Министерство образования и науки РФ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

по курсу «Высокопроизводительные вычисления»

на тему «Разработка параллельной программы для кластера

облачного типа».

Вариант №6.

Выполнил:

студент группы 17ВВ1

Кокин Д.Д.

Принял:

к.т.н., доцент

Федюнин Р.Н.

Пенза 2021

**Содержание**

[Введение 4](#_Toc71935370)

[1. Постановка задачи 5](#_Toc71935371)

[2. Выбор решения 6](#_Toc71935372)

[3. Описание разработки программы 8](#_Toc71935373)

[3.1. Разработка клиентской части приложения 8](#_Toc71935374)

[3.2. Разработка серверной части приложения 10](#_Toc71935375)

[4. Разработка диаграмм UML 12](#_Toc71935376)

[4.1. Разработка диаграммы вариантов использования 12](#_Toc71935377)

[4.2. Разработка диаграммы развертывания 14](#_Toc71935378)

[4.3. Разработка диаграммы последовательности 15](#_Toc71935379)

[4.4. Разработка диаграммы деятельности 16](#_Toc71935380)

[4.5. Разработка диаграммы классов 17](#_Toc71935381)

[5. Отладка и тестирование 18](#_Toc71935382)

[6. Руководство пользователя 19](#_Toc71935383)

[Заключение 25](#_Toc71935384)

[Список используемых источников 26](#_Toc71935385)

[Приложение А. Листинг сервера. 27](#_Toc71935386)

[Приложение А.1 Файл Main.java 27](#_Toc71935387)

[Приложение А.2 Файл ClientCommunication.java 27](#_Toc71935388)

[Приложение А.3 Файл CommunicationService.java 28](#_Toc71935389)

[Приложение А.4 Файл ServerDataconsumer.java 29](#_Toc71935390)

[Приложение А.5 Файл PrimeWindowController.java 30](#_Toc71935391)

[Приложение А.6 Файл CalculatingScheduler.java 31](#_Toc71935392)

[Приложение А.7 Файл CalculatingTask.java 31](#_Toc71935393)

[Приложение А.8 Файл Logger.java 32](#_Toc71935394)

[Приложение А.9 Файл TimeTracker.java 32](#_Toc71935395)

[Приложение А.10 Файл PrimeWindow.fxml 33](#_Toc71935396)

[Приложение Б. Листинг клиента. 34](#_Toc71935397)

[Приложение Б.1 Файл Main.java 34](#_Toc71935398)

[Приложение Б.2 Файл CommunicationService.java 34](#_Toc71935399)

[Приложение Б.3 Файл PrimeWindowController.java 35](#_Toc71935400)

[Приложение Б.4 Файл CalculatingScheduler.java 35](#_Toc71935401)

[Приложение Б.5 Файл CalcuatingTask.java 36](#_Toc71935402)

[Приложение Б.6 Файл GFG.java 37](#_Toc71935403)

[Приложение Б.7 Файл Logger.java 37](#_Toc71935404)

[Приложение Б.8 Файл PrimeWindow.java 37](#_Toc71935405)

# Введение

Сегодняшние реалии таковы, что разработка практически любого программного обеспечения требует хороших знаний параллельного и распределенного программирования. Обе эти области объединяет то, что и параллельное, и распределенное программное обеспечение состоит из нескольких процессов, которые вместе решают одну общую задачу.

Однако понятия параллельной и распределенной обработки данных не являются эквивалентными. Параллелизм подразумевает одновременное существование и выполнение задач. Распределение же обозначает территориальную удаленность процессов друг от друга. При этом задачи, связанные с распределенной обработкой данных, могут выполняться несколькими последовательными этапами в различные периоды времени. Конечно, в некоторых случаях параллельная обработка данных может быть эффективно реализована с помощью распределенного программного обеспечения, особенно при проведении большого объема вычислений, слабо связанных между собой.

Целью исследования, проводимого в рамках настоящей курсовой работы, является разработка параллельной программы для кластера облачного типа с пользовательским интерфейсом.

# Постановка задачи

1) Клиент-серверное приложение для управления вычислительным процессом должно состоять из двух частей: клиента и сервера, которые должны образовывать единую распределенную вычислительную систему облачного типа.

2) Серверное приложение должно обеспечивать одновременное подключение нескольких клиентов.

3) Клиентское приложение должно обеспечивать подключение к серверу (порт и IP сервера вводятся в окне клиента) и иметь возможность осуществить отключение от сервера в любой момент.

4) В окне серверного приложения должен отображаться прогресс решения вычислительной задачи и примерное время, оставшееся до ее решения с учетом текущей скорости вычислительного процесса, а также указываться список подключившихся клиентов и диапазон/шаг интегрирования.

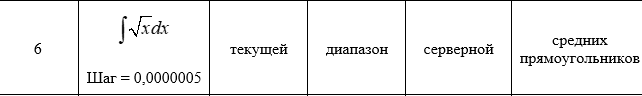
5) Для каждого клиента должна быть предусмотрена возможность указания предельно допустимого уровня использования вычислительных ресурсов (процессорного времени) в процентах.

6) Кластер должен решать задачу вычисления определенного интеграла (здесь указывается интеграл в соответствии с заданием).

7) Для интегрирования должен использоваться методом средних прямоугольников.

8) Приложения (клиент и сервер) должны обладать интуитивно понятным графическим интерфейсом.

9) Для обмена данными использовать протокол TCP/IP.



# Выбор решения

Для того чтобы лучше понять, что необходимо для реализации поставленной задачи, рассмотрим предметную область.

Исходя из требования к задаче, необходимо выбрать соответствующие инструменты для её реализации.

