given to the program.

You have to put the new loop inside the `do-while` loop that already exists in the program. You could also use a `boolean` variable to store information in the case that a given number has been entered previously. You should put the new loop before the `if` construct that you put

there in the previous exercise.

A `boolean` variable can be given two values: `true` or `false`. The necessary `boolean` variable and the loop could look like:

```
boolean integer_previously_given = false ;

// integer_index now tells how many integers have been given
// before the current integer.

for ( int index_for_previous_numbers = 0 ;
          index_for_previous_numbers < integer_index ;
          index_for_previous_numbers ++ )
{
   if ( given_integers[ index_for_previous_numbers ] ==
                                          integer_from_keyboard )
   {
      System.out.print( "\n   That integer has already been given.\n" ) ;

      //  Here you have to modify the boolean variable.

   }
}
```

In the above loop the used array is renamed to `given_integers`.

After the loop described above you should check in the `if` construct that the given integer is in the range 1 ... 39 and that the integer has not been entered previously. You can include the test of the value of the `boolean` variable into the boolean expression of the existing `if` construct.

Exercise 3:

After you have made the exercises above, you have a program that inputs numbers that could be used in Finnish national lottery game named Lotto.

Now you have to make the computer to generate its lottery numbers. In the folder
`http://www.naturalprogramming.com/javabookprograms/javafilesextra/`
you will find a program named **RandomNumbersInArray.java**. Copy suitable lines from that program to your program so that your program will have automatically generated lottery numbers.

After having done this, you have to improve your program so that it checks how many of the numbers given from the keyboard belong to the generated lottery numbers. You can think that the lottery numbers generated by the computer are the ’correct’ numbers, and by typing in numbers from the keyboard you try to ’play’ the lottery game.

To make testing of your program easier, you should make the program print the generated lottery numbers, so that you can enter some correct numbers from the keyboard.

Exercise 4:

Now the numbers used in the lottery game are in the range 1 ... 39. Try making this range smaller (e.g. 1 ... 14) and see how good results you can get by playing against the computer. In this exercise you should comment out some program lines so that it you cannot see the numbers generated by the computer.

EXERCISES RELATED TO STRINGS

Strings in computer programming are data structures that store textual information. A string usually contains a set of character codes which represent some particular text, such as a name entered from the keyboard, or a line of text from a file. In Java, textual information is stored inside `String` objects. For example, the text of this entire paragraph could be stored inside a `String` object.

Program **Fullname.java** is an example in which `String` objects are created from the texts that are input from the keyboard. Make a copy of that program and do the following exercises.

Exercise 1:

Make the program print how many characters there are in the first name and in the last name. By studying program **StringReverse.java** you can see how to get the length of a string.

After this modification a sample run of the program could look like

```
Please, type in your first name: Kari

Please, type in your last name:  Laitinen

Your full name is Kari Laitinen.

Your first name has 4 characters.
Your last name has 8 characters.
```

Exercise 2:

In addition to the previous printings, make the program print your full name in reverse character order. Again, you can study **StringReverse.java** for help. The new output of the program could look like

```
Your name in reverse order: nenitiaL iraK
```

It will be helpful if you add the following line to your program

```
String full_name = first_name + " " + last_name ;
```

Exercise 3:

Improve the program so that it prints your full name also as hexadecimal character codes. The first character of your name could be printed as a hexadecimal code with the following statement

```
System.out.printf( " %X", (int) full_name.charAt( 0 ) ) ;
```

in which the format specifier `%X` makes the `printf()` method print the `int` value in hexadecimal form. `char` value is converted to `int` before printing.

Your task now is to create a loop that prints all characters of your name in hexadecimal form.To ensure that your program works correctly, you can use the table at

http://www.naturalprogramming.com/for_reference/character_codes.pdf

Exercise 4:

Here your task is to improve the program so that it also prints the characters of your name in random order. One possibility to do this is to use a loop to remove the characters of your name randomly, and print them one by one until there are no characters left to print.

Because `String` objects cannot be modified or characters deleted from an existing object, you could create a `StringBuffer` object of your full name and operate then with that object. It is possible to remove or delete characters of a string that it contained inside a `StringBuffer` object.

The following program lines convert the `String` object to a `StringBuffer` object and remove a random character from the `StringBuffer`.

```
StringBuffer full_name_buffer = new StringBuffer( full_name ) ;

int random_character_index =
          (int) ( Math.random() * full_name_buffer.length() ) ;

char removed_character = full_name_buffer.charAt( random_character_index ) ;

full_name_buffer.deleteCharAt( random_character_index ) ;

System.out.print( "\n " + removed_character + " was removed from "
               +  full_name_buffer ) ;
```

The method named `length()` returns a value that tells how many characters are left in a `StringBuffer`. You could use the boolean expression `( full_name_buffer.length() > 0 )` to check when your `while` loop should terminate.

