Отчет по лабораторной работе №3

Разработка приложения печати графиков

1. Постановка задачи

Необходимо разработать приложение с графическим интерфейсом, основная задача которого – чтение данных из различных источников, построение по считанным данным графиков и вывод их на печать (сохранение в формате pdf).

Исходные данные для печати соответствуют некоторому типу, который определятся пользователем. Данные определенного типа могут отображаться конкретным графиком, который ориентирован на этот тип данных.

Примеры данных:

1. Данные характеризуются парой **[значение, дата]**, хранятся в БД SQLite.
2. Данные представлены JSON файлом. Формат данных [**значение, дата].**

Общие требования к GUI

1. Данные загружаются путем выбора нужного файла. Данные в ПО не отображаются, отображается только график, построенный относительно считанных данных.
2. При печати в pdf выбирается место сохранения графика.

Начальный вариант архитектуры ПО предложен на UML-диаграмме (рис. 1), в него требуется внести изменения с целью снижения связности архитектуры. Используется принцип внедрения зависимости. Реализация внедрения зависимости с помощью IOC контейнера.

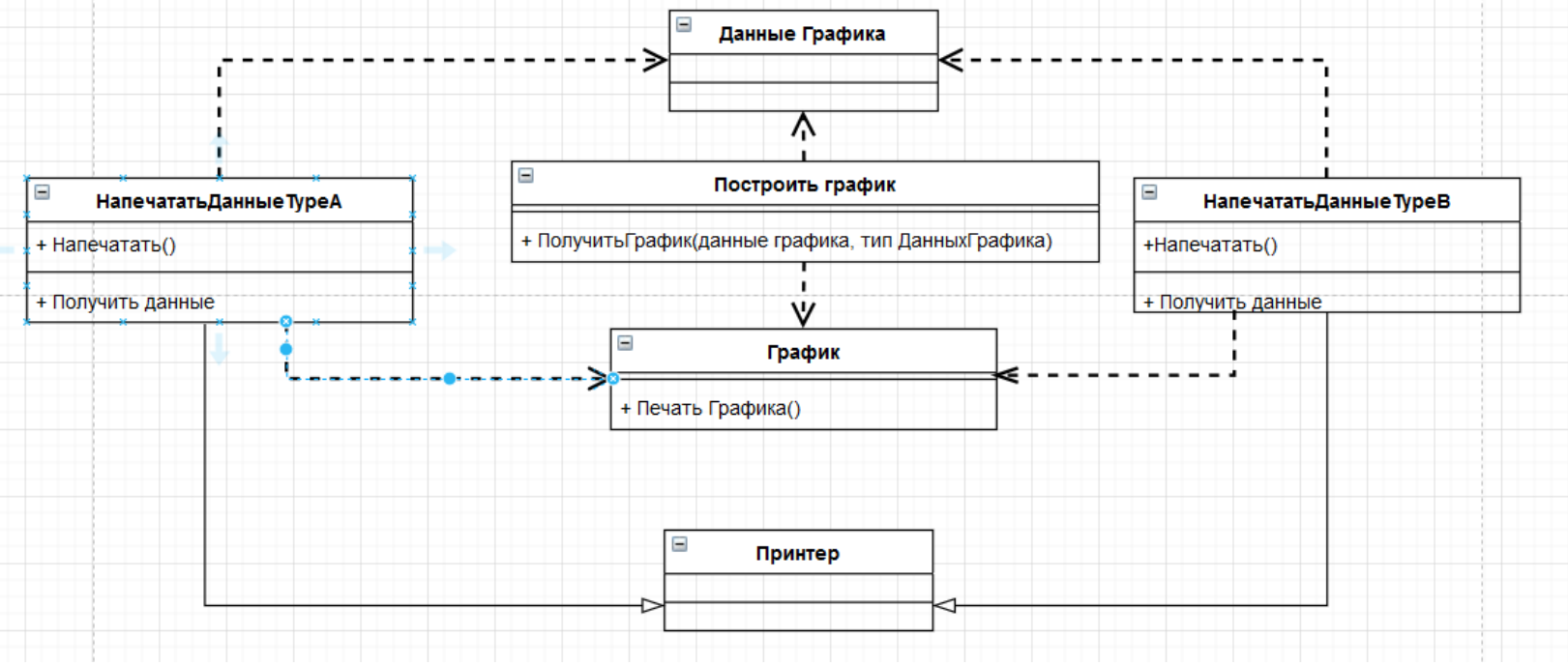


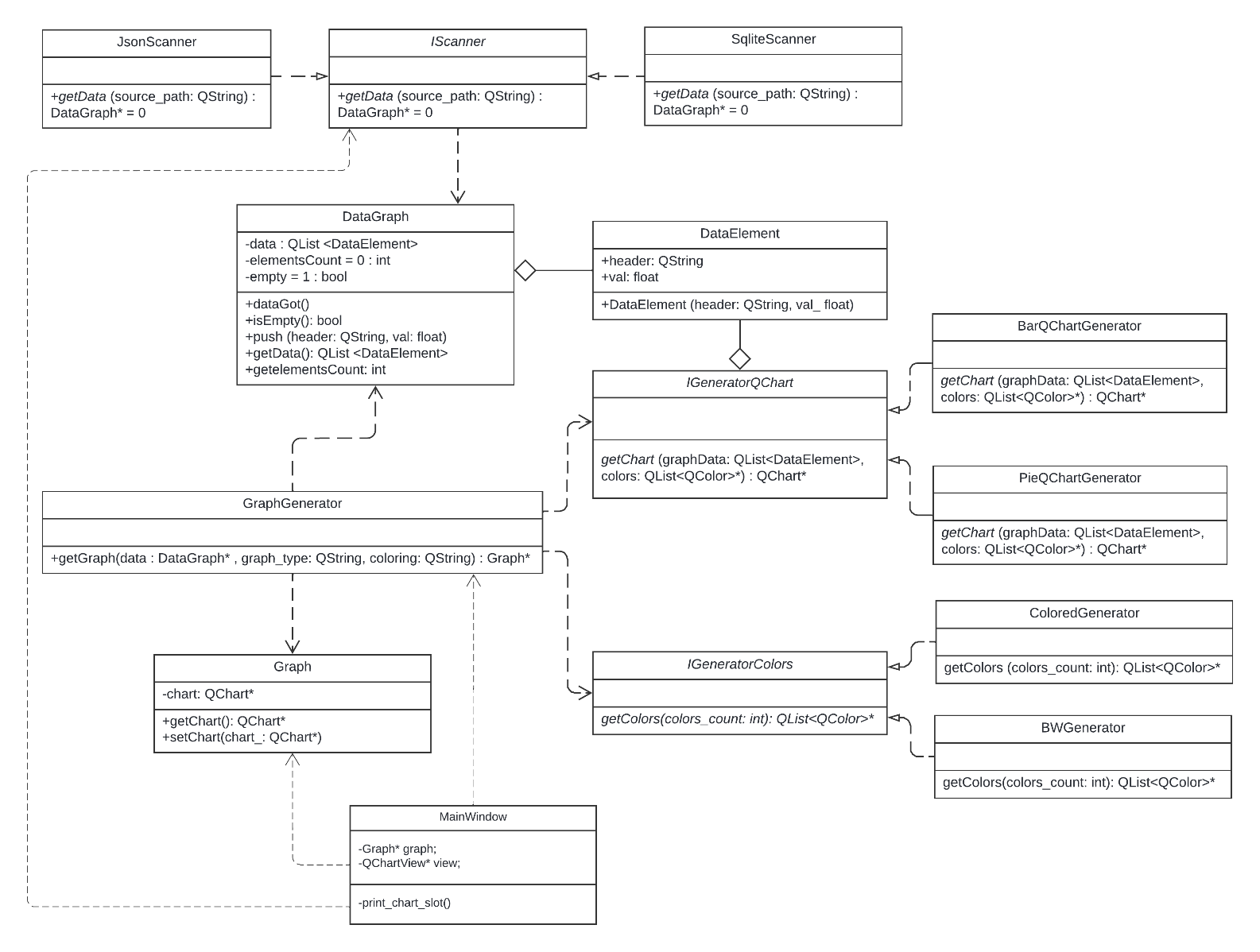
Рис. 1 Начальный вариант архитектуры

При разработке архитектуры необходимо учесть

1. Возможность добавления новых графиков (графики отличаются видом и данными)
2. Изменение визуального стиля графиков (цветной, черно белый).

2. Предлагаемое решение

Архитектура программы представлена на UML-диаграмме.