1. Необходимо определиться с языком программирования. Выбор был сделан в пользу Java. Java является объектно-ориентированным языком программирования, что позволяет разбить программу на классы. Это обеспечит многомодульность программы, что в свою очередь повысит наглядность и легкость восприятия кода программы. Серверное приложение предоставляет пользователю возможность вводить параметры интегрирования, максимально допустимый уровень использования вычислительных ресурсов, отображает список подключившихся клиентов, прогресс решения вычислительной задачи и примерное время, оставшееся до ее решения. Для наибольшей эффективности и корректности выполнения программы каждая функция сервера выполняет работу в отдельной нити. В Java имеется возможность относительно просто создавать дополнительные процессы и нити, что позволит реализовать параллельную обработку нескольких запросов.

2. Необходимо определить, каким образом будет осуществляться передача данных между сервером и клиентом, в данной работе на основе TCP протокола. TCP — это ориентированный на соединения протокол, предназначенный для обеспечения надежной передачи данных между процессами, выполняемыми или на одном и том же компьютере, или на разных компьютерах.

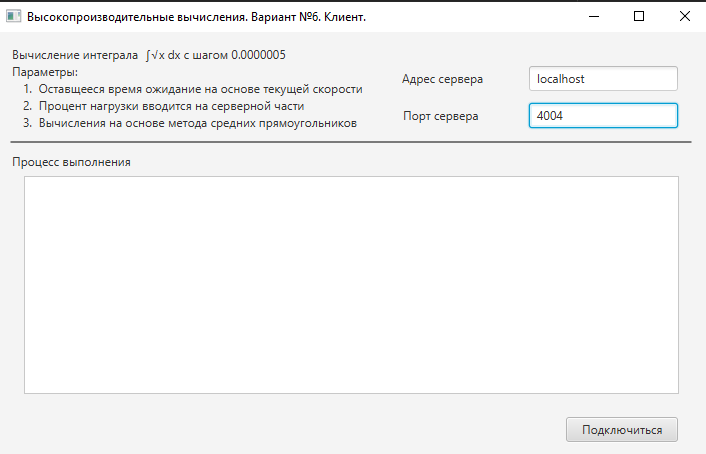
3. Для реализации графического интерфейса было решено воспользоваться редактором форм SceneBuilder. С его помощью, были заданы положение и размеры элементов графического интерфейса для ввода и вывода информации, необходимой для решения поставленной задачи. Для реализации логической части ПО были создан набор классов обеспечивающих корректную работу программ.

# 3. Описание разработки программы

## 3.1. Разработка клиентской части приложения

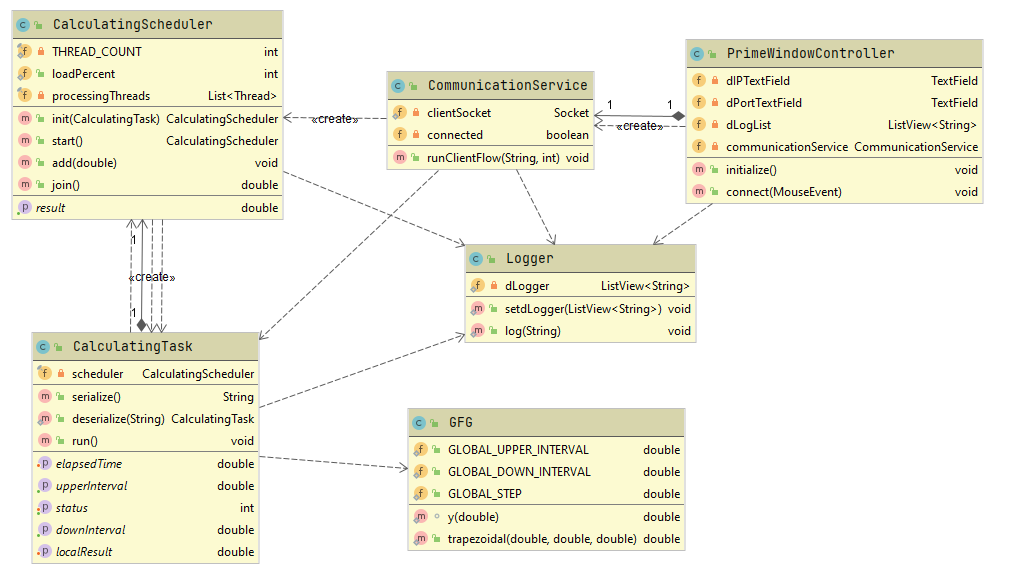
В соответствии с требованиями, клиентское приложение должно обеспечивать: приложение должно обеспечивать подключение к серверу (порт и IP сервера вводятся в окне клиента) и иметь возможность осуществить отключение от сервера в любой момент, выполнять параллельный расчет интеграла методом средних прямоугольников.

В окне процесс выполнения выводится вся системная информация в процессе выполнения, чтобы клиенту установить связь с сервером, пользователю необходимо ввести IP адрес и порт сервера, после чего нажать на кнопку «Подключиться» и дождаться подключения.



**Рисунок 1 - Интерфейс клиентской части приложения.**

При нажатии кнопки «Подключиться» происходит попытка подключение к серверному приложению. Если попытка удачная, то сервер регистрирует этого клиента. Клиент, получая задачу, разбивает её на подзадачи, количество которых равно числу логических ядер в системе, и каждая подзадача выполняется на своем логическом ядре.