Exercise 5:

Improve the feature developed in the previous exercise so that the program produces five variations of your name in random character order.

In this exercise you must put the loop created in the previous exercise inside a new loop. You can re-create the `StringBuffer` object inside the new loop.

The program could produce the following output if the full name is "Kari Laitinen"

```
Characters of your name in random order:
    ainKLarntii e
    iret inKnaaLi
    n iiieLtaKnra
    nit iLnKaerai
    a itnLKirniae
```


EXERCISES RELATED TO (STATIC) METHODS

You can find a program named **BigLetters.java** in the folder
`http://www.naturalprogramming.com/javabookprograms/javafilesextra/`
This program demonstrates the use of a static method in a computer program. The program has a method named `print_big_letter()` that is called from the `main()` method. A value of type `char` must be given as a parameter for the `print_big_letter()` method. The method then prints, if possible, the given character as a ’big letter’ to the screen.

Make a copy of **BigLetters.java** to your own folder, and do the following exercises.

Exercise 1:

Improve the program so that it can print also the characters ’D’ and ’E’ as big letters.

For each printable letter the program has a static array of strings that contains the strings that can be used to write the particular letter as a ’big letter’.

While doing this exercise it is good to learn to use Copy and Paste operations with your program editor.

Inside the `print_big_letter()` method, the letter data is printed with a ’foreach’ loop that processes all elements of an array. ’Foreach’ loops are written with the keyword `for` but their structure differs slightly from the ’traditional’ `for` loops. ’Foreach’ loops can only be used

with arrays and other collections. Unfortunately the characters inside a `String` object cannot be processed with ’foreach’ loops. All ’foreach’ loops can be replaced with traditional `for` loops. For example, the ’foreach’ loop

```
for ( String letter_data_line : letter_A_data )
{
   System.out.print( "\n " + letter_data_line ) ;
}
```

could be rewritten as the following traditional `for` loop

```
for ( int line_index = 0 ;
          line_index < letter_A_data.length ;
          line_index ++ )
{
   System.out.print( "\n " + letter_A_data[ line_index ] ) ;
}
```

As you can see, the ’foreach’ loop is shorter, and you do not have define an index variable for it. In the above ’foreach’ loop `letter_data_line` refers in turn to each `String` object stored in the array referenced by `letter_A_data`.

Exercise 2:

Currently the `print_big_letter()` method contains a long `if ... else if ... else if ...` construct. That program structure is slightly complicated as all program blocks inside the long `if ... else if ... else if ...` contain a 'foreach' loop. The structure of the program can be simplified if we write it as follows.

```
static String[] get_letter_data( char given_letter )
{
   String[] letter_data ;

   switch ( given_letter )
   {
      case 'A' : letter_data = letter_A_data ; break ;
      case 'B' : letter_data = letter_B_data ; break ;
      case 'C' : letter_data = letter_C_data ; break ;
      default:   letter_data = unknown_letter_data ;
   }

   return letter_data ;
}

static void print_big_letter( char given_letter )
{
   String[] letter_data = get_letter_data( given_letter ) ;

   for ( String letter_data_line : letter_data )
   {
      System.out.print( "\n " + letter_data_line ) ;
   }

   System.out.print( "\n" ) ;
}
```

The above two methods could replace the original `print_big_letter()` method of the program. A new method named `get_letter_data()` is used to find and return the correct letter data. Method `get_letter_data()` uses a program construct named `switch-case` construct instead of the long `if ... else if ... else if ...` construct. `switch-case` constructs can be used instead of complicated `if ... else if ... else if ...` constructs in some cases. (By comparing programs **Likejava.java** and **Likejavas.java**, you can find more information about `switch-case` constructs.)

In this exercise your task is to modify the program so that you replace the original `print_big_letter()` method with the above two methods. You should also modify the `get_letter_data()` method so that the modifications made in the previous exercise still work.

It is very important that you try to understand how the above two methods work.

