Practical Programming Exercises

Summer semester 2012

translation by Asen Asenov

Harald Schilly & Andreas Ulovec
harald.schilly@univie.ac.at & andreas.ulovec@univie.ac.at
`http://harald.schil.ly/teaching/ue-pp-12`

Introduction. The following examples are supposed to be treated using the Java programming language. Work on your own or at most with one partner per exercise. It is very important to get an in-depth understanding of the individual examples and therefore recommended to work independently.
In general, attendance is mandatory – except for those who finish their examples before the end of the course. Your examples will be evaluated and graded by the coaches and tutors. The number of stars in the rectangle is an indication of the degree of difficulty and the maximum number of points you can get by successfully solving the exercise.
The goal of this course is to become familiar with development environments, learn how to implement procedures, get to know data structures, understand how information is processed and stored and finally how to document your programming code to make accessible to others.
For more information, advices and literature we refer to the end of this document or to the website.
The first part of the exercises has to be sent by e-mail to the instructor until 17th/20th April. Your e-mail should primarily contain all necessary `*.java` files (label them by writing your name and student ID no. as a comment), in case you collaborated with somebody, list his or her name as well.
The second part is more evolved (as is indicated by the stars), the examples deal extensively with object oriented programming and standard algorithms, thus additional subjects such as code documentation, performance testing and testing of methods and algorithms become relevant. Overall you can achieve a total score of 42.
Additionally or alternatively to the examples in the first and second part, you can solve some examples from the end of exercise sheet, marked with $\oplus$, instead. These are all worth 6 points and are usually more sophisticated then the regular ones. The final mark for the course is based on the sum of all points of correctly solved examples, regardless of which part they belong to.
The second part of the examples should be send in until the 19th or 22th of June. On the 26th or 29th of June, the final interview and rating will take place.
Cancellations will be accepted no later than March 31st, see §8(2) the "Statute of the University of Vienna"[1] – all registered students will be graded without exception.

[1] `http://www.univie.ac.at/satzung/studienrecht.html`

Part 1

1. (Control Structures I, $\boxed{\star}$) Define variables for centuries, years, months, days, hours and minutes and initialize them with some values. The program has to calculate the total number of minutes (`long`-Type) for this period of time and output it. For simplicity we assume that one month has always 30 days and a year always has 360 days.
   Example: 2 centuries, 1 year, 3 months, 20 days, 12 hours and 2 minutes = 104 357 522

2. (Control Structures II, $\boxed{\star}$) Invert the previous program, i.e. given the total number of minutes, compute the original values of centuries, years, months, days, hours, and minutes, and output them. Test whether the values match when the preceding and this program are linked.
   Example: 602 = 10 hours and 2 minutes

3. (Control Structures III, $\boxed{\star}$) Store in the variables x and y a positive number as a lower and upper limit and in another variable s is the positive step size, such that $0 < x < y$ and $s > 0$. All variables are of type `double`. Calculate the product of **all** numbers ranging from x to y with step size s using this formula:

$$\prod_i r_i \text{ where } r_i = x + i \cdot s,\ r_i \leq y,\ i \in \{0, 1, 2, \dots\}.$$

4. (Control Structures IV, $\boxed{\star}$) Implement an algorithm for solving a quadratic equation $ax^2 + bx + c = 0$ via the function `static void solve(double a, double b, double c)` and display the result.
   It is sufficient to treat only real solutions, but do **not** forget any special cases (e.g. return “complex solution” or “empty set” if such a solution arises), `NaN` as a “solution” is not allowed.
   Show that your algorithm works by letting it calculate both an ordinary double solution and some special case.

5. (Control Structures V, $\boxed{\star}$) Add up in a loop randomly generated floating-point numbers between -5 to $+5$, until the absolute value of the sum exceeds a fixed constant K (e.g. 42). Repeat this 10 times and print each time the number of iterations that were necessary.
   Note: `Math.random()` returns uniformly distributed random numbers belonging to the interval $[0, 1)$.
   Question: Could this algorithm cause problems?

