1. Описание проекта

Проект представляет собой программную библиотеку, содержащую реализацию геометрических объектов, контейнеров и алгоритмов, необходимых для решения задач плоской геометрии.

Библиотека должна содержать реализацию следующих геометрических объектов:

* точка;
* отрезок.

Для хранения этих объектов необходимо реализовать контейнеры:

* цепь;
* контур;
* набор отрезков;
* набор контуров;
* слой отрезков;
* слой контуров.

Отличие слоя от набора заключается в отсутствии топологических пересечений объектов, хранящихся в слое.

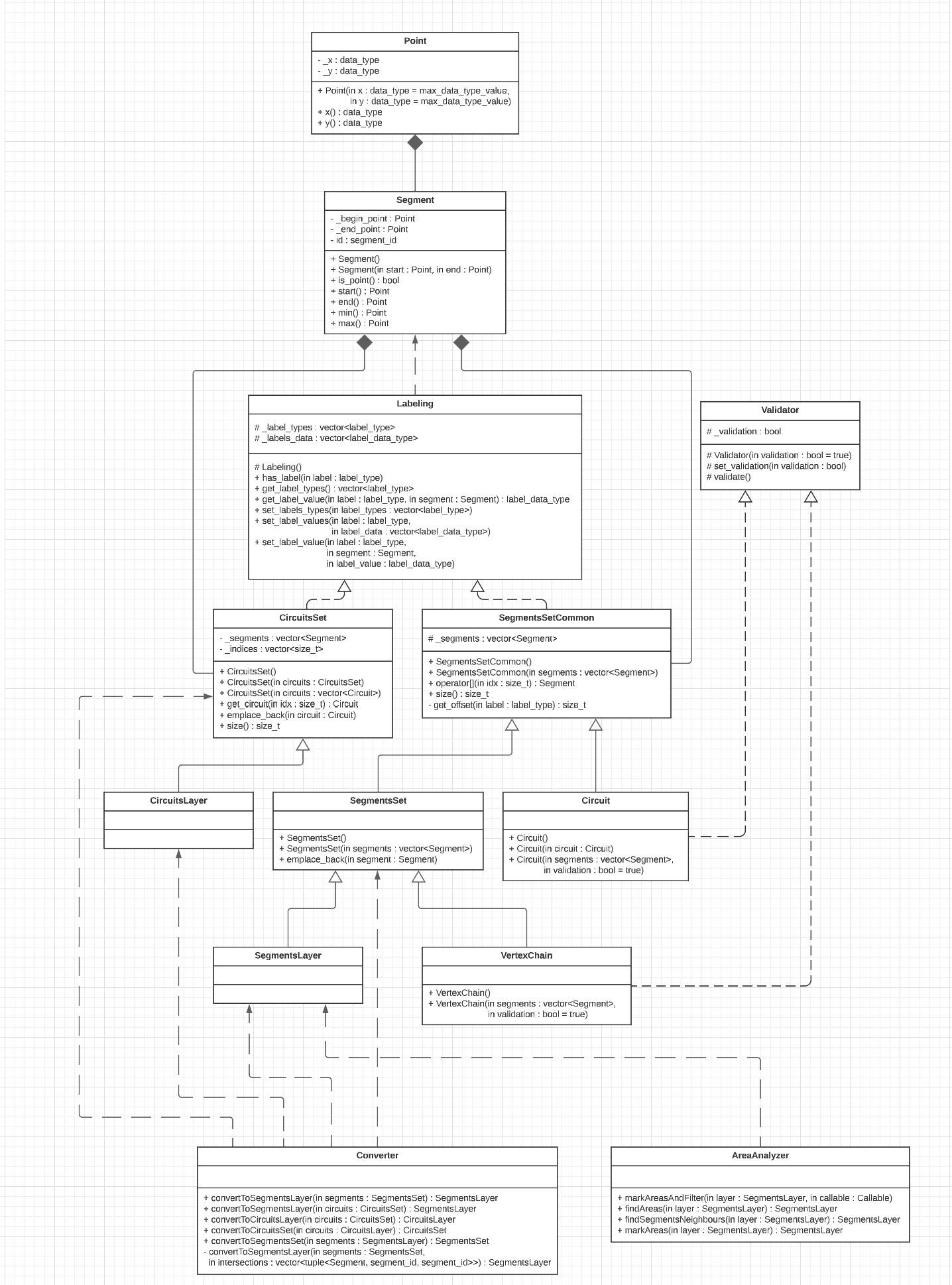
Библиотека должна содержать набор функций преобразования:

* из набора отрезков в слой отрезков;
* из слоя отрезков в набор отрезков;
* из слоя контуров в набор контуров;
* из набора контуров в слой отрезков.

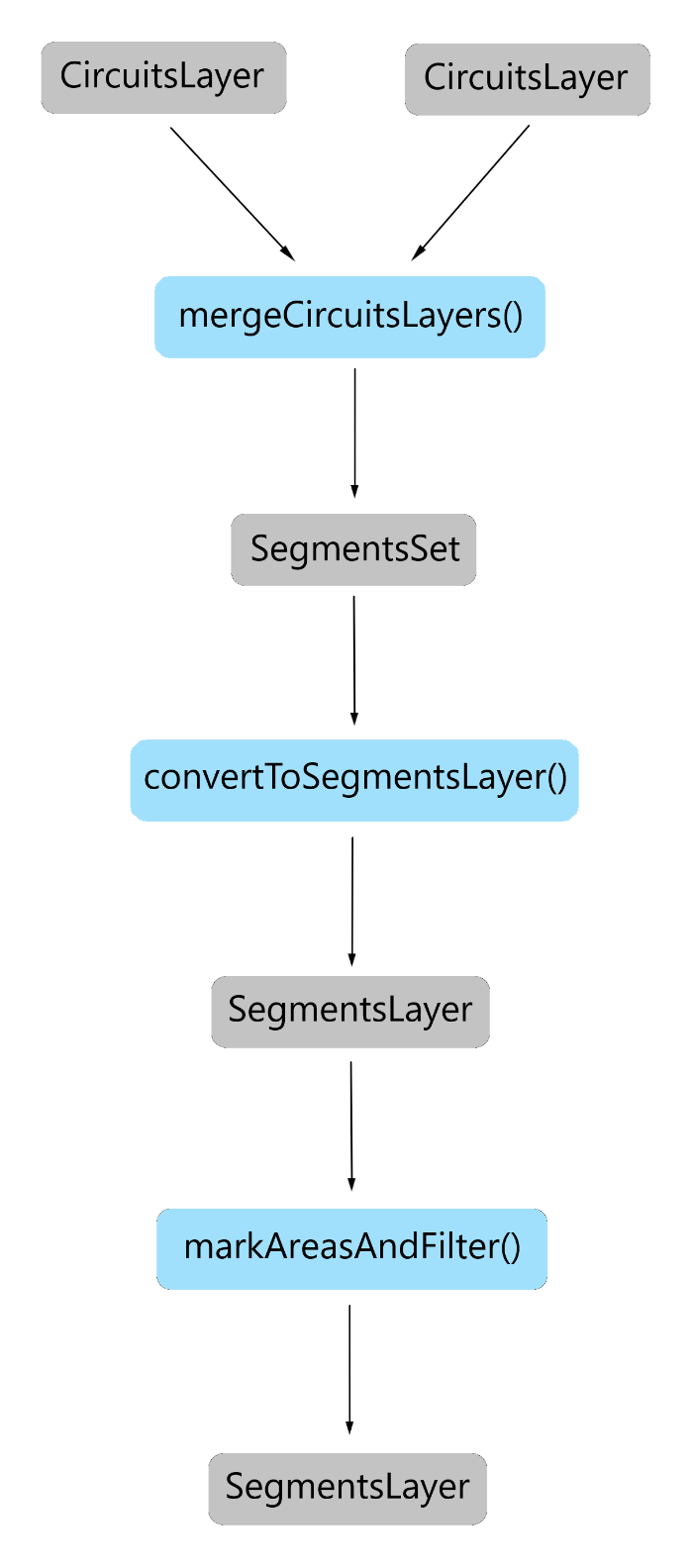
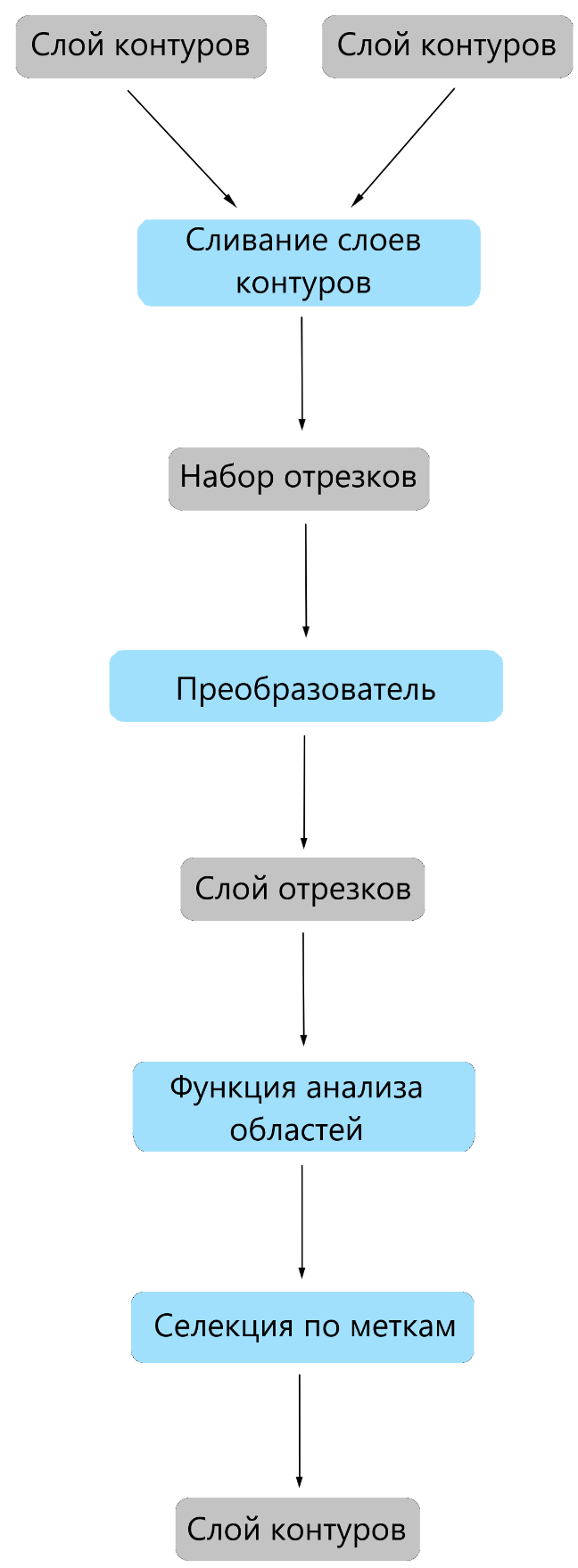
Библиотека должна содержать функционал, позволяющий выделять области из слоев отрезков и контуров с помощью операции “и”, удовлетворяющие заданной пользователем логической функции над слоями.

Требования для пользовательской функции:

* Вход: слой отрезков, рассматриваемый отрезок;
* Выход: логическое значение, означающее подходит ли рассматриваемый отрезок или нет.