Exercise 3:

Write a new method named `print_big_wide_letter()` to the program. This new method should work so that, while the original `print_big_letter()` prints the character 'A' in the following way

```
   xx
  xxxx
 xx  xx
xx    xx
xxxxxxxx
xx    xx
xx    xx
```

the new `print_big_wide_letter()` should print the letter 'A' in the following way

```
      xxxx
    xxxxxxxx
  xxxx    xxxx
xxxx        xxxx
xxxxxxxxxxxxxxxx
xxxx        xxxx
xxxx        xxxx
```

The new method can use the same letter data as the original printing method. You can start making the new method by first making a copy of `print_big_letter()`. In the new method you must print each character of the letter data twice. As characters of a `String` cannot be printed with a 'foreach' loop, you must use a traditional `for` loop, or a `while` loop, to print each character of a line twice.

The following statement prints a character of a `String` twice

```
System.out.print( "" + letter_data_line.charAt( character_index )
                    + letter_data_line.charAt( character_index ) ) ;
```

To test the new method, you must call it from the `main()` method.

Exercise 4:

Improve the program by adding yet another new method. The new method could be named `print_big_word()` and it could begin in the following way

```
static void print_big_word( String given_word )
{
   ...
```

If this new method is called from the `main()` method in the following way

```
print_big_word( "ABBA" ) ;
```

The following should appear on the computer screen

```
   XX     XXXXXXX   XXXXXXX      XX
  XXXX    XX    XX  XX    XX    XXXX
 XX  XX   XX    XX  XX    XX   XX  XX
XX    XX  XXXXXXX   XXXXXXX   XX    XX
XXXXXXXX  XX    XX  XX    XX  XXXXXXXX
XX    XX  XX    XX  XX    XX  XX    XX
XX    XX  XXXXXXX   XXXXXXX   XX    XX
```

Also in this exercise you should use the same letter data that is used by the other methods. Note that in the original letter data definitions there exist the necessary space characters at the end of each `String` of the letter data.

Java Recap Exercises: A program to convert temperature values

There are two common systems for measuring temperature. Degrees of Fahrenheit (ºF) are used in the U.S. and some other countries, while degrees of Celsius (ºC) are in use in most European countries and in many countries throughout the world. The freezing point of water is 0 degrees Celsius and 32 degrees Fahrenheit, 10ºC is 50ºF, 20ºC is 68ºF, 30ºC is 86ºF, and so on. You can see that 10 degrees on the Celsius scale corresponds to 18 degrees on the Fahrenheit scale.

Exercise 1:

Write a program that can convert degrees Fahrenheit to degrees Celsius, or vice versa.

Exercise 2:

Improve your program so that it converts the given numerical value to both Degrees Celsius and to Degrees Fahrenheit. For example, if the user of the program types in 30, your program should say how much 30 ºC is in Degrees Fahreinheit and how much 30 ºF is in Degrees Celsius.

Exercise 3:

Improve your program so that if the user types in the value 0 (zero), the program prints a table which looks like the following

```
 Celsius     Fahrenheit

-20.00        -4.00
-15.00         5.00
-10.00        14.00
 -5.00        23.00
  0.00        32.00
  5.00        41.00
 10.00        50.00
 15.00        59.00
 20.00        68.00
 25.00        77.00
```

You need an if construct to test whether the given value is zero. In addition you must print the temperature table in in a loop. Program Largeint.java is an example where if constructs are used. A while loop is demonstrated in program Whilesum.java. Programs Distance.java and Formatting.java show how to format the output on the screen.

Exercise 4:

Improve your program so that the above temperature table is printed with a separate static method. The method should look like

```
public static void print_temperature_table()
{
   ...
}
```

and it can be called from the main() method in the following way

```
if ( given_temperature  ==  0 )
{
   print_temperature_table() ;
}
```

Program Letters.java is an example that demonstrates how a parameterless method can be called.

Exercise 5:

If you still have time, improve the program so that the temperature table is stored into a file. Program Filecopy.java is an example in which text lines are written to a file. (If file handling is not yet covered in your studies, this exercise might be too difficult.)

EXERCISES WITH PROGRAM GuessAWord.java

You can find a program named **GuessAWord.java** in the folder

`http://www.naturalprogramming.com/javabookprograms/javafilesextra/`

This program is a simple computer game in which the player has to try to guess the characters of a word that is 'known' by the game. Study the program and play the game in order to find out how the game has been programmed.

Exercise 1:

Improve the Guess-A-Word game so that the word to be guessed is randomly taken from an array of `String` objects. Such an array can be created with a statement such as

```
String[] words_to_be_guessed =
                  { "VIENNA", "HELSINKI", "COPENHAGEN",
                    "LONDON", "BERLIN", "AMSTERDAM" } ;
```

A random index for an array such as the one above can be created with the Math.random() method in an expression like

```
(int) ( Math.random() * words_to_be_guessed.length )
```

The Math.random() method returns a `double` value in the range 0.0 .... 1.0 so that the value 1.0 is never returned. The above expression thus calculates a suitable random index. When `double` values are converted to `int` values, they are always rounded 'downwards' to the smaller integer value.