6. (Control Structures VI, $\boxed{\star\star}$) Model a virtual dice (with faces from 1 to 6) by “casting” random numbers from `Math.random()` to `int`-type. Roll the dice 9999-times, rolling three dice each round. Determine the probability that...
   a) ... all three dice show, the same number at the same time?
   b) ... only two of the three dice show the same number?
   c) ... all three dice show numbers ≤ 3?
   d) ... for two consecutive throws, the number 6 is seen exactly once?

   Print the estimated values.
   Note: We follow the naive approach to probability theory and understand the probability as the quotient of “desirable events” / “all events” considering all simulated throws.

7. (Code Analysis, $\boxed{\star\star\star}$) A computer program, as it is required in all the other exercises, is a sequence of instructions, that can be read by a computer, written to perform a specified task, usually described in linguistic terms (e.g. English or German). In this example, we do the opposite! Explain what the following two programs do and try to understand their ”purpose”. Write a text for an exercise that would imply these programs as solutions.

   Program 1:

```java
boolean ex1(int x) {
  if (x > 2 && x % 2 == 0) return false;
  for(int i = 3; i <= Math.sqrt(x); i+=2) {
    if(x % i == 0) return false;
  }
  return true;
}
```

   Program 2:

```java
long auxiliary_function(int x) {
  long p = 1;
  for (int i = 1; i <= x; i++) {
    p *= i;
  }
  return p;
}

double ex2() {
  double ret = 1;
  for (int i = 1; i < 50; i++) {
    ret += 1. / auxiliary_function(i);
  }
  return ret;
}
```

8. (Strings, $\boxed{\star}$) In this example we deal with the peculiarity of character strings (strings). Try first theoretically, with pen an paper, what result is expected. Then run the program and explain the behaviour of the result!

```java
public static void main(String... args) {
  String a1 = "foo";
  String a2 = a1;
  testStrings("a1, a2", a1, a2);
  String a3 = "foo";
  testStrings("a1, a3", a1, a3);
  String a4 = new String("fo") + "o";
  testStrings("a1, a4", a1, a4);
  String a5 = a4.intern();
  testStrings("a1 a5", a1, a5);
}

static void testStrings(String info, String a, String b) {
    System.out.println(info);
    System.out.printf("<%s>.equals(<%s>) -> %s\n", a, b, a.equals(b));
    System.out.printf("<%s>   ==   <%s>  -> %s\n\n", a, b, a == b);
}
```

9. (API, $\boxed{\star}$) Get to know the Java platform documentation and in particular the Java API (see bibliography). Where can it be found, how is it structured, what kind of documentation is available? Answer the following questions:

   a) What is a “Java HotSpot compiler” and what does it?

b) What peculiarity has the java.langjava.lang package in contrast to all other packages of the API?

c) Later on we learn that Java is an object-oriented programming language, and almost everything is derived from a very general class “Object”. Find this in the API. Write down the URL and all methods of that class.

d) Find the class `java.lang.String` and explain what the methods `length()`, `substring()` and `trim()` do.

e) Find the `ArrayList` class and explain, based on its description, what it does and give a possible application.

10. (Command Line, $\boxed{\star}$) In this example, we read out command line parameters passed over to the program. They are available in the variable `args`, from the main function - `public static void main(String[] args)`. In particular, the values must not be included directly in the program, they are to be assigned during the execution of the program! (Eclipse: Run ⇒ Run Configurations ⇒ Arguments; Netbeans: Run ⇒ Set Project Configuration ⇒ Customize: Run ⇒ Argument).
The task is as follows, for each parameter passed (i.e. `String`), calculate its length and display it. In addition, the average length should be calculated.

    Example:

    ```
    $ java CommandLine Jam is sweet
    Jam: 3
    is: 2
    sweet: 5
    Avarage: 3.3
    ```

11. (ASCII II, $\boxed{\star\star\star}$) The following program will draw a mathematical function in the console. Define a static function $f(x)$ with the signature `static double f(double x)`, that for example returns `return x*Math.sin(x)/2;`. Furthermore, there are variables a and b for the left and right limits of the representation. Scale the difference $b-a$ to a reasonable width of 80–100 characters and do so for the vertical direction, approx. 40–50 characters, as well ($\max(f(x)) - \min(f(x))\ \forall x \in [a, b]$). The function itself should be represented by a “chain” of connected characters $*$. Example:

    ```
    ***
       **              *****
         *          ***
         *       **
         *      *
          *    *
           ***
    ```

    It is advisable to set limits as well as a minimum and maximum value for the vertical values.