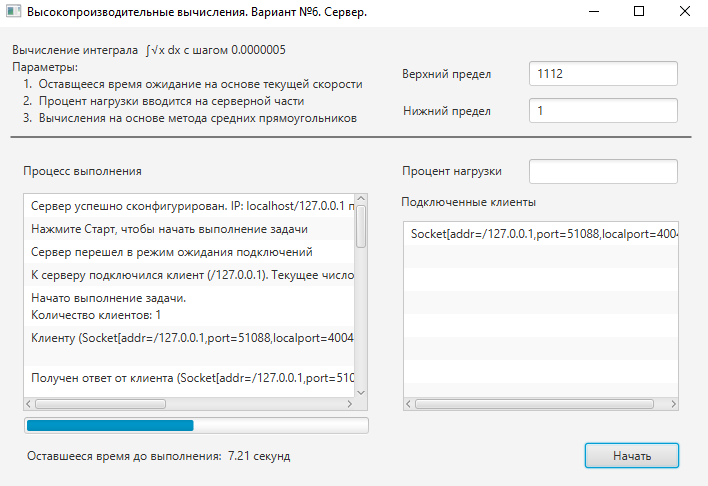


**Рисунок 2 - UML-диаграмма классов клиента.**

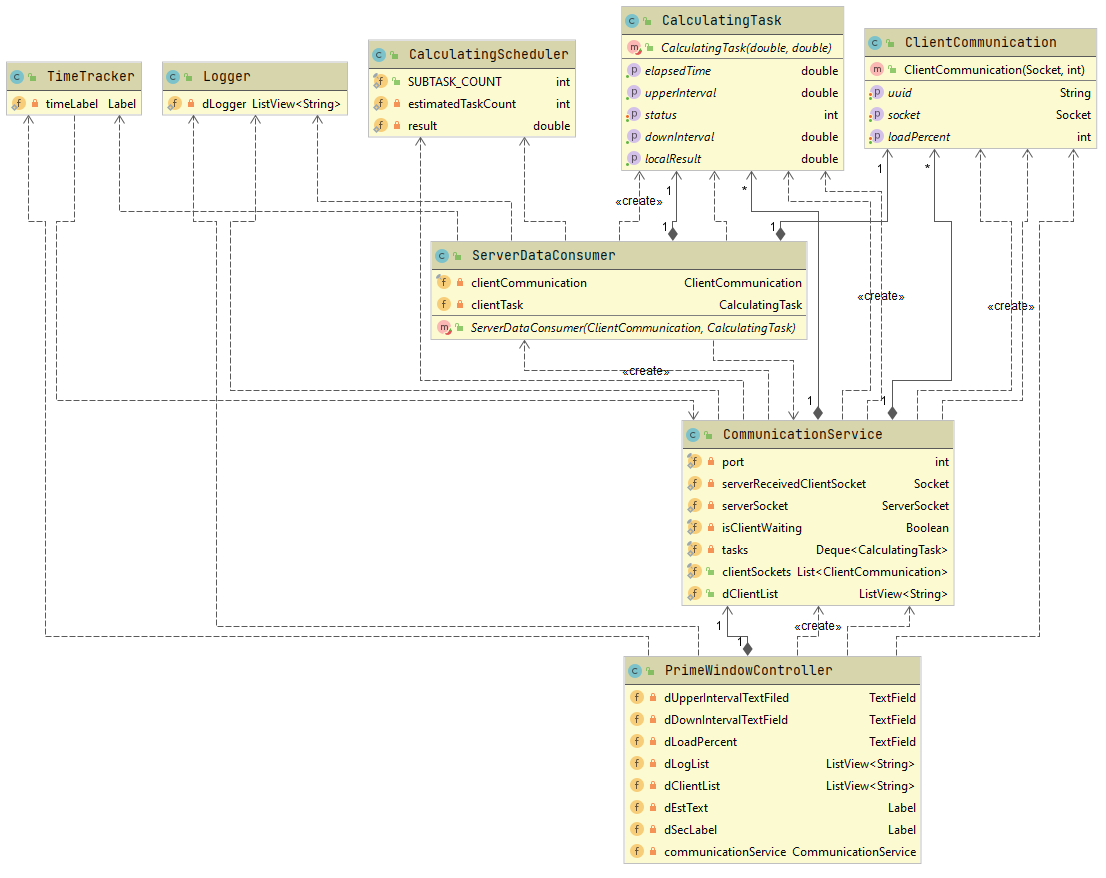
## 3.2. Разработка серверной части приложения

Сервер должен передавать сообщения всем клиентам и принимать от него сообщения параллельно своей работе. Для этого необходимо хранить всех подключенных клиентов в памяти. В качестве хранения всех клиентов использовался отдельный графический компонент.

После ввода данных и нажатие на кнопку «Начать», сервер формирует очередь подзадач и начинает их раздачу клиентам. Если клиентов нет, то сервер ждет подключения клиентов. Если клиенты есть, то им назначается подзадача. Так продолжается пока задача не будет решена. Если клиент отключился во время работы, то он удаляется из списка, отключается от сервера, а подзадача возвращается в очередь, тем самым обеспечивается надежность при решении задачи. Во время работы отображается прогресс решения задачи и примерное время, оставшееся до её завершения.



**Рисунок 3 - Интерфейс серверной части приложения.**



**Рисунок 4 - Интерфейс клиентского приложения.**

# 4. Разработка диаграмм UML

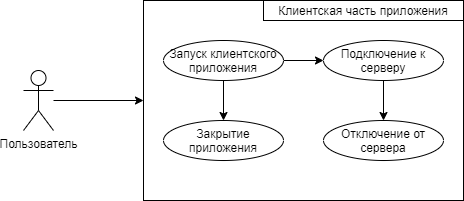
С целью упрощения и визуализации процесса проектирования программы были разработаны следующие диаграммы UML:

1. диаграмма вариантов использования;
2. диаграмма развёртывания;
3. диаграмма последовательности;
4. диаграмма деятельности;
5. диаграмма классов.

## 4.1. Разработка диаграммы вариантов использования

В обоих приложениях, как в серверном, так и в клиентском, пользователь является единственным участником системы.

