© Manfred Kerber
Alexandros Evangelidis

School of Computer Science
University of Birmingham

Worksheet 5

MSc/ICY Software Workshop

Assessed Exercise: 2% of the module mark.

Submission: Thursday 30 November 2017 2pm

5% late submission penalty within the first 24 hours. No submission after 24 hours.

JavaDoc comments are mandatory. Follow the guidelines in `submission.pdf`

This worksheet will be vivaed. No public tests will be provided.

Exercise 1: (Basic, 25%) In this exercise you have write four separate classes. You need in addition the `Customer` class.

- Using the `Customer` class from the lab lecture in Week 7 (see the corresponding Canvas page) write a sub-class `CustomerWithGoods` which has the additional field variable `ArrayList<Good> goods`, which stores the goods a customer has ever ordered. Your class should have a getter, a `public void buy(Good good)`, which adds a good to the ArrayList, a `public int valueOfGoods()` method that computes the total value of all goods the customer has ever bought, and a `toString()` method.

- An object of type `Good` is characterized by the two field variables `String name` and `int price`. Write a corresponding minimal class with a getter for the `price` and a `public String toString()` method.

- Furthermore, write also a class `CustomerBase` with the single field variable `private ArrayList<CustomerWithGoods> allCustomers`, an ArrayList which contains all customers of a company together with the goods they ever ordered.

  In addition to a constructor, the class should contain three methods, one to add a customer, `public void addCustomer(CustomerWithGoods customer)`, and the two methods `public ArrayList<CustomerWithGoods> filterLoyal()` and `public ArrayList<CustomerWithGoods> filterValued()`. The method `filterLoyal` is to collect from the customer base of all customers those customers who have an above average number of orders and `filterValued` all those customers for whom the value of all their orders combined is above average.

- Finally, write a class `CustomerBaseMain` with a `main` method, in which you define suitable objects so that you can demonstrate the other classes in your viva.

Exercise 2: (Basic, 25%) Eclipse allows you to generate a constructor, the getters, and the setters of a class automatically. Assume that a field variable is represented in a class `Var` with the two fields `private String typeOfVar` and `private String nameOfVar`. Write a corresponding class and a class `BuildClass` with the two fields `private String className` and `private ArrayList<Var> fields`. Write in the class `BuildClass` the following methods:

- `public String makeFields()` which generates a String corresponding to the field variable declarations at the start of the corresponding class file.
- `public String makeConstructor()` which generates a String corresponding to a standard constructor (as displayed, e.g., by Eclipse).
- `public String makeGetters()` which generates a String corresponding to all getters.
- `public String makeSetters()` which generates a String corresponding to all setters.
- `public String buildClass()` which combines all the methods, writes in addition the header of the class (`"public class ClassName{"`) and also adds the final closing `"}"`.

Exercise 3: (Medium, 20%) Eratosthenes of Cyrene invented more than 2000 years ago an efficient method to determine all prime numbers up to a maximum `n`. In modern terminology, you first write the numbers between `2` and `n` in a list, then the left most number is a prime number and you delete all its multiples from the list, the next left most number is again a prime number and you delete all its multiples from the list. You continue until you have reached the end of the list. Actually you can stop when the next prime number in the list is bigger than the square root of `n`.
Write a method `public static int[] sieve(int max)` which returns the array of all prime numbers between `1` and `n` (`n` included if it is a prime number) in increasing order. Use an array `sieve` of `boolean[]` to represent the sieve. Initially the values of `sieve[0]` and `sieve[1]` are set to false (since `0` and `1` are no prime numbers), and those for `sieve[2]` through `sieve[n]` are set to true (since all these are candidates to be primes). Rather than deleting elements from the list your method should iterate through the left most elements in the array which have a value of `true` and set the values of its multiples in the array to `false`. As a result, in the end the values in `sieve` are so that the value of `sieve[i]` is `true` if `i` is a prime number and `false` otherwise (for all `i` from `0` through `n`). For instance for `n` as 13:

```
    0     1     2     3     4     5     6     7     8     9     10    11    12    13
   [false false true  true  true  true  true  true  true  true  true  true  true  true]
2->[false false true  true  false true  false true  false true  false true  false true]
3->[false false true  true  false true  false true  false false false true  false true]
```

The next to process would be `5`, but is does not need to be processed since `5 > sqrt(13)`. Finally, copy all those indices into an array of type `int[]` of appropriate size for which `sieve[i]` is `true` and return this array.