Part 2

12. (Recursion I, Benchmark, $\boxed{\star\star}$) Calculate the values of a recursion. It should be possible to start with arbitrary initial values. The recursive formula is:

$$f(n) = f(n-1) + 101 * f(n-2) + 503 * f(n-3) + n^2 \pmod{997}$$

with, for example, initial values $f(0) = f(1) = 0$ and $f(2) = 1$.
Implement this recursion *iteratively* as well as *recursively*.
Make a comparison between the recursive and the iterative approach in form of a benchmark (in order to obtain meaningful numbers, repeat the same calculation often enough, see glossary “benchmark”).
Compute the first 30 numbers and analyse how the series behaves!

13. (Recursion II, Benchmark, $\boxed{\star\star}$) Improve the recursive calculation from the previous example in such a way, that the result of $f(i)$ is stored in an array, at position i. When computing $f(i)$, reuse the value if it is already known, otherwise perform its calculation and store the result in the array. By doing so no unnecessary recursions will be made. Once again compute a meaningful benchmark (see glossary).

14. (Classes, Vector algebra I.1, $\boxed{\star\star\star}$) Create a class `Vector2D` that allows to perform mathematical operations with two-dimensional vectors. A vector is defined as a pair of floating point numbers to represent the first and second direction in the plane. The respective directions or coordinates are to be *private* fields of type `double`.

    The class is supposed to have several constructors:

    - `public Vector2D(double r, double i)` – Main constructor
    - `public Vector2D()` – Vector with the value (0, 0)
    - `public Vector2D(Vector2D c)` – Generates an **independent**, new Vector2D-Object from an existing Vector2D Object (“*Copy-Constructor*”).

    In order to perform calculations with these vectors, methods, that take another object of this class as input and create a new object, namely the result, as output, need to be implemented.

    The following methods have to be implemented:

    `Vector2D`:

    - `public Vector2D add(Vector2D other)` – Addition of the vector `other` to the corresponding object (`this`).
    - `public Vector2D sub(Vector2D other)` – Subtraction.
    - `public double scalar(Vector2D other)` – Scalar product of two vectors.
    - `public Vector2D mult(double scale)` – Scale by factor `scale`.
    - `public double winkel(Vector2D other)` – Angle between the two vectors, in radian.
    - `public double length()` – returns the length of the vector (use `Math.hypot()`).
    - `public boolean parallel(Vector2D other)` – returns `true`, if the two vectors are parallel, otherwise `false`.

- `public String toString()` – generates a human-readable output (for example "(2, -3)"). In, for example `System.out.println(vector)`, this method is automatically called if objects of that class are turned over.

Perform the following calculations and display the results:

- What is the angle between the vectors $\overrightarrow{1, 1}$ and $\overrightarrow{2, 2.0001}$?
- Are the vectors $\overrightarrow{4.4, -1.1}$ and $\overrightarrow{-44.44, 11.11}$ parallel?

15. (Classes, Vector algebra I.2, $\boxed{\star\star}$) Furthermore, we now need as well a class `Point2D`, which represent a point in the plane. For the constructors, proceed quite analogously like you did `Vector2D`.

    `Point2D`:

    - `public Vector2D diff(Point2D other)` – Difference between two points: `Vector2D`.
    - `public double distance(Point2D other)` – Distance between two points. Use some suitable method from `Vector2D` but do not duplicate any code!
    - `public int quadrant()` – Check in which quadrant the point lies.
    - `public static double area(Point2D p1, Point2D p2, Point2D p3)`
      This is supposed to be a *static* (!) method, that calculates the area of the triangle, spanned by the three given points.

    Perform the following calculations and display the results:

    - The distance between the points $(-4.4, 42)$ and $(11, 3.1415)$.
    - Area of the triangle $\Delta\{(-1, -1), (0, 5), (3, -1)\}$.