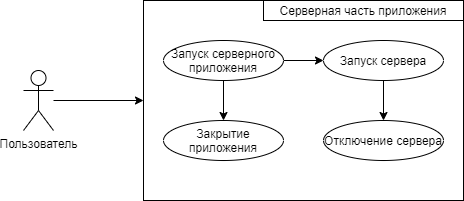
Клиент может в любой момент осуществить попытку подключиться к серверу, а также в любой момент разорвать с ним соединение. Диаграмма вариантов использования клиентского приложения показана на рисунке 5:



**Рисунок 5 – UML-диаграмма вариантов использования**

**клиентской части приложения**

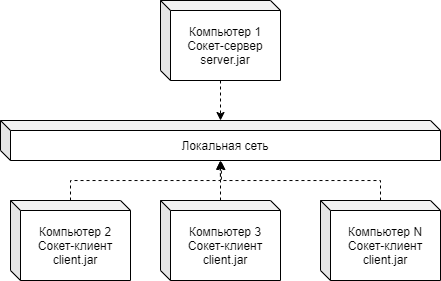
Аналогично клиенту, сервер так же может быть запущен и остановлен в любой момент. Диаграмма вариантов использования серверного приложения показана на рисунке 6.



**Рисунок 6 – UML-диаграмма вариантов использования серверного приложения**

## 4.2. Разработка диаграммы развертывания

В связи с требованиями к программному обеспечению, была разработана диаграмма развертывания, показанная на рисунке 7.

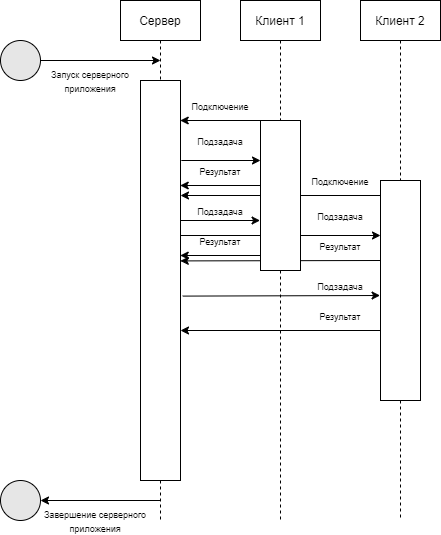


**Рисунок 7 – UML-диаграмма развёртывания**

Компьютеры с клиентскими сокетами подключаются к компьютеру с серверным сокетом по локальный вычислительной сети. Также возможна работа, когда Компьютер 1 может иметь как серверный, так и клиентский сокет. Таким образом развертывание возможно на одном компьютере, а членом локальной вычислительной сети в этом случае будет только один компьютер, посылающий сообщения себе по адресу 127.0.0.1.

## 4.3. Разработка диаграммы последовательности

На рисунке 8 приведён пример работы серверного и клиентских приложений в виде UML-диаграммы последовательности.



**Рисунок 8 – UML-диаграмма последовательности**

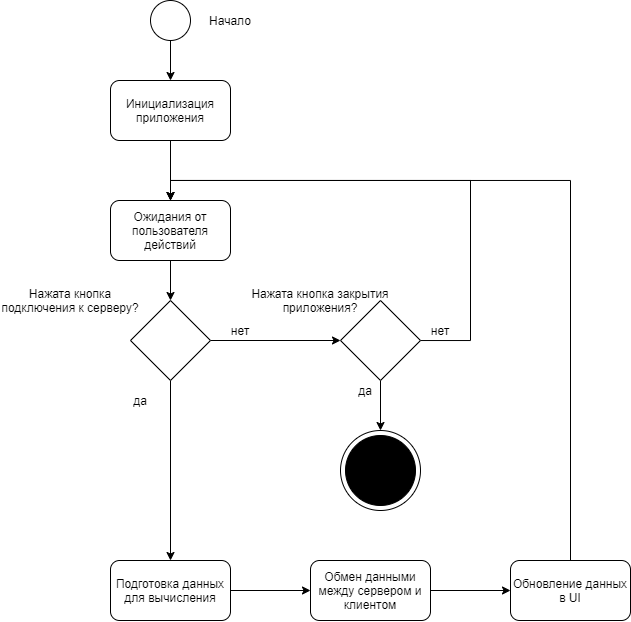
## 4.4. Разработка диаграммы деятельности

Разработка программы началась с разработки графического интерфейса пользователя. Это позволило к моменту написания рабочей логики приложения управлять состояниями программы без дополнительных средств отладки, параметров командной строки и др. Диаграмма активностей для серверного (Рисунок 9) и клиентского (Рисунок 10) приложения представлена ниже.



**Рисунок 9 – UML-диаграмма деятельности серверной части приложения**

Рабочая логика программа была написана в виде совокупности классов, обменивающихся информацией через сигналы и слоты. Был написан класс для общения с UI-частью программы (PrimeController), классы для обеспечения связи с клиентами (CommunicationService), класс распределения задач (CalculatingScheduler). На клиентской стороне также был создан класс PrimeController для управления графическими компонентами.



**Рисунок 10 – UML-диаграмма деятельности клиентской части приложения**

## 4.5. Разработка диаграммы классов

Диаграммы классов были представлены на рисунках 2 и 4, для клиентской и серверной части соответственно.