Классы изначального варианта архитектуры, соотносящиеся с классами предложенной, будут указаны в скобках.

Для получения объекта Graph (График), как и в изначальном варианте, используется класс GraphGenerator (Построить график). Нецелесообразным является вариант создания конкретного графика, введя четыре наследника GraphGenerator (ч/б столбачый, ч/б груговой, цветной столбачый, цветной груговой), так как это сильно ограничило бы возможность расширения архитектуры. Поэтому фабрика GraphGenerator, использует для получения объекта Graph три абстракции:

1. абстракцию данных графика - DataGraph (Данные Графика),
2. абстракцию списка цветов раскраски графика в виде QList<QColor>,
3. абстракцию различных по типу графиков в виде объекта QChart ().
4. DataGraph (Данные Графика) содержит данные в виде списка элементов структуры DataElement (структура хранит заголовок данного и его значение). За получение объекта DataGraph отвечает абстрактная фабрика IScanner с реализациями для конкретного типа читаемый файлов JsonScanner и SqliteScanner. Такая архитектура позволяет DataGraph не зависеть от типа читаемых файлов - устраняется зависимость между “НапечататьДанныеTypeA” “НапечататьДанныеTypeB” и “Данные графика” предложенной архитектуры.
5. За получение объекта QList<QColor> отвечает абстрактная фабрика IGeneratorColors с реализациями для конкретного набора цветов ColoredGenerator и BWColoredGenerator. Таким образом, Graph не зависит от способа получения цветов раскраски.