16. (Vector algebra II, Tests, $\boxed{\star\star}$) Write simple tests for the classes `Vector2D` and `Point2D` from the previous examples. To this end design another class `VectorOperationsTest`, which first instantiates some fixed points and vectors, then call all methods for each class and explicitly verify that, by explicitly comparing the results with a known outcome, that each result is correct. *All* methods have to be tested individually!
    Hint: Use the keyword `assert` and extend the class `Vector2D` and `Point2D` by a method `public boolean equals(Vector2D|Point2D other)`, which tests for equivalence.

17. (Javadoc, Comments, Code Conventions, $\boxed{\star}$) Read the necessary chapters of the documentation "javadoc". Format the code of the classes `Vector2D` and `Point2D` according to the "Java Code Conventions" (see bibliography).

    In particular, correct line breaks, indentation, and mind correct capitalization. Learn to identify the different elements by their writing! Write appropriate code descriptions for the classes, *all* methods (and parameters) and fields. Each class should carry the author's name in the header, and include cross-references (keyword "`@link`") within the text description of the methods in `Point2D.distance(Point2D other)` and `Vector2D.distance(Vector2D other)`.
    Finally generate the "javadoc"-documentation as a collection of HTML files using the program *javadoc* or the IDE.

18. (Classes, Matrix I, $\boxed{\star\star\star}$) Write a class `Vector2DMatrix`, which implements the main matrix operations. The class should allow general $m \times n$ matrices with `Vector2D` entries as elements. Document the class and all methods with a suitable "javadoc" and write a sample program that calls each method and returns the output to the console.
    `Vector2DMatrix` should include the following methods:

```java
// Constructor for a n x n Matrix
public Vector2DMatrix(int n)
// Constructor for a rows x cols Matrix
public Vector2DMatrix(int rows, int cols)
// Random matrix, n x n Matrix with random entries
public static Vector2DMatrix random(int n)
// Size of the Matrix, [rows, cols]
public int[] size()
// give back element at position (r,c)
public Vector2D get(int r, int c)
// set element (r, c) to the value of v
public void set(int r, int c, Vector2D v)
// add up a Matrix a
public Vector2DMatrix add(Vector2DMatrix a)
// calculate the vector sum of a row
public Vector2D rowsum(int r)
// calculate the vector sum of a column
public Vector2D colsum(int c)
// matrix multiply with a scalar a
public Vector2DMatrix mult(double a)
// Representation
public String toString()
```

Hints:

- The `toString()` method should make use of the `toString()` method of the `Vector2D`-class and display the matrix in tabular form by using e.g. `String.format(...)`.
- To create a matrix with random entries, a `public static Vector2D random()` method in the `Vector2D` class.

Note: If, and only then, the exercises from the above vector classes were omitted, then you can program this example using the type `double`.

19. (Collections, $\boxed{\star\star}$) Often not only an object, but a "collection" of similar objects (instances of the same Class, or with the same Interface) is required, which in turn are stored in a "collector"-object. This is a further development of Arrays (`<Class>[]`) and may be useful in different situations. Create a short program, which accomplishes the following:

    - Save 20 randomly chosen integers from −100 to 100 in
        - a `int[]`–Array.
        - in a `ArrayList<Integer>`.
          (The angle brackets contain the Class name and specify the type of objects held by this `ArrayList`; keyword "*Generics*").
        - in a sorted Set, namley the Class `TreeSet<Integer>`.
    - Then do the following:
        - Display all "collected" objects using `System.out.println()`. Hint: `Arrays.toString(varia`
          is helpful to display the Array correctly.
        - Calculate the sum, average number, minimum and maximum number, separately for each "collection".
        - Check, whether a randomly generated number appears in the respective " collection". Repeat the test until it is successful, "true", and return the location of the generated number in the respective data structure.

Find more information here[2] and here in the API[3].
Note: Since version 1.5 of Java `int` and `Integer` are equivalent - this is called "outboxing."

