3. homework assignment; JAVA, Academic year 2014/2015; FER

First: read end commends. I mean it! You are back? OK. This homework consists of three problems.

As a preparation for this homework, read in book about, and install the following tools: ant, PMD, Junit, and JaCoCo. I have uploaded at Ferko a demo project which uses these tools for build process. Download it, make it work, and then reuse the build.xml I have provided for the project from this homework. The book has updated section (appendix) on installation of ant and other tools mentioned in this homework (http://java.zemris.fer.hr/nastava/opjj/book-2015-03-27.pdf).

Problem 1.

Implement a support for working with complex numbers. Your task is to create a class `ComplexNumber` which represents an unmodifiable complex number. Place the class in the package `hr.fer.zemris.java.tecaj.hw3`. Each method which performs some kind of modification must return a new instance which represents modified number. This class must have:

- public constructor which accepts two arguments: *real* part and *imaginary* part (use `double` for both),
- public static factory methods:
  - `fromReal(double real): ComplexNumber`,
  - `fromImaginary(double imaginary): ComplexNumber`,
  - `fromMagnitudeAndAngle(double magnitude, double angle): ComplexNumber`,
  - `parse(String s): ComplexNumber` (accepts strings such as: "3.51", "-3.17", "-2.71i", "i", "1", "-2.71-3.15i"),
- public instance methods for information retrieval (the function should be clear for method names):
  - `getReal(): double`
  - `getImaginary(): double`
  - `getMagnitude(): double`
  - `getAngle(): double` (angle is in radians, from 0 to 2 Pi)
- public instance methods which allow calculations:
  - `add(ComplexNumber c): ComplexNumber`,
  - `sub(ComplexNumber c): ComplexNumber`,
  - `mul(ComplexNumber c): ComplexNumber`,
  - `div(ComplexNumber c): ComplexNumber`,
  - `power(int n): ComplexNumber`; $n \geq 0$,
  - `root(int n): ComplexNumber[]`; $n > 0$,
- public method for conversion to string:
  - `toString(): String`.

Example of usage:

```
ComplexNumber c1 = new ComplexNumber(2, 3);
ComplexNumber c2 = ComplexNumber.parse("2.5-3i");
ComplexNumber c3 = c1.add(ComplexNumber.fromMagnitudeAndAngle(2, 1.57))
                  .div(c2).power(3).root(2)[1];
System.out.println(c3);
```

Where needed, throw an appropriate exception.

Problem 2.

In the book I have written about some implementation details of class `String` in Java (starting from page 76). After this text was written, the implementation has changed (the internal character array are no longer shared).

The implementation of the class `String` as described in the book allowed performing operations such as `substring` in constant time (independent on the substring length). The general idea of the implementation was for multiple instances of strings to share a single character array and to remember which part of the array belongs to each instance. Since `String` instances didn't allow user to change current data (instead, such methods created new objects), this implementation was safe. Starting with Java 7 update 6, the internal implementation of `String` has changed; Strings are still unmodifiable, but substring operation now creates new objects with it own copy of character array which only contains characters belonging to new `String`.

Your job is to create a class `CString` which offers similar functionality as the old official implementation of the `String` class: it represents unmodifiable strings on which substring methods (and similar) must be executed in O(1) complexity, which you can achieve by sharing the character array. Here is the official list of methods your `CString` must support:

- Constructors:
  - `CString(char[] data, int offset, int length)`;
  - `CString(char[] data)`;
  - `CString(CString original)`; if originals internal character array is larger than needed, your new instance must allocate its own character array of minimal required size and copy data; otherwise it must reuse original's character array
  - `CString(String s)`; represent same character data as Java's `String`; you must copy its data
- instance methods:
  - `length()`;
  - `charAt(int index): char`;
  - `toCharArray(): char[]`; allocates a new array, copies string content into it and returns it
  - `toString(): String`;
  - `indexOf(char c): int`; returns index of first occurrence of char or -1
  - `startsWith(CString s): boolean`; returns `true` if this string begins with given string, `false` otherwise
  - `endsWith(CString s): boolean`; returns `true` if this string ends with given string, `false` otherwise
  - `contains(CString s): boolean`; returns `true` if this string contains given string at any position, `false` otherwise
  - `substring(int startIndex, int endIndex): CString`; returns new `CString` which represents a part of original string; position `endIndex` does not belong to the substring; `startIndex>=0`, `endIndex>=startIndex`
  - `left(int n): CString`; returns new `CString` which represents starting part of original string and is of length `n`; throw an exception if this can not be constructed; `n>=0`
  - `right(int n): CString`; returns new `CString` which represents ending part of original string and is of length `n`; throw an exception if this can not be constructed; `n>=0`
  - `add(CString s): CString`; creates a new `CString` which is concatenation of current and given string
  - `replaceAll(char oldChar, char newChar): CString`; creates a new `CString` in which each occurrence of old character is replaces with new character

- `replaceAll(CString oldStr, CString newStr): CString`; creates a new `CString` in which each occurrence of old substring is replaces with the new substring

Testing hint: do not forget to check what will be the result of:

```
new CString("abababab").replaceAll(new CString("ab"), new CString("abab"))
```

Problem 3.

In the book I have written about how Java implements the support for short form of for-loop: the idea is to have objects which represent multiple data act as iterators (*Iterator design pattern*) and provide those iterators from agreed-upon factory methods (agreement is specified through *Iterable* interface). Read from page 170.

Implement a class `IntegerSequence` which will allow user to loop from specified integer to specified integer with given step. Here is usage example which must work:

```
IntegerSequence range = new IntegerSequence(1, 11, 2);
for(int i : range) {
  for(int j : range) {
    System.out.println("i="+i+", j="+j);
  }
}
```

In this example *i* will be 1, 3, 5, 7, 9, 11 and for each value of *i*, the whole set of the same values for *j* will be printed (36 lines in total).

Important notes

Solve all of the problems in a single Eclipse project. Configure Eclipse to use two source directories: `src/main/java` for your source files and `src/test/java` for sources files of unit tests.

You are required to write the adequate number of unit tests for all of the classes developed in problem 1 and problem 2.

You must equip your project with `build.xml` script so that the project can be build from the command line. In that script, you must integrate all of the quality-checking tools which I have described in the book. It should be possible to run at least the following targets: `init`, `compile`, `compile-tests`, `run-tests`, `pmd`, `javadoc`, `clean`.

Please observe that the `build.xml` slightly differs from the example given in the book: instead of declaring paths to various tools as properties directly given in `build.xml` file, the file is made portable by externalizing user-specific properties into additional configuration file, which must be present in the same directory.

In this homework, *all of the classes* in all three problems should have appropriate javadoc.

Please check the results of PMD tool. You do not have to accept every complaining the tool gives, but trtry to solve all bigger issues.

Considering source writing conventions, you can ignore warnings that after `if`, `for` etc. a space is mandatory; you can write in your source: `if(` and you don't have to write `if (`. Also, you can ignore the warnings that method arguments and some variables should be declared `final`. It is OK to have code written as follows:

```
public int successorSquare(int n) {
   int succ = n+1;
   return succ*succ;
}
```

To make quality-check tools happy, you could rewrite it as follows (but you do not have to):

```
public int successorSquare(final int n) {
   final int succ = n+1;
   return succ*succ;
}
```

Please note. You can consult with your peers and exchange ideas about this homework *before* you start actual coding. Once you open you IDE and start coding, consultations with others (except with me) will be regarded as cheating. You can not use any of preexisting code or libraries for this homework (whether it is yours old code or someones else), unless explicitly provided by me. Additionally, for this homework you can not use any of Java Collection Framework classes or its derivatives. Document your code!

In order to solve this homework, create a blank Eclipse Java Project and write your code inside. You must name your project's main directory (which is usually also the project name) `HW03-`*yourJMBAG*; for example, if your JMBAG is 0012345678, the project name and the directory name must be `HW03-0012345678`. Once you are done, export the project as a ZIP archive and upload this archive to Ferko before the deadline. Do not forget to lock your upload or upload will not be accepted. Deadline is March 1st 2015. at 23:59.