6. За получение объекта QChart отвечает абстрактная фабрика IGeneratorQChart с реализациями для конкретного типа графиков BarQChartGenerator и PieQChartGenerator. Чтобы избежать зависимости фабрики от класса DataGraph, IGeneratorQСhart использует данные графика в виде списка QList<DataElement>, который извлекается из DataGraph в GraphGenerator. Аналогично избегается зависимость от IGeneratorColors - готовый QList<QColor> ему передает аргументом GraphGenerator.

Таким образом, фабричный метод GraphGenerator::getGraph принимает аргументами элемент DataGraph, строковое имя типа графика (piechart/barchart) и строковое имя раскраски (bw/colored). Последние два аргумента используются в методе для связывания интерфейсов IGeneratorQСhart и IGeneratorColors с нужными реализациями посредством использования IOC-контейнера.

Связывание же IScanner с наследником осуществляется аналогично, но в классе главного окна при получении имени файла.

Альтернативой класса Принтер вместе с наследниками в предложенной реализации является метод print\_chart\_slot() главного окна, печатающий объект QChartView – визуальное представление графика. Такому объекту привязана модель графика - объект QChart, причем право владения на данный QChart может принадлежать только одному QChartView.

Надо заметить, что при этом для отображения графика в главном окне также требуется привязать QChart к QChartView главного окна, поэтому print\_chart\_slot() работает с QChartView главного окна, а не с полученной моделью QChart. Иначе право на отображаемый в главном окне график уходило бы принтеру, и график бы исчезал из окна.