20. (Collections II, $\boxed{\star}$) In a coffee dispenser, the change has to be calculated. A coffee costs k cents (for example 55) and an amount of s cents is provided in a certain denomination. The denomination is an associative list (`Map<Integer, Integer>`) of `Integer` values of denomination size giving their number (of their appearance) as an `Integer` value.
For example: { $s[1] : 0$, $s[5] : 3$, $s[10] : 0$, $s[50] : 0$, $s[100] : 2$, $s[200] : 0$ },
meaning there is no 1-Cent coin, 3 times a 5-Cent coin, no 10- or 50-Cent coins, 2 times a 1-Euro coin, and no 2-Euro coins. Note, the algorithm should work with any denomination, this is just an example!

    Issue: Return an associative Map w with money for the change, i.e. $k = s - w$, where as few coins as possible are returned!

21. (Statistics I, $\boxed{\star\star\star}$) Due to the developments in recent years, the financial industry has become an interesting place for Mathematician. This example should do the following:
    a) Download historical(daily) data of stocks.
       For example: Apple's share from Google Finance in CSV text format[4]. (this is the "Download to Spreadsheet"-link where historical information about the shares is given).
       Equivalently you can find Google's shares at Yahoo! finance[5] and under "historical Prices" a "Download to Spreadsheet" option. Find daily data of Google's shares here[6].
    b) Parse the CSV data and store it in an appropriate class. It is useful to program multiple vectors of equal length to for the respective columns. Do not forget to save the date.
    c) Calculate the following characteristic figures:
       i. Minimum and maximum of the 'Close'-value, which days was it?
       ii. Average value and median of the 'Close'-value.
       iii. Calculate the date of the top 3 days, when the price has risen or fallen the most, i.e. consider the difference between "Open" to "Close".
       iv. Calculate a monthly summary of daily data in the following way: For each month calculate the weighted (by "Volume") average [7] of the average of the "Low" and "High" values. Return the list of these figures stating the corresponding months and year.

    Hints: To import the CSV file, it is useful to use the class `java.util.Scanner`, and the class `java.util.HashMap` to perform the statistics.

    For the download, start with the following code:

```
URL url = new URL("http://...");
InputStreamReader in = new InputStreamReader(url.openStream());
```

[2] http://download.oracle.com/javase/tutorial/collections/TOC.html
[3] http://download.oracle.com/javase/6/docs/technotes/guides/collections/index.html
[4] http://www.google.com/finance/historical?q=NASDAQ:AAPL&output=csv
[5] http://finance.yahoo.com/
[6] http://ichart.finance.yahoo.com/table.csv?s=GOOG&a=07&b=19&c=2004&d=01&e=20&f=2012&g=d&ignore=.csv
[7] Average with weight vector w_i: $\bar{x}|_w = \left(\sum_{i=1}^n w_i \cdot x_i\right) \Big/ \left(\sum_{i=1}^n w_i\right)$

```
BufferedReader data = new BufferedReader(in);
for(String line; (line = data.readLine()) != null;) {
   System.out.println(line);
}
```

Alternatively, download the CSV file manually and read it with a `FileReader`

The date field is best be stored and read using a “`Calendar`” instance:

```
// 1. Global custom date format, as it exists in the CSV file:
final DateFormat df = new SimpleDateFormat("yyyy-MM-dd");
...
String date = "2011-01-11";
// 2. Instance of the "Calendars", date set.
final Calendar pdate = Calendar.getInstance();
pdate.setTime(df.parse(date));
// 3. Specific fields reading. Note: January has the value "0"!
System.out.println("day: " + pdate.get(Calendar.DAY_OF_MONTH)); //   11
System.out.println("month: " + pdate.get(Calendar.MONTH));      //    0
System.out.println("year: " + pdate.get(Calendar.YEAR));        // 2011
```

22. (Sorting I, $\boxed{\star\star}$ + $\boxed{\star}$) Create a class that sorts words from a text file, using an arbitrary algorithm. The “Bubble-Sort”-method for example works in such a way, that adjacent items in the list are continuously swapped until no further swaps are necessary.

