Linnéuniversitetet
Linnaeus University

Problem Solving and Programming

Assignment 2
In the second assignment the focus is on many of the fundamental parts of programming – iteration, methods and arrays. In order to complete them, you need to have a firm understanding on the parts that were covered by assignment 1. This part is also an important foundation before object orientation is covered in the next assignment, so it is important to understand it.

Carefully read the assignment rules as defined on Moodle. In brief, do the assignments yourself, hand them in at the deadline and don't cheat.

Problems? Do not hesitate to ask your teaching assistant at the practical meetings (or the teacher at the lectures) if you have any problems. You can also post a question in the assignment forum in Moodle.

Validating input You only have to validate the input if the task explicitly says you have to. If nothing is mentioned, you can assume that the user will input the correct type of value.

Submission We are only interested in your .java files. Hence, zip the directory named `YourLnuUserName_assign2` (inside directory named `src`) and submit it using the Moodle submission system.

2 Iteration

The first tasks in this section will deal only with iteration (and sequence) but it will become increasingly more complex when selection is added.

2.1 Printing Pink Floyd

The British artists Pink Floyd are among the most successful bands of all time with classics like *Dark Side of the Moon* and *The Wall*. To celebrate them, you are going to create a program that you call `PinkFloyd.java` where you print *Pink Floyd rules!* five times using iteration. See below for an example.

```
Pink Floyd rules!
Pink Floyd rules!
Pink Floyd rules!
Pink Floyd rules!
Pink Floyd rules!
```

2.2 Every other number

Write a program called `EveryOther.java` that asks the user for a start and end integer and the prints every other number between the first (inclusive) and the second (inclusive if in the range) number. Notice that we can assume that the user inputs a range (with the first number being smaller than the second). For this task, only iteration should be used – no selection. See below for an example run.

```
First number: 5
Second number: 20
5 7 9 11 13 15 17 19
```

2.3 Counting Numbers

Create a program called `CalcNumbers.java` that asks for integers (positive or negative) until the user enters 0. When 0 is entered, the program should display the sum of all the numbers that were entered. See an example execution below:

```
Give me a number: 5
Give me a number: 3
Give me a number: -6
Give me a number: 4
Give me a number: 0
The sum is: 6
```

2.4 Dangerous Work

You have decided to take on some very dangerous work with some rather strange payment. For each day you stay at the work, the salary is doubled – it will begin with 0.01 of a Swedish krona (also called 1 "öre") the first day and it will be 0.02 the second, 0.04 the third, 0.08 the fourth and so on. Write a program called `DangerousWork.java` that helps you calculate your salary based on number of days you work. You do not need to validate the input, but you should keep in mind that you probably won't survive working more than 30 days... See an example of the program running below:

```
How much would you like to earn? 1000000
You will have your money in 27 days.
```

2.5 Right-angled triangle

Write a program called `SimpleTriangle.java` that reads a positive integer from the user. If the user enters a valid number, a *right-angled* triangle of that height should be displayed on screen. If not, print a message stating that the number was not

valid. The program does not need to do more checking than that, and it does not need to start over when done. See below for an example execution:

```
Enter a positive number: 7
*******
 ******
  *****
   ****
    ***
     **
      *
```

2.6 Diamonds

This task is to create an *diamond of stars* by having the user input a positive number which corresponds to the height till the middle of the diamond. In this task the user will be able to create new diamonds until a negative number is entered and you should also check for validity of the input. This task should be completed *without* using `StringBuilder` to create strings, but rather by printing stars directly. Call the program `Diamonds.java` and see an example execution on the next page.

```
Give a positive number: 5
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

```
Give a positive number: 3
  *
 ***
*****
 ***
  *
Give a positive number: -3
```

2.7 Second Largest

Write a program `SecondLargest.java` that reads 10 integers from the keyboard and then prints the second largest one. An example of an execution:

```
Provide 10 integers: 67 -468 36 1345 -7778 0 34 7654 45 -666
The second largest is: 1345
```

Try to design the program such that changing the number of integers to be read (10) is easy. **Recommendation**: Use a smaller value than 10 while developing the program. **Notice**: You are not allowed to use arrays or any other data structure for storing all the integers.

2.8 Histogram

Create a program called `Histogram.java` that simulates the rolling of dice and presenting the results in a histogram. The program should roll 500 dice and present the number of ones, twos, threes and so on by showing the corresponding number of stars. For this task no arrays are to be used, see an example of the running program below:

```
One   (65): *****************************************************************
Two   (80): ********************************************************************************
Three (86): **************************************************************************************
Four  (107): *****************************************************************************************************
Five  (91): ******************************************************************************************
Six   (71): ***********************************************************************
```

2.9 Birthday Candles – VG-task

Exercise 2.9 is marked as VG task → only mandatory for those of you that aspire for a higher mark (A or B).

Write a program `BirthdayCandles.java` that computes how many boxes of candles a person needs to buy each year for his birthday cake. You can assume that the person reaches an age of 100, the number of candles used each year is the same as the age, that you save non-used candles from one year to another, and that each each box contains 24 candles. Also, at the end, we want you to print the total number of boxes one has to buy, and how many candles that are available after having celebrated the 100th birthday. An example of an execution:

```
Before birthday 1, buy 1 box(es)
Before birthday 7, buy 1 box(es)
Before birthday 10, buy 1 box(es)
Before birthday 12, buy 1 box(es)
Before birthday 14, buy 1 box(es)
...
Before birthday 95, buy 3 box(es)
Before birthday 96, buy 4 box(es)
Before birthday 97, buy 5 box(es)
Before birthday 98, buy 4 box(es)
Before birthday 99, buy 4 box(es)
Before birthday 100, buy 4 box(es)

Total number of boxes: 211, Remaining candles: 14
```

Notice: In our example we only have a print-out of those birthdays where you must buy boxes. In the non-printed years (e.g. 2–6 and 8–9) you can handle the birthdays without having to buy any more candles.

Methods

After sequence, selection and iteration, the next important and often present part in a programming language is *modularisation*. In Java this is primarily done using classes, but an important part of the class is the *method*. This part of the assignment will deal with methods without necessarily use other classes than the main one.

2.10 Palindrome Method

Create a program called `Palindrome.java` that has a method that checks if a word is a *palindrome*, that is – the same forwards and backwards. The method should take one string as ingoing parameter and return `true` or `false` depending on if the string is a palindrome or not. The method should remove all spaces and make all letters lower case before checking. The program should continue to run until the user enters `stop` as the string to check. See below for an example run:

```
Enter a word or sentence: Euston
That wasn't a palindrome
Enter a word or sentence: Sue
That wasn't a palindrome
Enter a word or sentence: Euston saw I was not Sue
That was a palindrome.
Enter a word or sentence: stop
```

2.11 Prime Number

In this task you will create a program called `PrimeNumber.java` that will ask the user for a positive integer. In the program you should have a methodthat returns whether the number is a prime or not. The program must continue until the user enters a negative number. See next page for an example execution of the program.

```
Give me a number: 45
This is not a prime number.
Give me a number: 43
This is a prime number.
Give me a number: 7
This is a prime number.
Give me a number: -6
Good bye!
```

2.12 Date Format

A date can, unfortunately, be written in several different ways. In Sweden, the most common way of writing a date is on the format `yyyy/mm/dd` where y is year, m is month and d is day. Other possible ways is to use `dd/mm/yyyy` and even `mm/dd/yyyy`. These formats can be called *big-endian*, *little-endian* and *middle-endian* respectively. Write a program called `DateFormat.java` that accepts a year (a number), a month (a number) and a day (a number) and also asks the user to choose between the endian types. The date format should be returned as a string from a method that takes the year, month, day and format as input. You need not check the validity of the numbers, but if, for example, month 8 (August) is entered, the month should be `08` in the returned string. The program only needs to run once, see below for an example:

```
Enter a year: 1977
Enter a month (number): 5
Enter a day (number): 25
Enter a format (b/l/m): b
1977/05/25
```

2.13 Password – VG-task

Exercise 2.13 is marked as VG task → only mandatory for those of you that aspire for a higher mark (A or B).

Create a program called `Password.java` that has a method that checks that a proposed password is valid. A password is valid if it has at least *ten* characters of which *two* need to numbers, *two* upper case letters and *two* special characters (like @$€¥ and so on). The program should continue to run until a correct password is given. See below for an example run:

```
Enter a password: nisse4president
Enter a password: N1ss34Pr#sident
Enter a password: N1ss34Pr#sid#nt
Password is valid!
```

Arrays and ArrayLists

The main objective of the following tasks is to understand and use arrays. Most of the tasks will use the primitive array type, but towards the end, there will also be `ArrayLists` as described in chapter 11.11 in the latest version of the book.

2.14 All Odd

Create a program called `AllOdd.java` where an array of size 10 is created and filled with random numbers between 1 and 100. The array should be printed in whole by the program and after that all the odd numbers should be printed on a separate line. See below for an example:

```
Complete array:  23 87 68 59 46 62 44 46 14 98
All odd numbers: 23 87 59
```

2.15 Swedish Lotto

The Swedish Lotto works by selecting seven of 35 numbers (in sequence). Write a program called `Lotto.java` which creates a valid lotto sequence, that is seven numbers in order from 1 to 35 without duplicates. You may use either a normal array or an `ArrayList` for this task. See below for an example execution:

```
The Lotto numbers this week:
5 6 18 21 24 31 32
```

2.16 The Ant

Imagine a chess board and an ant. The ant is randomly put on the board and after that it can walk up, down, left and right (not diagonally). The ant cannot walk over the edge of the chess board (if it tries, it is not counted as a movement). The task is to create a program called `Ants.java` that simulates the walking over the chess board of an ant. To walk to another square than the one the ant is currently on is called a "step" (even though it will take the ant several steps to move...). Each simulation should calculate the number of "steps" the ant takes to visit all squares on the chess board. The simulations must be done ten times and an average should be calculated at the end of the simulation. An example run of the simulation is shown below:

```
Ants
=====

Number of steps in simulation 1: 708
Number of steps in simulation 2: 818
Number of steps in simulation 3: 953
Number of steps in simulation 4: 523
Number of steps in simulation 5: 671
Number of steps in simulation 6: 338
Number of steps in simulation 7: 535
Number of steps in simulation 8: 702
```

```
Number of steps in simulation 9: 548
Number of steps in simulation 10: 418
Average amount of steps: 621.4
```

2.17 Lego Builder

Exercise 2.17 is marked as VG task → only mandatory for those of you that aspire for a higher mark (A or B).

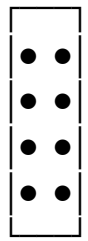
Building with Lego is great fun! In this task you are going to create a program called `Bricks.java` that will print different Lego bricks on screen. The program will ask the user for a size of the lego brick in how many studs are to be used for width and height. If the width is negative, the program should end nicely. The "building" of the Lego brick will be done in a method which returns a *two dimensional array of strings* which contains the brick itself. We need to use `String` even though we will only store one character in each position as the primitive `char` is to limiting. In the main program the brick will be printed by going through the array.

To make things more interesting, suitable *unicode* characters should be used to display the brick itself. A table over the different characters available to use, depending on what the typeface used in the terminal can handle, can be seen on `https://unicode-table.com/en/`. It doesn't have to be perfect, but it should be better looking than using only the characters available on the keyboard.

On the next page there is an example of the program running, where you can see what characters have been used to build the bricks.

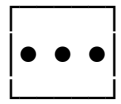
```
Lego Builder

Width: 2
Height: 4
```



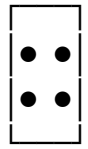
```
Lego Builder

Width: 3
Height: 1
```



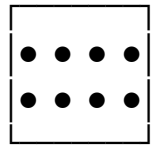
```
Lego Builder

Width: 2
Height: 2
```



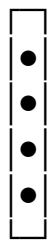
```
Lego Builder

Width: 4
Height: 2
```



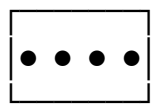
```
Lego Builder

Width: 1
Height: 4
```



```
Lego Builder

Width: 4
Height: 1
```



```
Lego Builder

Width: -2
Bye!
```