Exercise 2:

Now the program is such that it terminates when the game is finished. Modify the program so that the game can be played several times during a single run of the program. In the above-mentioned folder there is a program named **RepeatableGame.java** which should be a helpful example.

Exercise 3:

Improve the program so that it counts how many guesses the player makes during a game. After a game is played, and before a new game starts, the program should print how many guesses were made. The following variable could be useful in this task

```
int number_of_guesses = 0 ;
```

Exercise 4:

Improve the program so that it prints game statistics before the program terminates. This means that the program shows which words were being guessed and how many guesses were made for each word. The game statistics could look like the following.

```
PLAYED WORD       GUESSES

COPENHAGEN          7
LONDON              6
COPENHAGEN          4
BERLIN              5
HELSINKI            4
```

As the ’played words’ will be randomly selected from an array, it is possible that the same word is played several times.

You can use the following kind of data items to store data of games:

```
int      games_played = 0 ;
String[] played_words = new String[ 50 ] ;
int[]    guesses_in_games = new int[ 50 ] ;
```

A variable of type `int` can be used to count how many games are played, and it can also be used to index the arrays. New data should be put to the arrays after each game is played, and the data should be displayed on the screen in the end when the user no longer wants to play new games.

If you are familiar with classes and objects, you can alternatively use objects to store the above mentioned gaming data.

EXERCISES WITH PROGRAM Animals.java

Exercise 1:

Write a new method named make_stomach_empty() to class Animal in Animals.java. The new method could be called

```
animal_object.make_stomach_empty() ;
```

and it should make stomach_contents reference an empty string "".

Exercise 2:

Add the new data field

```
String  animal_name ;
```

to class Animal in program Animals.java. You have to modify the first constructor of the class so that an Animal object can be created by writing

```
Animal  named_cat  =  new  Animal( "cat", "Ludwig" ) ;
```

You also need to modify the copy constructor so that it copies the new data field. Method make_speak() must be modified so that it prints something like

```
Hello, I am a cat called Ludwig.
I have eaten: ...
```

Exercise 3:

Modify method make_speak() in program Animals.java so that it prints something like

```
Hello, I am ...
My stomach is empty.
```

in the case when stomach_contents references just an empty string. The stomach is empty as long as method feed() has not been called for an Animal object. You can use the standard string method length() to check if the stomach is empty. Method length() can be used, for example, in the following way

```
if ( stomach_contents.length() == 0 )
{
   // stomach_contents references an empty string.
   ...
```

Exercise 4:

Write a default constructor for class Animal in program Animals.java. A default constructor is such that it can be called without giving any parameters. The default constructor should initialize the data fields so that the program lines

```
Animal  some_animal  =  new  Animal();
some_animal.make_speak() ;
```

would produce the following output on the screen

```
Hello, I am a default animal called no name.
...
```

Exercise 5:

Modify program Animals.java so that you add there a new class named Zoo. You can write this new class after the Animal class. Objects of class Zoo should be objects that contain a set of Animal objects. You do not necessarily need a constructor in class Zoo if you initialize the data members when they are declared. The Zoo class should hava a method named add_animal() with which a new Animal object can be added to the zoo. Moreover, the Zoo class should contain a method named make_animals_speak(). Inside this method the make_speak() method should be called for each Animal object. The Zoo class can look like the following:

```
class  Zoo
{
   Animal[]  animals_in_zoo  =  new  Animal[ 20 ] ;

   int  number_of_animals_in_zoo  =  0 ;

   public void add_animal( Animal new_animal_to_zoo )
   {
      ...

   public void make_animals_speak()
   {
      for ( int animal_index  =  0 ;
                animal_index  <  number_of_animals_in_zoo ;
                 animal_index  ++ )
```

```
{
   ...
```

This Zoo class contains an array of type Animal[] which stores references to Animal-objects. The variable number_of_animals_in_zoo is used to count how many animals have been added to the zoo. By studying program Olympics.java, you can find out how an array of objects can be used.

You can test your new Zoo class by writing the following statements to method main():

```
Zoo  test_zoo  =  new  Zoo() ;

test_zoo.add_animal( cat_object ) ;
test_zoo.add_animal( dog_object ) ;
test_zoo.add_animal( another_cat ) ;
test_zoo.add_animal( some_animal ) ;

test_zoo.make_animals_speak() ;
```

Exercise 6:

In this exercise we will modify only the internal structure of class Animal. The functionality of the methods of class Animal may not change although their internal statements will be modified. This means that you do not need to modify the main() method.