1.  UML диаграмма классов
   1. Описание классов
      1. Базовые структуры:

* Point – структура, описывающая геометрическую точку;
* Segment – структура, описывающая геометрический отрезок.
  + 1. Интерфейсы:
* Labeling – интерфейс, обязывающий реализовать методы, необходимые для обеспечения функционала по работе с системой меток (установка, чтение, проверка наличия);
* Validator – интерфейс, обязывающий реализовать метод проверки валидности структуры. Решение запуска метода валидации остается за классом потомком.
  + 1. Контейнеры:
* SegmentsSetCommon – класс, обеспечивающий базовую функциональность контейнера отрезков (инициализирование, чтение, изменение) с поддержкой функционала меток;
* SegmentsSet – класс, описывающий набор отрезков с возможностью добавления новых отрезков;
* VertexChain – класс, описывающий цепь с возможностью валидации данных и добавления новых звеньев цепи;
* Circuit – класс, описывающий контур с возможностью валидации данных;
* CircuitsSet - класс, описывающий набор контуров с возможностью добавления новых контуров;
* SegmentsLayer – класс, эквивалентный SegmentsSet, получаемый посредством конвертации из набора отрезков, без возможности изменения данных;
* CircuitsLayer – класс, эквивалентный CircuitsSet, получаемый посредством конвертации из набора контуров, без возможности изменения данных.
  + 1. Другое:
* Converter – класс, содержащий методы по преобразованию одного контейнера в другой. Доступные преобразования:
  + Набор отрезков Слой отрезков
  + Набор контуров Слой контуров
  + Набор контуров Слой отрезков
  + Слияние двух контуров в один контур
* AreaAnalyzer – класс, содержащий методы, необходимые для селекции отрезков, удовлетворяющих заданной пользователем функции отбора.

1. Пример типового использования предоставленного функционала
   1. Пример типового использования функционала в коде

#include "gkernel/objects.hpp"

#include "gkernel/containers.hpp"

#include "gkernel/converter.hpp"

#include "gkernel/area\_analyzer.hpp"

using namespace gkernel;

int main() {

Circuit first\_circuit = {{

{{8, 13}, {16, 13}},

{{16, 13}, {10, 10}},

{{10, 10}, {8, 13}}

}};

Circuit second\_circuit = {{

{{8, 6}, {11, 12}},

{{11, 12}, {16, 12}},

{{16, 12}, {8, 6}}

}};

Circuit third\_circuit = {{

{{4, 3}, {7, 9}},

{{7, 9}, {12, 9}},

{{12, 9}, {15, 3}},

{{15, 3}, {4, 3}}

}};

Circuit fourth\_circuit = {{

{{2.5, 6}, {16, 6}},

{{16, 6}, {10, 3}},

{{10, 3}, {2.5, 6}}

}};

CircuitsLayer first\_layer = {{ first\_circuit, third\_circuit }};

CircuitsLayer second\_layer = {{ second\_circuit, fourth\_circuit }};

auto merged\_layers = Converter::mergeCircuitsLayers(first\_layer, second\_layer);

auto segments\_layer = Converter::convertToSegmentsLayer(merged\_layers);

SegmentsLayer filtered = AreaAnalyzer::markAreasAndFilter(segments\_layer, [](const SegmentsLayer& segments, const Segment& segment) {

return segments.get\_label\_value(0, segment) == 1 && segments.get\_label\_value(1, segment) == 1 &&

!(segments.get\_label\_value(2, segment) == 1 && segments.get\_label\_value(3, segment) == 1) ||

!(segments.get\_label\_value(0, segment) == 1 && segments.get\_label\_value(1, segment) == 1) &&

segments.get\_label\_value(2, segment) == 1 && segments.get\_label\_value(3, segment) == 1;

});

for (std::size\_t idx = 0; idx < filtered.size(); ++idx) {

std::cout << filtered[idx] << std::endl;

}

}