Assignment 3: Methods and Classes

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

Prepare Eclipse for Assignment 3

Create a new *package* with the name YourLnuUserName_assign3 inside the Java project 1DV506 and save all program files for this assignment inside that package.

Lecture 6 - Methods

In exercises 1 and 2 below you are supposed to create a number of static methods. They should all be created inside the same class that contains the main method. The main method should work as test program that demonstrates how the different methods in your class can be used.

- **Exercise 1**
  Notice: In this exercise ou are supposed to implement all methods from scratch without any use of the array related methods in the Java library.

  Create a class Arrays.java that apart from the main method also contains the following **static** methods:
  1. Method int sum(int[] arr), adding all elements of the arrayarr and returning the sum.
  2. Method String toString(int[] arr), creating a string containing a nice-looking print out of the content of the array. It should be possible to use it in the following way.

     ```
     int[] n = {3,4,5,6,7};
     String str = Arrays.toString(n);
     System.out.println("n = " + str);
     ```

  3. Method int[] addN(int[] arr, int n), creating and returning a new array, where n have been added to all elements in arr. Array arr should be left unchanged.
  4. Method int[] reverse(int[] arr), creating and returning a new array, consisting of all elements in arr in reverse order. Array arr should be left unchanged.
  5. Method boolean hasN(int[] arr, int n), returning true if n is contained in the array arr, false otherwise.
  6. Method void replaceAll(int[] arr, int old, int nw), replacing all occurrences of old with nw in arr.
  7. Method int[] sort(int[] arr), returning a new sorted array (increasing order), containing the same set of integers as arr. Array arr should be left unchanged.
  8. **(VG-exercise)** Method boolean hasSubsequence(int[] arr, int[] sub), returning true if the sequence sub is a subsequence of array arr, false otherwise. The case hasSubsequence({1,2,3,4,5}, {3,4,5}) should return true since {3,4,5} is a part of {1,2,3,4,5}. The case hasSubsequence({1,2,3,4,5}, {1,3,5}) should return false since the exact sequence of elements {1,3,5} cannot be found in{1,2,3,4,5}.
  9. **(VG-exercise)** Method boolean isLarger(int[] a1, int[] a2), returning true if array a1 is larger than a2. Starting from the head, we compare the two arrays element by element. As soon as we find an element that is larger than the corresponding element in the other array we stop and declare the array with the largest element as “larger”. If they have the same initial sequence of elements but one array contains more elements, than the longer array is larger. A few examples;

     ```
     {4,3} vs {1,2,3,4,5} ==> {4,3} is larger since 4 > 1
     {1,2,3,4} vs {1,3,5} ==> {1,3,5} is larger since 3 > 2
     {1,2,3,4} vs {1,2,3} ==> {1,2,3,4} is larger since the first array is longer
     ```

- **Exercise 2**
  Create a class SweID.java that apart from the main method also contains a number of static methods related to the Swedish identity number in the form YYMMDD-NNNN. Information about the structure of Swedish identity numbers can be found at Wikipedia (Wikipedia: Personal identity number (Sweden)).

  We expect you to consider each ID number as a single string of type "YYMMDD-NNNN". The class should contain the following static methods:

  1. Methods getFirstPart(String sweID) and getSecondPart(String sweID), returning the first part (YYMMDD) and second part (NNNN) of the identity number, respectively.
  2. isFemaleNumber(String sweID), returning true if the personal identity number belongs to a woman.
  3. areEqual(String id1, String id2), comparing two ID numbers checking if they correspond to the same identity number.
  4. **(VG-exercise)** isCorrect(String sweID), returning true if the number is a correct identity number. To get a passed result you have to check that the date is correct (i.e. the year, month and day

should be correct). You must also check that the last digit of the number is correct according to the rules given in the link above.

Feel free to add more methods, if you think anything is missing. Suitable types for the return values are up to you to decide.

Examples:

- 640123-8826 is a correct female number
- 550414-0913 is a correct male number
- 551314-0913 is not correct number (unvalid month)
- 550414-0912 is not correct number (unvalid last digit)

Clarification: all students are supposed to do subtasks 1, 2, and 3. To get the highest grades you must also do subtask 4.

Lecture 7 - Create Your Own Classes

In the exercises below you are supposed to create your own classes. We also want that you, for each class (e.g. MultiDisplay), to create a test program (e.g. MultiDisplayMain) containing a main method that demonstrates how the different methods in your class can be used.