Таким образом, print\_chart\_slot работает с абстракцией визуального представления графика в виде QChartView и не зависит ни от его типа, ни от раскраски, ни от способа получения данных. Так как печатаемый объект QChartWiew зависит только от QChart из класса Graph, при необходимости расширения архитектуры

* в направлении **новых форматов читаемых файлов** print\_chart\_slot остается от них независимым: потребуется только добавить наследников IScanner, или, при необходимости, ввести другой способ получения DataGraph (генерация случайных данных, например, не обязана удовлетворять интерфейсу фабрики IScanner)
* в направлении добавления **новых типов и расцветок** графиков print\_chart\_slot остается от них независимым: потребуется только добавить наследников IGeneratorQChart, IGeneratorQColors.
* в направлении **другого расширения** графиков print\_chart\_slot также остается от них независимым: потребуется добавить новые зависимости IGeneratorGraph (например, ввести абстрактную фабрику, создающую различные оси на графике).

Такая архитектура отвечает требованиям принципа инверсии управления, который реализуется за счет механизма внедрения зависимости посредством IOC-контейнера.

MVC (Model-View-Controller)

При разработке приложения также учитывалась концепция MVC (Model-View-Controller), основная идеология которой – разделить части программы, отвечающие за хранение и доступ к данным, их отображение и взаимодействие с пользователем на три отдельных модуля:

* Модель – осуществляет соединение с источником данных, служит их логической моделью, предоставляет интерфейс другим компонентам архитектуры
* Вид (представление) – обеспечивает конечное (например, экранное) представление данных для пользователя. Вид получает из модели модельные индексы, являющиеся ссылками на элементы данных. Сообщая модельные индексы модели, вид может получить элементы из источника данных.
* Контроллер - отвечает за пользовательский интерфейс. Он обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель и представление для реализации необходимой реакции.

Так, например, в классе MainWindow:

* QFileSystemModel \*table\_model - файловая **модель** для папки выбора файла, предназначенная (в контексте данной программы) для отображения в QTableView \*table\_view – табличного **представления** этой папки.
* Член-данное QChart \*chart объекта Graph\* graph – **модель** графика, которая будет отображаться в QChartView\* view – визуальном **представлении** графика. Пустая модель графика QChart \*empt также в некоторых случаях отображается в том же представлении. Более того, каждый раз создавая график, мы получаем его модель, и только затем отображаем в QChartView\* view методом QChartView::setChart(QChart\*), сам объект визуального представления не обновляется.

1. Коды программ

Файлы находятся в приложенном архиве и на удаленном репозитории

* DataElement.h – структура DataElement
* DataGraph.h – класс DataGraph
* Graph.h - класс Graph
* GraphGenerator.h – класс GraphGenerator
* IChartData.h – класс IChartData
* IGeneratorColors.h – интерфейс IgeneratorColors, классы ColoredGenerator, BWGenerator с реализацией в IGeneratorColors.cpp
* IGeneratorQChart.h – интерфейс IgeneratorQChart, классы PieQChartGenerator , BarQChartGenerator
* IOCContainer.h – IOC-контейнер
* IScanner.h - интерфейс IScanner, классы SqliteScanner, JsonScanner
* mainwindow.h – класс главного окна с реализацией в mainwindow.cpp
* main.cpp – начальная точка выполнения программы

4.Инструкция пользователя

Работа с графиком начинается с нажатия кнопки «Open directory». В открывшемся диалоговом окне выбирается нужная папка, хранящая файл с данными. Файл выбирается нажатием на него в табличном представлении, после чего в правой части окна отобразится график. Для выбора другого файла изменяется выбор в табличном представлении. Также возможно выбрать новую директорию. Тип графика изменяется выбором в выпадающем списке, а раскраска графика изменяется на черно-белую/цветную установлением/снятием галочки напротив надписи «B/w chart». Для сохранения выведенного графика в формате pdf предназначена кнопка «Print graph», открывающая диалоговое окно выбора папки сохранения. Файл сохраняется под названием «out.pdf». Если в папке назначения уже есть такой файл, он перезаписывается.

5.Тестирование

Тестирование описано в приложенном файле «Тестирование лаб\_3.xlsx». запусков приведены ниже.