Object Oriented Programming Assignments(OOP)

I semester

To complete the tasks, you need to create an account on the public Internet resource github.com and a repository "**OOP**".

Tasks are performed in the *Java* language. We recommend using the *Community* version of the IntelliJ IDEA IDE.

All decisions must be accompanied by a set of tests that check the correctness of the completed task.

To complete the first "*Heap Sort*" task, you need to create the **Task_1_1** project. Name the following tasks appropriately.

Examples for tasks that can be used as the first test are indicated in green.

Additional requirements for the task are indicated in gray, which are not mandatory for implementation, but bring additional points.

Comments to the problem are highlighted in yellow.
Additional terms and restrictions.

1. Familiarity with the Java programming language

1.1. Heapsort

Implement the classic heapsort algorithm with a suite of tests.

Input	Output
{5,4,3,2,1}	{1,2,3,4,5}

1.2. Determine if a substring exists in a file

The input is the name of the file and the string to be found. The string can contain letters of any alphabet in UTF-8 encoding.

Implement a function to determine an index start each occurrence of the specified substring.

The size of the input file can be much larger than the RAM of the computing device.
The size of the required substring is many times less than the available amount of RAM.

Test.txt	**Input**	**Output**
I want a pie!	Test.txt pie	{9}
I want juice!	Test.txt pie	{}

2. Containers

2.1. Stack

Implement a stack class with support for "*push*" (add an element), "*pop*" (take an element), and "*count*" (find out the number of elements) operations. Stack elements can be of different types. It should be possible to use a standard iterator to traverse the container.

In the implementation it is forbidden to use standard containers of the Java language, except for an array.

Input	Output
push(2) push(7) pop count	{2}

2.2. Priority queue

Implement a priority queue class. Two Insert operations *(key, value)* are supported - "*Insert"* into the queue and "*ExtractMax"*- deletes and returns the value with the maximum key. The data type of the key and value can be arbitrary. Standard container iteration mechanisms and *Stream API*s must be supported.

Input	Output
Insert 200, «dog» Insert 10, «human» ExtractMax –> 200, «dog» Insert 5, «pinguin» Insert 500, «parrot» ExtractMax -> 500, «parrot»	{{5,«pinguin» },{10,«human» }}

2.3. Quadtree

For a set of initial objects on the plane, build a "Quadtree" with the following operations:

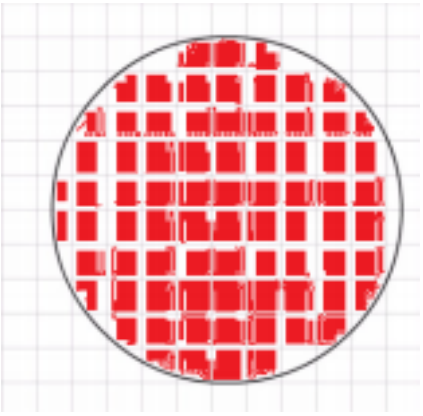
1) **add** a new object;
2) **delete** an existing object;

3) **Get an object corresponding to the coordinates (x, y)**, the values of the coordinates are arbitrary;
4) get all objects in a given rectangle;

Each node of the tree corresponds to a user-defined object.

Standard container iteration mechanisms and `ConcurrentModificationException` must be supported.

Based on the created container, implement the function of constructing a circle with a given center, radius, error in determining whether a point belongs to a circle. The created container must contain the minimum number of nodes.



An example of a circle with no minimum number of nodes.

Input	Output
Quadtree t = CreateCircle(xcenter = 3,ycenter = 4, r=5, tolerance = 0.01) *print t.get(x = 0,y = 0)*	"circle"

3. Data types

3.1. Gregorian calendar

Implement a Gregorian calendar class with the following behaviors:

1) every fourth year is a leap year (the date of February 29 is added to a leap year)
2) every final year of the century, unless it is divisible by 400, is not a leap year
3) determine the day of the week for the given date
4) Supports addition and subtraction arithmetic operations (date - date; date ± day, month, year)

Perform operations with the calendar:

1) What day of the week will be in 1024 days?
2) How many years, months and days ago was the victory day on May 9, 1945?
3) What day of the week were you born?
4) What month will be in 17 weeks?
5) How many days until the new year?
6) The next Friday the 13th of the month?

Using the `java.util.Date`, `java.util.Calendar` classes, the `java.time` package, and other third-party date libraries is **ONLY** valid in tests.

3.2. Record book

Implement a class of electronic student record book of *FIT* and provide the following functions:

1) Current GPA for the entire period of study.
2) Can a student get a "red" diploma with honors?
3) Will there be an increased scholarship this semester?

For the first test, use the data from your record book.

Requirements for a diploma with honors:

- 75% of marks in the diploma supplement (last mark) - “excellent”
- No final marks “satisfactory” in the grade book
- The qualification work is protected as “excellent”

3.3. Ordered sets

Implement a library for performing classical operations on objects: a set with, a partially ordered set (POS), a lattice-ordered set (LOS), a linearly-ordered set (Total Order):

1) check if transitivity is performed for the comparison operation between array elements;
2) topological sorting of the array in ascending order (the current element of the sorted array is greater or not comparable than all the previous ones);
3) finding the maximum elements in the array.

With the help of the created library, solve the applied problem "Find the smartest students in the group".

Input	Output
Group: *Maria, Vasily, Tatiana, Dmitry* 1) Maria > Vasily 2) Dmitry > Vasily 3) Maria > Dmitry	Maria, Dmitry, Vasily

4. Console Applications

4.1. Calculator

Implement an engineer calculator for real numbers. The user enters a prefixed expression on standard input. The calculator calculates the value and prints it to standard output. In addition to the standard operations (*+, -, *, /*), there is a set of functions (*log, pow, sqrt, sin, cos*).

Redefine the behavior of basic operations using dynamic library loading (plugin). Add support for working with degrees and complex numbers.

Input	Output
sin + - 1 2 1	0

4.2. Notebook

Make a notebook with a set of functions available from the command line:

- Add a note
- Delete entry
- Display all entries sorted by added time
- Display entries sorted by time of addition from a specified time interval containing keywords in the title.

The notebook data is serialized to a *JSON* file. It is recommended to use the `Jackson` or `Gson` libraries to work with the *JSON* format. For the analysis of command line parameters, it is also recommended to use third-party libraries.

Examples:
notebook -add “My note” “Very important note”
notebook -rm “My note”
notebook -show
notebook -show “14.12.2020 7:00” “17.12.2020 13:00” “My” “Your” “Me”