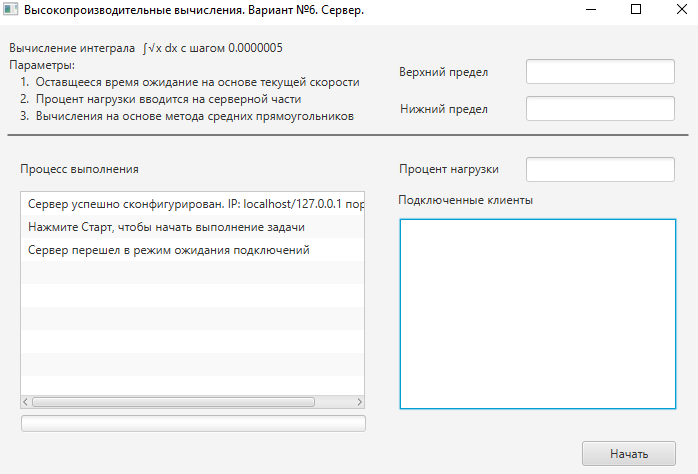
# 5. Отладка и тестирование

В качестве среды разработки выступает программный продукт InteliJ IDEA. Программа предоставляет все средства необходимые при разработке и отладке разрабатываемых модулей и программ. Для отладки использовались такие инструменты как точка остановки, выполнение кода по шагам, анализ содержимого локальных и глобальных (для определенного класса) переменных.

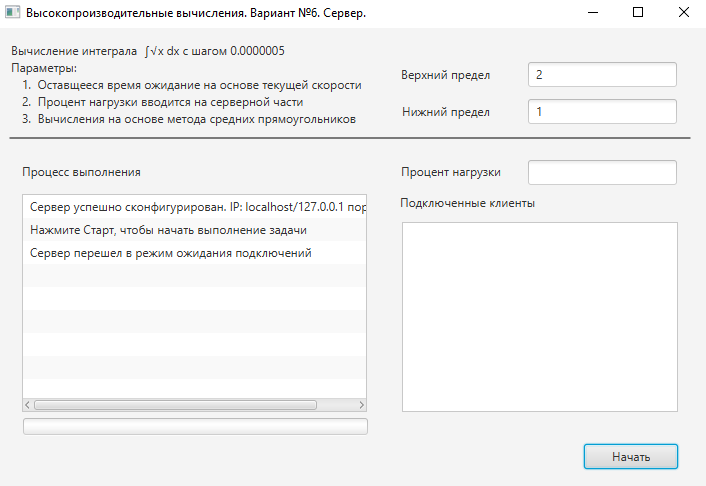
Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с взаимодействием сервера и клиента и реагированием сервера на подключение и отключение клиентов.

# 6. Руководство пользователя

При запуске серверной части приложение перед пользователем откроется начальное окно, в котором необходимо ввести такие начальные данные как: верхний предел интегрирования и нижний. В области Процесс выполнения можно увидеть сообщения программы, в которых содержится информация о текущем адресе приложения и порту, который будет использоваться для обмена сообщениями с клиентами.

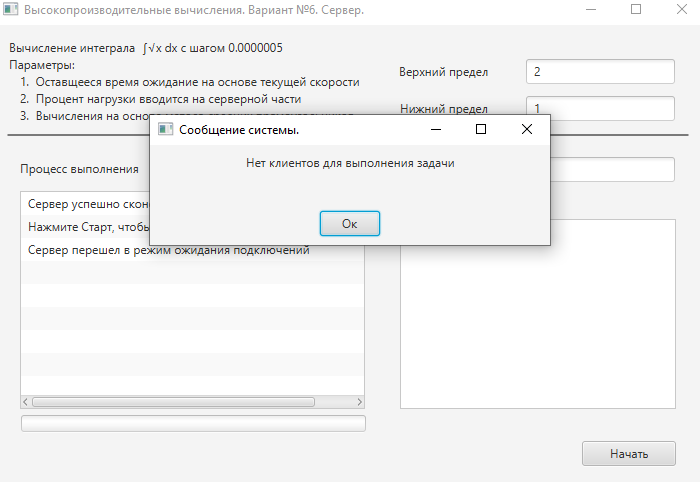


**Рисунок 11 – Начальное окно серверной части приложения**



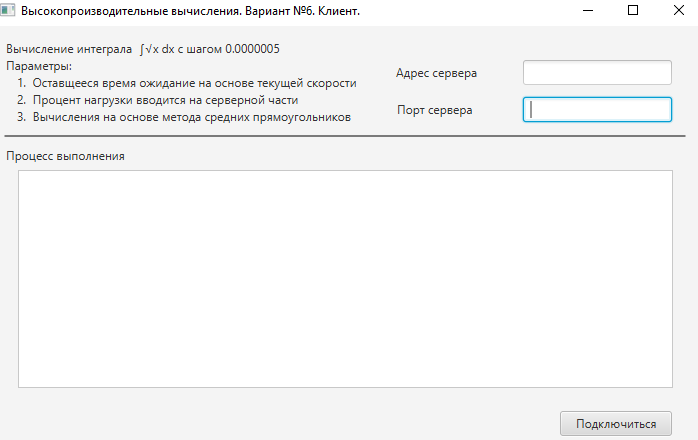
**Рисунок 12 – Инициализация данных для серверной части приложения**

Если пользователь попытается начать выполнение задачи без подключенных, откроется модальное окно, сообщение в котором будет говорить об этой исключительной ситуации. Рисунок 13 иллюстрирует эту ситуацию.



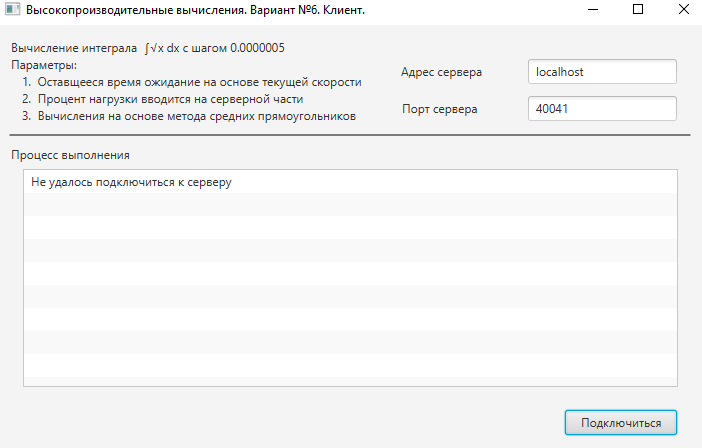
**Рисунок 13 – Обработка исключений**

При запуске клиентской части приложение перед пользователем откроется начальное окно, в котором необходимо ввести такие начальные данные как: адрес сервера и его порт. В области Процесс выполнения можно будет увидеть сообщения программы о том, какие задачи она выполняет в данный момент.