- **Exercise 3**
  Create a class MultiDisplay that when executed using this code:

```
MultiDisplay md = new MultiDisplay();

md.setDisplayMessage("Hello World!");
md.setDisplayCount(3);
md.display();                    // ==> print-out

md.display("Goodbye cruel world!", 2);  // ==> print-out

System.out.println("Current Message: "+ md.getDisplayMessage());
```

  results in the following console print-out:

```
Hello World!
Hello World!
Hello World!
Goodbye cruel world!
Goodbye cruel world!
Current Message: Goodbye cruel world!
```

  The class MultiDisplay should of course be able to handle other messages and other numbers of display counts.

- **Exercise 4**
  Download and install the class AlarmClock. Then write a program AlarmMain that uses AlarmClock to:
  1. Set clock time to 23:48
  2. Display time
  3. Set alarm to wake up at 6:15
  4. Let the clock "tick" for 500 minutes
  5. Display time again

  Notice: You are not allowed to make any changes in the AlarmClock class except maybe to change the package name.

- **Exercise 5**
  Create a class Radio that when executed using this code:

```
System.out.println("Radio 1");
Radio r1 = new Radio();
System.out.println( r1.getSettings());  // Default settings

// Update Radio 1 settings
r1.turnOn();
r1.setVolume(3);
r1.channelUp();
r1.channelUp();
r1.channelUp();
System.out.println( r1.getSettings()); // New settings

r1.turnOff();
System.out.println( r1.getSettings());  // Reset default settings

System.out.println("\nRadio 2");
Radio r2 = new Radio();
r2.volumeUp();   // Radio off ==> No adjustment possible

r2.turnOn();
r2.volumeDown();   // volume = 0 ==> OK!
r2.volumeDown();   // volume < 0 ==> error and neglect
r2.setChannel(15); // out of range = ==> error and neglect
System.out.println( r2.getSettings());
```

results in the following console print-out:

```
Radio 1
Settings: On false, Channel 1, Volume 1
Settings: On true, Channel 4, Volume 3
Settings: On false, Channel 1, Volume 1

Radio 2
Radio off ==> No adjustment possible
New volume not within range!
New channel not within range!
Settings: On true, Channel 1, Volume 0
```

Notice

- The default settings of a Radio is `on = false, channel = 1, volume = 1` .
- The volume must be in range [0,5]
- The channel must be in range [1,10]
- You cannot adjust either volume or channel when the radio is off

Adjustments not in range, or if radio is off, should generate an error message and not change the settings.

Finally, the class `Radio` should contain the following nine methods

```
getSettings()  // A string with current settings
turnOn(), turnOff()  // turnOff() ==> restore default settings
setVolume(int newVolume), volumeUp(), volumeDown() // up/down ==> +- 1 step
setChannel(int newChannel), channelUp(), channelDown() // up/down ==> +- 1 step
```

Hint: Do not try to add all features at once. We suggest that you start with one of the adjustable parameters (say Volume) and try to make it work without any range checks. Then add range checks, then add Channel.

- **Exercise 6**
  Create a class `Point` that when executed using this code:

```
Point p1 = new Point();
Point p2 = new Point(3,4);

System.out.println(p1.toString());   // ==> (0,0)
System.out.println(p2.toString());   // ==> (3,4)

if (p1.isEqualTo(p2))              // False!
 System.out.println("The two points are equal");

double dist = p1.distanceTo(p2);
System.out.println("Point Distance: "+dist);

p2.move(5,-2);         // ==> (8,2)
p1.moveToXY(8,2);      // ==> (8,2)

if (p1.isEqualTo(p2))              // True!
 System.out.println("The two points are equal");
```

results in the following console print-out:

```
(0,0)
(3,4)
Point Distance: 5.0
The two points are equal
```

The class `Point` should of course be able to handle other points with different (x,y) values.**Notice:**

- The coordinates (x,y) are always integers.
- The method `toString` returns a string with coordinates suitable for print-outs.
- Distance between two points is computed in the same way as in Exercise 15, Assignment 1.
- Two points are *equal* if they have the same coordinates.
- Method `move` moves the point certain steps in x- and y-direction.
- Method `moveToXY` provide a new set of coordinates.

Lecture 8 - More Classes

This section contains a number of exercises where you are supposed to create you own classes. For each task, we expect a `Main` class, showing how all methods in the class or classes work. For example, for the class `Fraction.java` there should be a class`FractionMain.java` showing how all methods of `Fraction.java` can be used.

All classes are supposed to be commented and follow principles such as encapsulation.