Exercise 4: (Advanced, 15%) In this exercise you should write a class `Chess.java` with a single field variable `chessBoard` of appropriate type which represents a chess board. Write a `public String toString()` method which represents the initial board in the form displayed below (and any other board accordingly). Furthermore implement an interactive way to play a game of chess (following the example from week 4 for an interactive program for a bank account, see `BankAccountInteractive.java`). It should be possible to enter moves of the type `a2a4` from the command line in order to move the figure from a2 to a4 on the chessboard. Initially the chess board is to be displayed (as shown below) and after each move as well. If the input does not follow the pattern `"[a-h][1-8][a-h][1-8]|q"` an exception is to be thrown that has then to be caught appropriately. On entering `q` the program should terminate.
Note, you are NOT expected to check whether a move is legal according to the chess rule book. The task is only to display the board after each move entered according to the pattern provided above.

```
      a        b        c        d        e        f        g        h
  +--------+--------+--------+--------+--------+--------+--------+--------+
8 |rook    | knight | bishop | queen  | king   | bishop | knight | rook   |  8
  +--------+--------+--------+--------+--------+--------+--------+--------+
7 |pawn    | pawn   | pawn   | pawn   | pawn   | pawn   | pawn   | pawn   |  7
  +--------+--------+--------+--------+--------+--------+--------+--------+
6 |        |        |        |        |        |        |        |        |  6
  +--------+--------+--------+--------+--------+--------+--------+--------+
5 |        |        |        |        |        |        |        |        |  5
  +--------+--------+--------+--------+--------+--------+--------+--------+
4 |        |        |        |        |        |        |        |        |  4
  +--------+--------+--------+--------+--------+--------+--------+--------+
3 |        |        |        |        |        |        |        |        |  3
  +--------+--------+--------+--------+--------+--------+--------+--------+
2 |PAWN    | PAWN   | PAWN   | PAWN   | PAWN   | PAWN   | PAWN   | PAWN   |  2
  +--------+--------+--------+--------+--------+--------+--------+--------+
1 |ROOK    | KNIGHT | BISHOP | QUEEN  | KING   | BISHOP | KNIGHT | ROOK   |  1
  +--------+--------+--------+--------+--------+--------+--------+--------+
     a        b        c        d        e        f        g        h
```

Exercise 5: (Advanced, 15%) "RSA (Rivest-Shamir-Adleman) is one of the first public-key cryptosystems" [`https://en.wikipedia.org/wiki/RSA_(cryptosystem)`]. Why the RSA method works requires some mathematical understanding. This understanding is, however, not required in order to implement it.

The task here is to build an implementation as described on the Wikipedia page. Note that you can select prime numbers `p` and `q` of type `long` using the method provided by the Sieve of Eratosthenes. This would typically give you primes up to a size `Integer.MAX_VALUE`, that is, `2147483647`. Note that prime numbers used in a real encryption contain at the time of writing this worksheet hundreds of digits. In order to use them in Java you would need a `BigInteger` implementation, which is essentially the same but is more complicated to write down.

The RSA method works by first randomly selecting two different big prime numbers `p` and `q`. Their product is called `n`. In order to work in the type `long` this means that the prime numbers `p` and `q` must be small enough that their product is smaller than `Long.MAX_VALUE`, that is, `9223372036854775807`. The RSA method relies heavily on the fact that it is easy to compute `n` as `p*q`, but that no efficient method is known to compute `p` and `q` from `n` (for big `p` and `q`).

Furthermore you determine `lambda` as the least common multiple (`lcm`, see Lecture notes of week 3) of `p-1` and `q-1` and a third prime number `e` so that `e` does not divide `lambda`. The last number playing a role is a number `d` which is the `inverse` of `e` with respect to `lambda`. Pseudocode to compute the inverse can be found on `https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm`.

A sender takes from the recipient the numbers `n` and `e`. In order to encrypt now a number `plainNumber` the sender takes it to the power `e` modulo `n`. The so calculated number `encryptedNumber` can then be sent in plain to the recipient. Only the recipient would know the secret decryption number `d`. Decrypting is done by taking the `encryptedNumber` to the power `d` modulo `n`.

Remember when you write your class first decide on the field variables, then write a suitable constructor. There should not be any setters in your class.

Note that for the method you should write a method `public static long power(long x, long a, long modulus)`. Naively this can be done by multiplying `x` by itself `a` times, each time taking the modulus. This is inefficient for big numbers `a`. There are, however, efficient methods. You should try to find one and implement it.

Use your implementation to encrypt (and then decrypt the result) for the number `65` with the following triples for `p`, `q`, and `e`:

p	q	e
61	53	17
293	317	97
99991	8999	14983