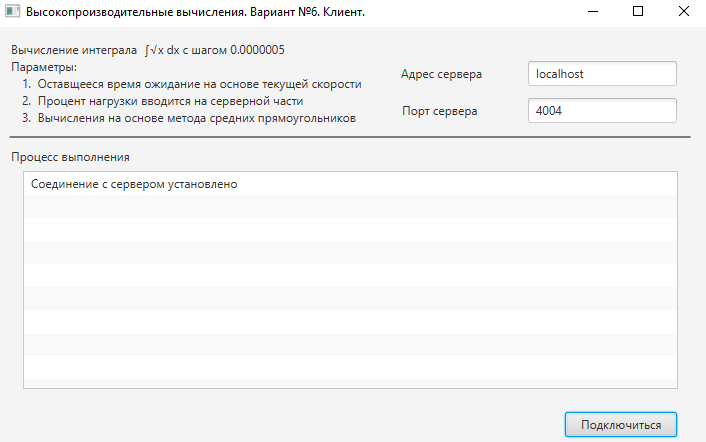
**Рисунок 14 – Начальное окно клиентской части приложения**

В процессе подключения могут возникнуть разного рода ошибки – от опечатки в адресе сервера, до отсутствия подключения к удаленному компьютеру. При их возникновении пользователю будет выдано соответствующее сообщение в области Процесс выполнения.



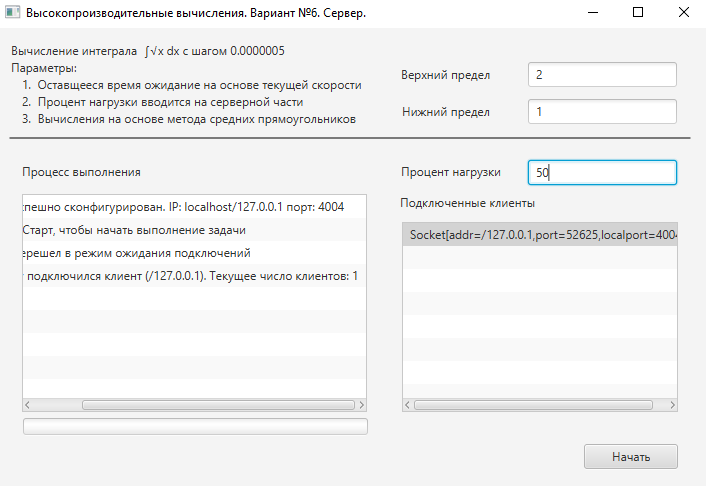
**Рисунок 14 – Обработка исключений**

При успешном подключении к серверу выдается аналогичное сообщение.



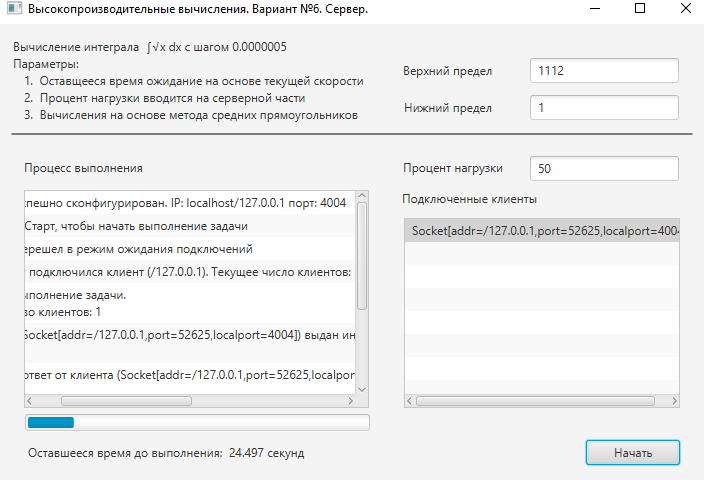
**Рисунок 15 – Подключение клиента к серверу**

При успешном подключении клиента к серверу, в окне последнего в области Подключенные клиенты появится новая запись с адресом клиента. При клике по этой записи в поле Процент нагрузки будет выведено значение, которое соответствует выбранной нагрузке на данного клиента, по умолчанию это значение равно 50%.



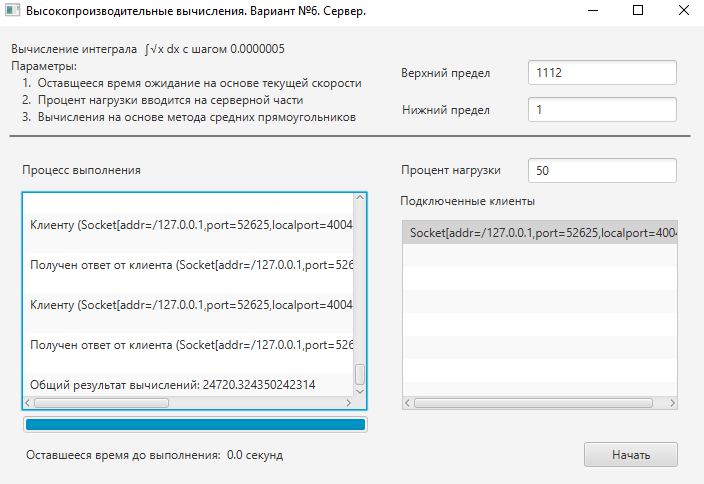
**Рисунок 16 – Просмотр списка клиентов**

Для того чтобы начать выполнение задачи пользователю необходимо нажать кнопку Начать, в области Процесс выполнения будут появляться сообщения, говорящие о процессе выполнения задачи, ниже будет представлен индикатор прогресса и текстовое сообщение с примерным временем ожидания.