- **Exercise 7**
  Create a class `Fraction.java`, representing a fractional number of the form`N/D`, where `N` (the numerator) and `D` (the denominator) both are integers. If the denominator is equal to zero, an error message

should be printed. The class should include the following members:

1. A constructor, creating and initializing a new fractional number.
2. Methods getNumerator and getDenominator, returning the numerator or denominator, respectively.
3. Method isNegative, returning true if the fractional number is negative.
4. Methods add, subtract, multiply, divide, performing the corresponding operations on two fractional numbers and returning a new fractional number. It is up to you to decide a proper way of handling the case when one of the fractional numbers have a zero denominator.
5. isEqualTo, comparing two Fraction-instances, checking if they correspond to the same fractional number.
6. toString, returning a string representation of the fractional number on the form N/D.

Feel free to add more methods, if you think anything is missing. Suitable argument and return types are up to you to decide.

Extra, voluntary work if you are interested in mathematics: Make sure that the fractional number is in the most simplified form possible. For example, the fractional numbers 2/4 and 35/50 should internally be represented as 1/2 and 7/10. This means that the internal representation always should be the two smallest integers N and D corresponding to the given fractional number. Useful information can be found at Wikipedia: Euclidean algorithm.

- **Exercise 8**
  Create a class Card, representing a playing card in an ordinary card deck with 52 cards. A card has a *suite* (4 different) and a *rank* (13 different). Write a class Deck initially containing 52 different objects of the class Card. The class Deck should contain methods for shuffling the deck, deal a card and telling how many cards are still in the deck. Note that it should only be possible to shuffle a deck if it contains 52 cards. (Information at Wikipedia about card decks and card games.)

  Also write a program PlayCardsMain, creating a card deck and dealing some cards, telling the number of remaining cards and which cards that have been dealt.

  Hint: Use enumeration types.

- **Exercise 9 (VG-exercise)**
  In this exercise you should use the Deck class from the previous exercise. In the patience (single player card game) 1-2-3 you take one card at a time from the deck at the same time as you are counting 1,2,3,1,2,3,1,2,3 etc. You lose the game as soon as you get an Ace when counting "one", a 2 card when counting "two", or a 3 card when counting "three". The chances to win, to make it through the whole deck without losing, are quite small. But how small?

  Write a program Play123Main that plays the 1-2-3 game 10000 times and then computes the probability (%) that you win the game. The program should use a method play123 that plays the game once and reports true if you win (or false if you lose) that particular game.

- **Exercise 10 (VG-exercise)** (The Drunken Walker)
  Create a class RandomWalk.java, simulating a random walk. A random walk is basically a sequence of steps in some enclosed plane, where the direction of each step is random. The walk terminates when a maximal number of steps have been taken or when a step goes outside the given boundary of the plane.

  For this task, assume a plane given by a grid, with the point (0, 0) at the center. The size of the plane is given by an integer; if the given integer is *k*, then the values of the *x* and *y* coordinates can vary from -*k* to *k*. Each step will be one unit up, one unit down, one unit to the right or one unit to the left (no diagonal movements).

  The class RandomWalk will have the following instance data :

  - X coordinate of the current position
  - Y coordinate of the current position
  - The maximum number of steps in a walk
  - The number of steps taken so far in the walk
  - The size of the plane (according to the description above)

  Other members

  - RandomWalk(int max, int edge): the maximum number of steps is max and edge is the size of the plane. The start position is set to (0, 0).
  - String toString(): returns a string containing the number of steps taken so far and the current position.
  - void takeStep(): simulates taking a single step. Generate a random number, taking on four different values, and let them correspond to a movement up, down, to the right and to the left, respectively. The method should also increase the number of steps taken.
  - boolean moreSteps(): returns true if the number of steps taken is less than the maximal number of steps, otherwise false is returned.
  - boolean inBounds(): returns true if the current position is inside the boundary of the plane, otherwise false is returned.
  - void walk(): simulates a random walk, i.e. steps are taken until the maximum number of steps has been taken or until a step goes outside the boundary of the plane.

Simulation
Create another class DrunkenWalk, simulating walks of drunken people on a platform in a lake. The program should read the boundary, the maximum number of steps, and the number of drunks to simulate. One drunk at a time should be put on the platform and perform its walk. Your program should count how many drunks fall into the water. Test your program for some different values of size and number of steps. Example of an execution:

```
Enter the size: 10
Enter the number of steps: 50
Enter the number of walks: 150
Out of 150 drunk people, 14 (9.34%) fell into the water.
```

Submission

All exercises should be handed in and we are only interested in your .java files. (Notice that the VG exercises 1.8, 1.9, 2.4, 9 and 10 are not mandatory.) Hence, zip the directory named YourLnuUserName_assign3 (inside directory named src) and submit it using the Moodle submission system.