Modify the data field stomach_contents in class Animal so that it becomes an array of type String[]. You can write it as follows

```
String[] stomach_contents  =  new  String[ 30 ] ;
```

This array can store 30 references to String objects. When an Animal object is being fed, the given "string of food" should always be stored in the first free array position. To store the strings into this array, you need to have a variable that counts how many times the animal in question has been fed. For this purpose you can use a data field like

```
int number_of_feedings  =  0 ;
```

which can also serve as an array index.

To use the above-defined array, you need to modify the internal statements of the methods in class Animal. For instance, the feed() method must be modified so that the given food is stored to the array. This can be accomplished with a statement like

```
stomach_contents[ number_of_feedings ]  =
                                    food_for_this_animal ;
```

Method make_speak() can find out whether or not the the animal has been fed by checking the value of number_of_feedings. Stomach contents can be printed with a loop that begins in

the following way

```
for ( int food_index  =  0 ;
          food_index  <  number_of_feedings ;
          food_index  ++ )
{
   System.out.print( ...
           // print one "string of food" at a time
```

Exercise 7:

Derive from class Animal a new class named Carnivore. (A carnivore is an animal that eats other animals.) The Carnivore class must contain a new version of method feed(). With this new method it will be possible to feed other animals to a Carnivore object. The new feed() method can begin in the following way:

```
public void feed( Animal animal_to_be_eaten )
{
```

An animal can be eaten so that the data field species_name is copied to the stomach of a Carnivore object. Inside the new feed() method, the data field species_name of the given Animal object can be referred to in the following way:

```
animal_to_be_eaten.species_name
```

The following statement would copy a reference to this data field

```
stomach_contents[ number_of_feedings ]  =
                                  animal_to_be_eaten.species_name ;
```

You can build the new Carnivore class gradually so that you first write the class so that it only has a constructor. After you have found out that the new class works so that you can construct and use Carnivore objects, you can add the new feed() method.

You can write the new class after the Animal class into the existing .java file. (A single .java file may contain several classes if the classes are not declared with the keyword public.)

Please, study the programs BankPolymorphic.java and Windows.java to find out how a class can be derived from an existing class. In Java, a new class is derived with the keyword extends in the following way:

```
class Carnivore extends Animal
{
   ...
```

Your Carnivore class could be tested with statements like:

```
Carnivore tiger = new Carnivore( "..." ) ;
Animal    cow   = new Animal( "..." ) ;

tiger.feed( cow ) ;
```

EXERCISES RELATED TO CLASS LocalDate

When you write Java programs, you can use many standard Java classes in your programs. One such class is named `LocalDate`. With this class you can do various calculations related to dates and our calendar. Example programs which show how the `LocalDate` class can be used include **Apollo11.java**, **ImportantBirthdays.java**, **BadLuckDays.java**, **Weddingdates.java** and **TitanicTimes.java**.

Exercise 1:

Write a program that calculates your current age in years, months, and days. You can accomplish this when you first create `LocalDate` objects in the following way

```
LocalDate  my_birthday  =  LocalDate.of( 1977, 07, 14 ) ;
LocalDate  date_now  =  LocalDate.now() ;
```

By studying program **TitanicTimes.java**, you'll find out how the time difference between two `LocalDate` objects can be calculated. In program **TitanicTimes.java** a method named `until()` is used to create a `Period` object that contains the time difference between two `LocalDate` objects. Note that a `Period` object represents a different concept than a `LocalDate` object.

You can do these exercises so that you start modifying program **TitanicTimes.java**. You should, however, change the names in the program so that they reflect what you are

calculating. You can use the names given above.

Exercise 2:

Improve your program so that it prints a list of your most important birthdays and tells on which day of week those birthdays occur. You should study program **ImportantBirthdays.java** to find out how to do this. Actually, you can copy suitable lines from **ImportantBirthdays.java** and change the names so that they match the names used in the program developed in the previous exercise.

While doing these exercises, you should realize that our calendar is built into the methods of class `LocalDate`. For example, there is a method named `plusDays()` with which it is possible to add days to a date contained inside a `LocalDate` object. The `plusDays()` method returns a new `LocalDate` object. While adding days to a date, the `plusDays()` method takes care of varying month lengths, leap years, etc.

