Assignment 3: Methods and Classes

Problems?
Do not hesitate to ask your teaching assistant 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**
  Create a class `Arrays.java` that apart from the `main` method also contains the following **static** methods:
  - Method `int sum(int[] arr)`, adding all elements of the array `arr` and returning the sum.
  - Method `String toString(int[] arr)`, creating a string containing a nice-looking print 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);
```

  - 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.
  - 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.
  - Method `boolean hasN(int[] arr, int n)`, returning `true` if `n` is contained in the array `arr`, `false` otherwise.
  - Method `void replaceAll(int[] arr, int old, int nw)`, replacing all occurrences of `old` with `nw` in `arr`.
  - 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.
  - 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}`.

Notice: You are supposed to implement all these methods from scratch without any use of the array related methods in the Java library.

- **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` and `getSecondPart`, returning the first part (YYMMDD) and second part (NNNN) of the identity number, respectively.
  2. `isFemaleNumber, isMaleNumber`, returning `true` if the personal identity number belongs to a woman or a man, respectively.
  3. `areEqual`, comparing two ID numbers checking if they correspond to the same identity number.
  4. **(VG-exercise)** `isCorrect`, 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 arguments and 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`), 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 `TextAnalyzer` that when executed using this code:

```
String text = "My name is Anakin Skywalker. My age is 42.";
TextAnalyzer ta = new TextAnalyzer(text);

System.out.println("Char Count: "+ ta.charCount());
System.out.println("Upper Case Count: "+ ta.upperCaseCount());
System.out.println("Whitespace Count: "+ ta.whitespaceCount());
System.out.println("Digit Count "+ ta.digitCount());

if (ta.containsChar('x'))
        System.out.println("The text contains char \'x\'");
if (ta.containsString("nakin"))
        System.out.println("The text contains substring \"nakin\"");
```

  results in the following console print-out:

```
Char Count: 42
Upper Case Count: 4
Whitespace Count: 8
Digit Count 2
The text contains substring "nakin"
```

  The class `TextAnalyzer` should of course be able to handle other texts in a correct way. The methods `containsChar` and `containsString` should of course also give a correct result (true/false) for other characters and strings.

- **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.

Submission

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