    The words should be read using a `FileReader` or `Scanner` and then displayed in a new text file. Note that only words should be read, no punctuation; also convert all characters to lower-case.

    - The algorithm must be implemented by hand. The use of standard routines such as the `sort()`-method of the class `java.util.Arrays` is allowed only purposes of comparison.
    - Reading and sorting should be encapsulated in separate methods. The methods should include appropriate parameters and return values.
    - Measure the average, minimum and maximum speed of the program when sorting different-sized amounts of words (benchmark, see Glossary). Compare with others and make assumptions about the complexity, optionally implement another procedure!
    - One extra point $\boxed{\star}$ will be given, if the sorting method is programmed generally for objects, implementing the `Comparable` interface. The idea is that the sorting routine operate on this list: `List<Comparable>` and uses the method `compareTo(...)`.
      Test the now gained flexibility by using the same code, but instead of a list of `Strings`, sort list of random `Integer`-objects.

23. (Statistics II, ASCII III, $\boxed{\star\star}$) Display the data from “Statistics I”, as a histogram of the most common stock values.

    Depict the histogram as a bar chart in the command line and display also the number of values corresponding to each interval as a right-justified integer (use `System.out.printf`). Pay attention to a meaningful horizontal scale.
    Example: The minimum price was “10.0” and the maximum “22.0”. The histogram divides this area into for example 9 equal-sized “bins” and counts how often the stock values lied within these intervals.

```
10.0   1 *
11.5   2 *
```

```
13.0   8 ***
14.5  11 *****
16.0   9 ****
17.5  23 ***********
19.0   5 ***
20.5   6 ***
22.0   1 *
```

Note: A “reasonable” number of “Bins” is the rounded up square root of the number of records.

Further examples

The following examples are intended for deeper examination of the material or as an additional examples.

1. (Product Management, ⊕) The purpose of this example is to write an administration system, for the products in a store. Basically, it should first be able to manage a list of products, then it should be possible to make "Purchases". In list is issued the total price + VAT invoice.

   All prize values should be recorded as `int` amount of cents, which prevents happening of rounding errors, if we use `double` numbers.

   Create the following Classes:

   - `IWithPrice` – A "Interface" that all Classes will implements, which specify a price operations:
     - `int getNetPrice()`
     - `int getVatInEuro()`
     - `int getPriceWithVat()`
     - `int getVat()`
   - Class `Product implements IWithPrice`:
     - Constructor `Product(String name, int productId, int stockprice, boolean food)`
       The field `productId` should be clear, the flag `food` is used to calculate the food tax, which is 10% for food and 20% for everything else.
     - `String getName()`
     - `int getProductId()`
   - Class `Purchase implements IWithPrice`:
     - Constructor `Purchase(Date date)`
     - The class includes private fields for the list of products as `ArrayList<Product>` and the date of purchase as `Date` object.
     - `addProduct(Product item)` – adds the product.
     - `List<Product> getProducts()` – the list of all products.
   - Class `ProductCatalog` – This contains a list of all products in the store. Store all types of products in a `HashMap<Integer, Product>` object, and associate "`productId`" with each product.

     The method `Product getProduct(int productId)` returns the item, with a given number.
   - Class `ReceiptPrinter` – this class construct the receipt from shopping of an `Purchase`-object. All numbers should be formatted and positioned correctly (`System.out.-printf`).
     Example:

```
++++++++++ RECEIPT ++++++++++
Your purchase of
Tue Feb 21 12:00:34 CET 2012
```

```
------------------------------
Article        Price/PCs  VAT
------------------------------
10001 Apple      0.34     10%
10002 Pear       0.34     10%
10002 Pear       0.34     10%
20001 Vodka      9.99     20%
==============================
Total amount: 11.01
         VAT:  2.10
Thank you for your purchase!
```

- Below is one example of the Main Class with `main()` method.
  Sketch:

```java
ProductCatalog catalog = new ProductCatalog();
catalog.addProduct(new Product("..." ... );
...
Purchase purchase = new Purchase(new Date());
purchase.addProduct(catalog.getProduct(...));
...
ReceiptPrinter printer = new ReceiptPrinter();
printer.printer(purchase);
```