Exercise 3:

Improve your program so that it prints dates when you are 10000 and 20000 days old. The age of a person is 10000 days, when his/her 'conventional' age is approximately 27 years and 4 1/2 months. With this feature in your program, you'll get new days for partying. This feature can be programmed when you increment a day counter and an `LocalDate` object inside a loop, for example, in the following way.

```
//  First we'll make a copy of the origianal birthday object.

LocalDate date_to_increment = LocalDate.of( my_birthday.getYear(),
                                            my_birthday.getMonth(),
                                            my_birthday.getDayOfMonth() ) ;
int  day_counter  =  0 ;

while ( day_counter < 20001 )
{
   // We'll increment the LocalDate object with method plusDays()
   date_to_increment = date_to_increment.plusDays( 1 ) ;
   day_counter  ++  ;

   if ( ( day_counter  %  10000 )  ==  0 )
   {
      // Continue the program in a suitable way.
```

Above, the `LocalDate` object that contains the birthday is first copied. It is important to do this because the original 'birthday object' is needed in later exercises.

Exercise 4:

Improve the feature that you programmed in the previous exercise so that the program tells your exact age—in years, months, and days—on the days when you are 10000 or 20000 days old. After you have done this exercise, your program should produce an output that looks like the following:

```
10000 days old on 2004-11-29 (Monday)  27 years, 4 months, and 15 days.
20000 days old on 2032-04-16 (Friday)  54 years, 9 months, and 2 days.
```

Exercise 5:

Improve your program so that it tells when you are 1000000000 seconds old. Also this feature can be programmed so that you count days starting from your birthday. Each day has 24 * 60 * 60 seconds. 1000000000 seconds will be reached some time after you are 31 years old. Your program should print your age in years, months, and days on the day when you are 1000000000 seconds old. Again you'll have one more day to celebrate!

You can do this exercise first so that you suppose that the day on which you were born is a 'full day' of 24 * 60 * 60 seconds. Thus, you do not need know the exact hour of day when you were born, or to count any seconds in this exercise. Later on, if you want, you can make a more accurate calculation, in which you take into account the exact time of day when you were born. In such a calculation it might be better to use the standard Java class `LocalDateTime`.

EXERCISES WITH PROGRAM Olympics.java

Exercise 1:

Modify the program so that you replace the existing `olympics_table` with the alternative `olympics_table` that is given in comments at the end of the program file. This can be done with Copy and Paste operations. In this exercise you can still let the 'absurd' `Olympics` object be at the end of the table.

Update also the olympics table so that the latest known Olympic games have Olympics objects created.

Ensure that the program works correctly after these modifications.

Exercise 2:

Now the olympics table contains an 'absurd' `Olympics` object with the year 9999. This 'absurd' `Olympics` object is used to mark the end of the data in the table.

After you have done the previous exercise, the length of the olympics table is such that the last `Olympics` object is referenced from the last position in the table. There is thus no need to have the 'absurd' object to mark the end of data. Instead, it is possible to use the `length` property to check when the end of table has been reached.

In this exercise your task is to remove the 'absurd' `Olympics` object from the table, and modify the `if` construct of the program so that it checks first if the end of table has been reached. The `if` construct inside the `while` loop could begin in the following way:

```
if ( olympics_index >= olympics_table.length )
{
   //  The end of olympics data has been reached.
```

The operation of the program should not change in this exercise. The purpose is just to make the structure of the program better.

Exercise 3:

Create a new class named `WinterOlympics` so that the new `WinterOlympics` class will be a subclass of the original `Olympics` class. The definition of the `WinterOlympics` class should begin

```
class  WinterOlympics extends Olympics
{
   ...
```

You can write the new `WinterOlympics` class after the `Olympics` class in the program file. With the keyword `extends` we can make `WinterOlympics` to inherit the `Olympics` class. We can say also that class `WinterOlympics` is *derived* from the `Olympics` class.

Usually a superclass must have a default constructor because the contructor of superclass is

executed automatically before the constructor of the derived class. Therefore, you must add the constructor

```
public Olympics() {} // Empty default constructor
```

to class `Olympics`.

In this exercise you can make the `WinterOlympics` class such that it behaves in the same way as its superclass `Olympics`. You just need a new constructor inside the the new class. You can make it easily by copying the constructor of class `Olympics` and renaming it. The constructor of a class has the same name as the class itself.

In this exercise the `WinterOlympics` class will be such that it contains only the constructor.

You can test your new class by adding the following object to the table

```
new WinterOlympics( 2006, "Torino", "Italia" )
```

If your program can find the data of Torino olympics, you have successfully carried out this exercise.

