COSC102 Spring 2015

More Procedural Programming Homework 2

Due date: **Monday February 9th, at 11:55 p.m. on Moodle**

Overview

In this assignment you manipulate arrays and strings using static methods that you implement. It is good Java practice to prepare you for Exam 1.

Be sure to use the same method names as the one given in this handout. You should write your code in a file called `Hw2.java`, that is stored in a `02` folder under your `cosc102\hw` path. It is important to learn to keep your files organized.

You should include a `main()` method with tests that validate the implementation of the methods described below.

Question 1: Duplicate removal

As an easy practice for Exam 1, implement the method

```
public static int[] removeDuplicates (int[] array)
```

that takes as input an array and (always) returns a copy of the array with all duplicates removed.

More specifically, the returned array should contain elements from the input array in order of their first occurrence in the input array, and elements should appear only once in the returned array. Moreover, the returned array's length should be exactly the number of unique items in the input array, i.e., there should be no extra "unfilled" spaces in the returned array.

Question 2: String slicing

This question is more challenging as it demands to imitate Python's string slicing operations with Java methods. It requires to observe how Python's slicing handles default cases, i.e. figuring out assumptions made on special cases.

Your task is to implement the following two methods so as to imitate Python's String slicing:

```
1. public static String slice (String source, int begin, int stride)

2. public static String slice (String source, int begin, int end, int stride)
```

The slice functionality provided by these two methods is always based on a `source` String and use either two or three additional parameters to describe the substring to return.

Note — We say that the `slice` method is overloaded since there are two different methods with the same name but different method signatures. (A method signature is the name of a method, along with its number and type of parameters.)

A general description of the `slice` behavior is that the returned string retrieves characters from the original string, `source`,

- starting at position `begin`,
- until the `end` index is reached or exceeded or its default value when not specified (i.e. in the method `1.`)
- using a `stride` to move to the next character. The `stride` parameter permits to skip characters: a `stride` of value 2 includes every other character.

Similar to Python, this functionality must also accept negative indices. For `begin` and `end` an index of -1 refers to the right-most character of the string (it is like counting backwards from the end of the string). For example, in the string *toucan*,

- index -1 corresponds to the final character, *n* (also accessible with index 5) and
- index -4 corresponds to *u*, also accessible with index 2.

  (See the diagram below with indexing forwards/backwards labelled.)

Negative `stride` values are also accepted. They permits to go backwards when retrieving characters.

The goal is to reproduce Python slicing operation *as close as* possible but with less default alternatives. With the above two signatures the slice functionality you are implementing does not include the default cases used when the begin index and/or the stride are omitted. In particular Python permits the following square-bracket syntax `[:y:]`, while the two signatures only cover `[x:y:z]` and `[x::z]`.

You should start implementing the `slice` method `2.`, i.e., the three-integer version of the overloaded method.

In the `slice` method `1.`, i.e., the two-integer version of the method, as Python operates, a default value is used for the missing index and the `slice` method `2.` will be handy then.

Investigate Python implementation which defaults the missing value to whatever makes sense in the circumstances, considering the `stride` direction used.

Since negative and out-of-bounds indexing are complicated, I recommend that your first step in implementing `slice` method `2.` is

- to convert negative `begin` and `end` indices to sensible indices (Javas friendly, mostly positively ones) and
- to make them "legal" for the string `source`.

Once you have done that and implemented the most usual situations, think about the special cases. They may include but are not limited to the following

- `end` index greater than `begin` index with negative `stride`
- `begin` index greater than `end` with positive `stride`
- `stride` of 0
- `begin/end` index (after conversion if necessary) out of range

To do this exercise and deal with the special cases it is important to understand how Python string slicing operates. The following examples may help you to initially start and to later refine your two methods. You should experiment on your own.

1. In short

```
slice("toucan", 1, 5, 2)  -->  "oc"
slice("toucan", 4, 0, -1) -->  "acuo"
slice("toucan", 2,    -1) -->  "uot"
slice("toucan", 1,     2) -->  "ocn"
```

2. From sourceforce.net

```
mystr = "This is what you have"
   #    +012345678901234567890  Indexing forwards  (left to right)
   #     1098765432109876543210- Indexing backwards (right to left)`
   # note that 0 means 10 or 20, etc. above

mystr[5:7]     returns  "is"
mystr[-8:-5]   returns  "you"
```

3. From experimenting in IDLE "' string = "0123456789" >>> string[0:5] '01234' >>> string[0:5:-1] '' >>> string[5:1:1]" >>> string[0:4:0]

Traceback (most recent call last): File "", line 1, in string[0:4:0] ValueError: slice step cannot be zero

string[0:11:1] '0123456789' string[0:-8:-1] '' string[-8:-11:-1] '210' string[-10:3:-1] '' string[0:-13:-1] '0' string[-1:-13:-1] '9876543210' "'

4. More examples, diagrams and descriptions

https://docs.python.org/release/1.5.1p1/tut/strings.html

https://docs.python.org/2/tutorial/introduction.html#strings

http://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/strings3.html

http://www.pythoncentral.io/cutting-and-slicing-strings-in-python/

http://forums.udacity.com/questions/2017002/python-101-unit-1-understanding-indices-and-slicing

Additional Requirement

String concatenation is expensive. Correct but inefficient code will receive partial credit.

To make the code more efficient for full credit, try one of the following techniques to make the code run in linear time, rather than quadratic time.

- Use a StringBuilder object. You can instantiate an empty `StringBuilder` object and append characters onto it using the `append()` method, and finally return the result of calling `toString()` on the object to get an actual `String`. Look back at the decription in the textbook appendix.
- Create an array of characters of the right length (you can calculate this from the integer parameters), and then put the characters in that array. Finally, use the `String` constructor that takes an array argument to return a `String` object from the array in linear time, without a lot of concatenation.

Submission

Submit only one zip file `hw2.zip` containing only your `Hw2.java` file, where the three methods described above are implemented.

Remember that you should declare and use helper functions in your code.

Write as comments in your `main` method the input you gave to each method and the resulting output they produce so as to show us you **fully** tested your implementation.

WARNING: Code that does not compile will automatically receive a zero. Code that doesn't compile is impossible to test, let alone reason about. You should compile and submit often so that you always have a running, even if partially complete, version of your code uploaded.