2. (Statistics IV, ⊕) The aim of this exercise is to implement formula, from the literature of computer science, in a (relatively simple) program. It is about calculating the entropy of a given text, according to Shannon. This is in *"Prediction and Entropy of Printed English"*, (1950) on page 51 the formula (1) states:

   The entropy is

   $$F_N = -\sum_{i,j} \Pr(b_i, j) \log_2(\Pr(j|b_i))$$

   where b_i are programs for all n, (b_i, j) means that the letter j is appended to the n-grams b_i and $\Pr(j|b_i)$ is the conditional probability that a j to b_i follows – this is $\Pr(b_i, j)/\Pr(b_i)$.

   Are there differences between German, English, French, ... ?

   Tip: Be careful when parsing the text, to read only words, but no other signs! Then generate a list of b_i of *all* n-grams (the letters are arrays of length n) - also known later is that a good value for n is 3, 4 and 5 (see Formula(2)). Then iterate over all n-programs, browse the texts and determine the probabilities for the two formula.

   Literature: `http://languagelog.ldc.upenn.edu/myl/Shannon1950.pdf`

3. (Big Numbers, ⊕) Write a class `BigNumber` that permit calculations with arbitrarily large integers in $\mathbb{Z}$. Internally use a `ArrayList` of type `Byte` to store the single digits from 0 through 9.
   Constructors:

   - `BigNumber()` – the value 0
   - `BigNumber(String)` – Parse the string, consisting of the character "-" and "0" to "9".
   - `BigNumber(BigNumber number)` – Copy-Constructor
   - `BigNumber(Integer)` – produce from conventional integer.

   The following calculations should be possible and the correct operation to be underpinned with appropriate tests.

- `String toString()` – Representation as a string
- `BigNumber add(BigNumber other)` – Addition
- `BigNumber add(int other)` – Addition with usual `int`
- `BigNumber mult(BigNumber other)` – Multiplication
- `BigNumber mult(int other)` – Multiplication with usual `int`
- `BigNumber sub(BigNumber other)` – Subtraction
- `Double approx()` – Approximation of the BigNumber as a floating point number of type `Double`

Test to seek treatment of numbers with more than 1000 digits!
Note: It is of course not allowed to use `BigInteger` or `BigDecimal`!

4. (Geometry, ⊕) In the previous examples, two classes for two-dimensional points and vectors were introduced. Extend this very simple system, for the calculation of geometric objects, with the following classes:

   - `Line2D` – This is a class that contains a Point and a Vector.
   - `Circle2D` – This is a class consisting of a Point and `Double` value as the radius.
   - `Triangle2D` – This class consists of 3 Points.

   Extend these classes(only if it is necessary and appropriate!), the class `Vector2D` or `Point2D`, so that the following issues can be solved:

   - Intersection of two lines: return either `null` or `Point2D`. Test both cases!
   - Midpoint between two points, `Point2D getMidpoint(Point2D other)`.
   - The straight line, which passes at right angles, to a vector, or straight line passing through a given point; i.e. `Line2D orthogonal(Point2D, Vector2D)` and the class `Line2D` has one method `Vector2D getVector()`.
   - Perimeter of a triangle, i.e. `Triangle2D getPerimeter()`.

   If you are still interested, implement additional methods and routines!

5. (Math Game, ⊕) Program a game for learning basic arithmetic. The game should have at least the following features:

   - Different calculations, which are randomly generated as a questions. Example: $2 + 2 =$, $3 * (21 - 5) + 1 =$, $18/9 + 1 =$, $\ldots$
   - The player should be able to enter the answers and the Program should verify them. When there are fractures or floating point numbers results, be aware to read the input correctly and consider the floating point rounding errors.
   - Furthermore, program should be able to manage different users. For this read the username, as a parameter, from the command line. See the example “Command Line”.
   - Record the results in a log file <username>.log:
     - a time stamp
     - what Question was asked
     - what response was given
     - how long is it take for answering
     - whether the result was correct or not

- Adapt the level of difficulty, to depend on how well the players respond. A higher degree of difficulty is achieved by longer calculations and larger numbers.