Exercise 4:

Improve the new `WinterOlympics` class by writing a new version of method `print_olympics_data()` into it. The new method should print a text that contains the word 'winter'. The output of the method could look like the following

```
In 2006, Winter Olympics were held in Torino, Italy.
```

In this exercise you can copy the corresponding method from class `Olympics`, and modify the text that is generated by the method.

When a subclass contains a method that has the same name and similar parameters as a method in the superclass, we say that the method is *overridden* in the subclass. In this exercise we override the method `print_olympics_data()` and the new version of the method will be automatically be used for `WinterOlympics` objects.

Exercise 5:

Earlier the winter and summer olympics were orginized during the same year. For example, in 1984 the winter olympics were in Sarajevo, Yogoslavia, and the summer olympics were in Los Angeles, U.S.A.

If you add `WinterOlympics` objects that describe these earlier winter games to the olympics table of the program, there will be problems. The search algorithm finds only the first olympics that were held in the given year. If there are two objects for the same year, the latter object will not be found.

Modify the program so that it will always process all objects referenced from the olympics table, and print data of those objects that contain the given year.

After this modification there will be the problem that the program does not know whether or not it could find a suitable olympics object in the table. To solve this problem you might use the following variable

```
boolean olympics_data_was_found = false ;
```

You can give this variable the value `true` after a suitable olympics object is found. If the value of this variable is `false` in the end, it means that information for the given year was not found in the table.

To test the new version of the program you should add more `WinterOlympics` objects to the table.

Exercise 6:

Improve the program so that if the user types in 0 as the year, the program will print data of all summer olympic games. Then, if the user types in 1, the data of all winter games will be printed.

You can solve this problem by using the `instanceof` operator of Java. With the `instanceof` operator you can check whether an object in the olympics table is of type `WinterOlympics`. The boolean expression in the following `if` construct will be `true` if `olympics_object` refers to an object that is of type `WinterOlympics` or of some subtype of `WinterOlympics`.

```
if ( olympics_object instanceof WinterOlympics )
{
   ...
}
```

Then, if you would like to know whether some olympics object is of type `Olympics`, you can use an `if` construct such as

```
if ( ! olympics_object instanceof WinterOlympics )
{
   ...
}
```

Note that you cannot use the instanceof operator to test whether an object is of type `Olympics`. The reason for this is that `instanceof` returns `true` also when the object on its left side is an object of some subclass of the class given on the right side of the operator.

EXERCISES WITH PROGRAM MonthCalendars.java

A program named **MonthCalendars.java** can be found in the folder
`http://www.naturalprogramming.com/javabookprograms/javafiles3new/`
This program has a menu from which the user can select a few operations related to month calendars. The program contains a class named `EnglishCalendar`, which is inherited by another class called `SpanishCalendar`. Class `EnglishCalendar` contains, among other things, a method named `print()` that prints the calendar of the month that is specified in data members. The `print()` method, that may look quite complicated, is inherited by the lower classes and does not need to be modified in these exercises.

Play first with the program and find out how it works.

Exercise 1:

At the beginning of the `main()` method two calendar objects are created. Modify the year and month parameters used in these definitions to find out how calendar objects are created.

Exercise 2:

By typing letter ’n’ the user can now print the calendar of the next month. By typing ’p’ the user can print the current calendar. Modify the program so that, instead of current calendar, the selection ’p’ prints the calendar of the previous month.

To achieve this, you must add a new method named `decrement_calendar_month()` to class

`EnglishCalendar`. This can be done quite easily by first making a copy of the `increment_calendar_month()` method.

Exercise 3:

Define a new class named `GermanCalendar` into the program. You can make this new class a subclass of `EnglishCalendar`. The new class can be similar to the `SpanishCalendar` class so that it only needs a new constructor as the methods are inherited from its superclass `EnglishCalendar`. In the new `GermanCalendar` class you can use the following definitions

```
String[]  german_names_of_months  =

   { "Januar", "Februar", "Marz", "April",
     "Mai", "Juni", "Juli", "August",
     "September", "Oktober", "November", "Dezember" } ;

String   german_week_description  =

     "Woche   Mon  Die  Mit  Don  Fre  Sam  Son" ;
```

You should test your new class by defining a `GermanCalendar` object and calling the `print()` method for it.

If you prefer, you can make, instead of a `GermanCalendar` class, some other calendar class in this exercise. The important thing here is that the new class must be a subclass of `EnglishCalendar`.

Exercise 4:

Improve the program so that there will be a new selectable item in the menu of the program. When the user types ’g’ the program should switch to the use of German calendars.