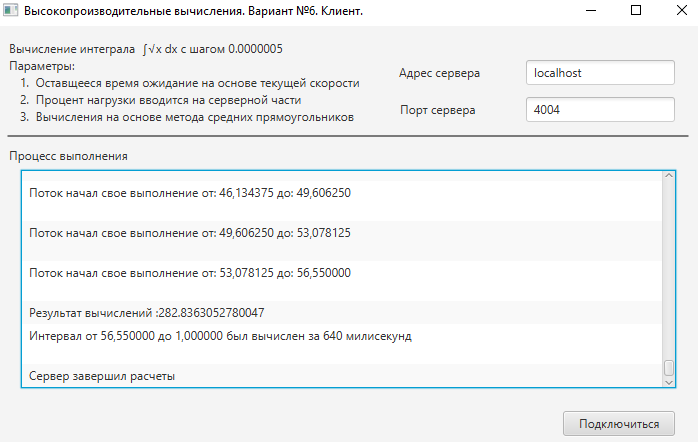
**Рисунок 17 – Процесс выполнения задачи**

По окончании выполнения задачи, индикатор будет полностью заполнен, результат выполнения задачи можно будет найти в последнем сообщении логов.

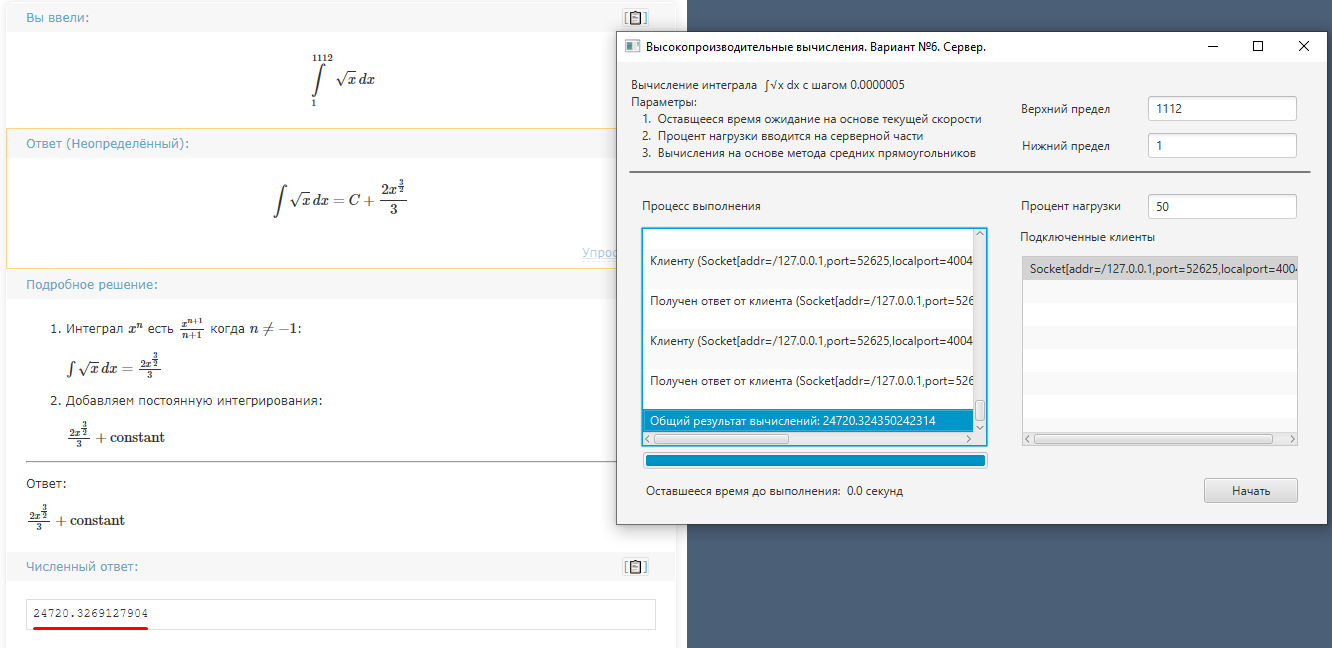


**Рисунок 17 – Завершение выполнения задачи на серверной стороне**

На клиентской стороне также можно просмотреть процесс выполнения задачи и сообщения о том, что расчеты были завершены.



**Рисунок 18 – Завершение выполнения задачи на клиентской стороне**



**Рисунок 19 – Сравнение результатов**

# Заключение

При выполнении данной курсовой работы было разработано клиент-серверное приложение, позволяющее производить вычисления в рамках локальной вычислительной сети. Программа носит практический характер и предназначена для одновременного распределенного решения единой задачи несколькими компьютерами.

Были освоены принципы сетевого взаимодействия, принципы работы с сокетами и протоколом TCP/IP.

В дальнейшем программу можно улучшить, дав пользователю самостоятельно выбрать подынтегральную функцию, дополнительно оптимизировав алгоритм работы как серверного, так и клиентского приложения.

# Список используемых источников

1. Рихтер, Джеффри CLR via. Программирование на платформе Java / Джеффри Рихтер. - М.: Питер, 2013. - 928 c.

2. Вагнер, Билл Java Эффективное программирование / Билл Вагнер. - М.: ЛОРИ, 2013. - 320 c.