Note: You can read the entries from console, using this:

```
Scanner scanner = new Scanner(System.in);
...
System.out.print("<TEXT>: ");
String inputLine = scanner.nextLine();
```

Tips

- Error message `java.lang.NoClassDefFoundError` when starting from command line: The compiled Java class – or rather, the root directory of the compiled program – is not sought by the class path. The solution is to set the `classpath`, for example:

  `java -cp <path> <ClassName> [Arguments]`

  or in a directory hierarchy for the package root directory:

  `java -cp <RootDir> package/hierarchy/<ClassName> [Parameter]`

  for a class in the package `package.hierarchy.<ClassName>` whose root directory is in `<RootDir>`.

- When is possible program that way, so
  - no code is duplicated,
  - all variables and methods no longer derived the keyword `final`,
  - limited access to all methods, to which we have package access to "default" (i.e no information), all local to `private`, and only a few selected methods and classes on `public` access modifier,
  - generate as few new objects as possible, especially within loops and
  - parameters passed from untrusted sources must be reviewed.

Literature

- Guido Krüger, Thomas Stark, " Handbook of the Java programming", 5Th Edition, a free download at `http://javabuch.de`. The book is a frequently updated and well thought out introduction to the Java language. Emphasize on the chapters: 1, 2.2, 2.3 (2.3.3), 4, 5, 6, 11, 11.4, 13.2, 15 for the basics, 7, 8, 9 for object-oriented programming (OOP), the other 10.4.1 , 17.2 and 21 and also 51.1, 51.2 and 51.5.
- "Oracle Java JDK 5/6 documentation", online at[8]. Complete documentation of the J2SE (Standard Edition) including the API. In case of doubt, this is the best source for everything that affects Java. Despite the Java is somewhat cumbersome language, it is important to find the way in the API.
- "Java Code Conventions", online at[9]. Since the majority of our time goes for processing already-written code, it is important to maintain a consistent style. This facilitates the identification of certain elements such as classes, fields, variables, structures, and this way reduce time, needed to understand foreign codes (see Chapter 7 and 9).

[8]`http://download.oracle.com/javase/6/docs/`

[9]`http://www.oracle.com/technetwork/java/codeconv-138413.html`

- “Documentation Comments with the Javadoc Tool”, online at[10]. Almost as important as a program that work is a documentation of the classes, methods and fields. This information is extracted, using `javadoc` command, in HTML documents (and PDF, etc.) or can be collected, for example, to be displayed in the context of IDEs help. This facilitates the understanding of code and later the others, who read your program.

Glossary

- *Benchmark*: This is a test to determine the performance of a program. For this purpose we measure the time it takes to execute. It is often useful to change the size of the issue, to obtain a statement about the scalability. The time is measured best by differences in the System time: `System.currentTimeMillis()` in milliseconds (corresponding to 1/1000 second!) or `System.nanoTime()` in nanoseconds.

  By the nature of the Java virtual machine, code first to interpret and over the time the parts are translated into efficient machine code, which is performed frequently and is difficult to make benchmarks. Therefore it is advisable to make several repetitions of the instructions, to come to meaningful values. This may also be refined in the way that you calculate in two nested loops, the minimum over an inner loop in the calculated mean.

  Besides the time, the spent working memory can also be of interest.

  Closer monitoring is possible with tools such as `jvisualvm`. See[11] for CPU and memory profiling applications.

- *Recursion*: We call this the technique, when a function calls itself again. Note that there is always an appropriate termination, to avoid infinite recursion depth. In some instances it may be advantageous to store intermediate results in order to avoid repeatedly calculating the same sub-recursion. This is called “ Dynamic Programming”.

- *Test*: A test is an additional part of the program, which checks whether methods or functions work correctly. For example, a function `int = func1(int a)`, that add to the passed variable a the value 10, is called with a particular value (for example 3) and the output is compared with the expected value of 13.

  There is also a Testing frameworks, such as `JUnit`, which are supported by all popular development environments.

[10] `http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html`
[11] `http://download.oracle.com/javase/6/docs/technotes/guides/visualvm/index.html`