Exercise 5:

Add a new menu item so that by typing ’y’ the user can see the calendar of the first month of the next year (in relation to the calendar that is currently shown).

Exercise 6:

Modify the program so that when it starts executing it shows the English calendar of the current month. This requires that you get the current year and month from the computer. This can be achieved with the following program lines.

```
LocalDate current_system_date = LocalDate.now() ;

int current_year = current_system_date.getYear() ;
int current_month = current_system_date.getMonthValue() ;
```

EXERCISES RELATED TO CLASS ArrayList AND FILES

With the following exercises we will learn to use the standard Java class named `ArrayList`. Objects of class `ArrayList` are dynamic arrays. Unlike conventional arrays, dynamic arrays can 'grow' when new objects are inserted to the array.

We will also study how a program can write text lines to a file. Java provides standard classes with which files can be handled.

The goal of these exercises is that you will build a kind of text editor program, i.e., you can produce a new text file by running your program.

Exercise 1:

To begin, write a program that inputs text lines from the keyboard and stores the text lines to an `ArrayList`-based array. The needed `ArrayList` can be specified with the statement

```
ArrayList<String> given_text_lines =
                                  new ArrayList<String>() ;
```

This statement creates an `ArrayList` array that can be used to store `String` objects, i.e., the text lines that will be given from the keyboard. You need the following import statement in your program

```
import java.util.* ;  // ArrayList, Scanner, etc.
```

Your program should work so that it reads text lines from the keyboard until the user gives a line that contains a full stop '.' as the first character of the text line. This kind of line is a sign for the program that it should stop reading more text lines. The following could be the structure of your loop.

```
boolean user_wants_to_type_more = true ;

while ( user_wants_to_type_more == true )
{
   String text_line_from_user = keyboard.nextLine() ;

   if ( text_line_from_user.length() > 0 &&
        text_line_from_user.charAt( 0 ) == '.' )
   {
      user_wants_to_type_more = false ;
   }
   else
   {
      // Here you should add the string to the
      // ArrayList array.
   }
}
```

Class `ArrayList` has a method named `add()` with which you can add a new element to the end of the array.

To ensure that your input loop works, you can add the following loop to your program, after the input loop:

```
System.out.print( "\nGIVEN LINES: \n" ) ;

for ( String text_line : given_text_lines )
{
   System.out.print( "\n" + text_line ) ;
}
```

The above loop is a ’foreach’ loop that prints all the text lines in the `ArrayList` array. When you have this loop in your program, it should output all the text lines that were input in the first loop.

Exercise 2:

After having completed the previous exercise, you can be sure that you have the given text lines in an `ArrayList` array. Now your task is to store the text lines to a file. By studying the program **Findreplace.java** of the book, you will find out how the text lines of an `ArrayList` array can be stored to a file. In fact, **Findreplace.java** contains a ready-to-use method for this purpose. You can copy this method to your program, and call it to store the text lines of your `ArrayList` array. You can store the text lines to a file named **test.txt**. You need to have the following import statement in your program

```
import java.io.* ;    // Classes for file handling.
```

To ensure that your text lines have indeed been stored to a file, you can use a text editor, or you can use the following command in a command prompt window

```
type test.txt
```

Exercise 3:

Improve your program so that after it has input the actual text lines, it will ask a file name to which the text will be stored, i.e., the text lines will no longer be automatically stored to a file named **test.txt**.

Exercise 4:

Improve your program so that it will ensure the following things related to the given file name:

- If a file with the given name already exists, the program will ask a new file name.
- The program will not accept an empty string as a file name.
- The program accepts only file names that end with ".txt".

To test all these things you need a loop that begins in the following way (The first two tests are already written there.).

```
String given_file_name = "notknown.txt" ;

boolean acceptable_file_name_given = false ;

while ( acceptable_file_name_given == false )
{
   System.out.print(
            "\nGIVE FILE NAME TO STORE THIS TEXT: " ) ;

   given_file_name = keyboard.nextLine() ;

   if ( new File( given_file_name ).exists() )
   {
      System.out.print( "\nThis file already exists!" ) ;
   }
   else if ( given_file_name.length() == 0 )
   {
```

Exercise 5:

To complete your program, make it such that it automatically prints the contents of the new created text file. You should write a method that can be called to print the contents of a text file. The method could be called in the following way at the end of your program:

```
System.out.print( "\nTHE FOLLOWING IS YOUR NEW FILE:\n" );
print_text_file( given_file_name ) ;
```

By studying the example program **Fileprint.java**, you will find out how to write the needed method.