3. Fall K. R., Stevens W. R. TCP/IP illustrated, volume 1: The protocols. – addison-Wesley, 2011.

4. Hejlsberg A., Wiltamuth S., Golde P. Java language specification. – Addison-Wesley Longman Publishing Co., Inc., 2003

# Приложение А. Листинг сервера.

## Приложение А.1 Файл Main.java

public class Main extends Application {  
 public static Stage *mPrimaryStage*;  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
  
 @Override  
 public void start(Stage primaryStage) throws Exception {  
 *mPrimaryStage* = primaryStage;  
 primaryStage.setOnCloseRequest(event -> {  
 System.*exit*(1);  
 });  
 *mPrimaryStage*.setTitle("Высокопроизводительные вычисления. Вариант №6. Сервер.");  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(getClass().getResource("/PrimeWindowFXML.fxml"));  
 Parent root = loader.load();  
 Scene scene = new Scene(root);  
 primaryStage.setScene(scene);  
 primaryStage.show();  
 }  
}

## Приложение А.2 Файл ClientCommunication.java

@Data  
public class ClientCommunication {  
  
 private Socket socket;  
 private int loadPercent;  
 private String uuid = UUID.*randomUUID*().toString();  
  
 public ClientCommunication(Socket socket, int loadPercent) {  
 this.socket = socket;  
 this.loadPercent = loadPercent;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 ClientCommunication that = (ClientCommunication) o;  
 return Objects.*equals*(uuid, that.uuid);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(uuid);  
 }  
}

## Приложение А.3 Файл CommunicationService.java

public class CommunicationService {  
  
 private static final int *port* = 4004;  
 private static Socket *serverReceivedClientSocket*;  
 private static ServerSocket *serverSocket*;  
 private static final Boolean *isClientWaiting* = true;  
  
 private static final Deque<CalculatingTask> *tasks* = new ArrayDeque();  
 public static final List<ClientCommunication> *clientSockets* = new ArrayList<>();  
 public static ListView<String> *dClientList*;  
  
 public synchronized static int getTaskCount(){  
 return *tasks*.size();  
 }  
  
 public synchronized static void pushTask(CalculatingTask task){  
 *tasks*.push(task);  
 }  
  
 public synchronized static CalculatingTask getTask() {  
 return *tasks*.pollFirst();  
 }  
  
 public synchronized static void removeClient(ClientCommunication clientCommunication){  
 var clientIndex = *clientSockets*.indexOf(clientCommunication);  
 *clientSockets*.remove(clientCommunication);  
 Platform.*runLater*(()->{  
 *dClientList*.getItems().remove(clientIndex);  
 *dClientList*.refresh();  
 });  
 Logger.*log*(String.*format*("От сервера отключился клиент (%s). Текущее число клиентов: %d", clientCommunication.getSocket(), *clientSockets*.size()));  
 }  
  
 public synchronized static boolean hasTask() {  
 return !*tasks*.isEmpty();  
 }  
  
 public void start(double upperInterval, double downInterval){  
 new Thread(() -> {  
 Logger.*log*("Начато выполнение задачи.\nКоличество клиентов: " + *clientSockets*.size());  
 var calculatedPeriod = (upperInterval - downInterval) / CalculatingScheduler.*SUBTASK\_COUNT*;  
 var taskInterval = downInterval;  
 CalculatingScheduler.*reset*();  
 *tasks*.clear();  
 CalculatingScheduler.*setEstimatedTaskCount*(CalculatingScheduler.*SUBTASK\_COUNT*);  
 for (var i = 0; i < CalculatingScheduler.*SUBTASK\_COUNT*; i++) {  
 *tasks*.addFirst(new CalculatingTask(taskInterval + calculatedPeriod, taskInterval));  
 taskInterval = taskInterval + calculatedPeriod;  
 }  
 *clientSockets*.forEach(client -> {  
 new Thread(new ServerDataConsumer(client, null)).start();  
 });  
 }).start();  
 }  
  
 public void runServiceFlow() throws IOException {  
 *serverSocket* = new ServerSocket(*port*);  
 Logger.*log*("Сервер успешно сконфигурирован. IP: " + InetAddress.*getByName*("localhost") + " порт: " + *port*);  
 Logger.*log*("Нажмите Старт, чтобы начать выполнение задачи");  
 Logger.*log*("Сервер перешел в режим ожидания подключений");  
 while (*isClientWaiting*) {  
 *serverReceivedClientSocket* = *serverSocket*.accept();  
 if (!*clientSockets*.stream().anyMatch((client -> client.getSocket().getLocalAddress().equals(*serverReceivedClientSocket*.getLocalAddress())))) {  
 *clientSockets*.add(new ClientCommunication(*serverReceivedClientSocket*, 50));  
 Platform.*runLater*(()->{  
 *dClientList*.getItems().add(*serverReceivedClientSocket*.toString());  
 *dClientList*.refresh();  
 });  
 Logger.*log*("К серверу подключился клиент (" + *serverReceivedClientSocket*.getInetAddress() + "). Текущее число клиентов: " + *clientSockets*.size());  
 }  
 }  
 }  
}

## Приложение А.4 Файл ServerDataconsumer.java

@AllArgsConstructor  
public class ServerDataConsumer implements Runnable {  
  
 private final ClientCommunication clientCommunication;  
 private CalculatingTask clientTask;  
  
 @Override  
 @SneakyThrows  
 public void run() {  
 try {  
 BufferedReader br = new BufferedReader(new InputStreamReader(clientCommunication.getSocket().getInputStream()));  
 PrintWriter pw = new PrintWriter(clientCommunication.getSocket().getOutputStream(), true);  
 while (CommunicationService.*hasTask*()){  
 clientTask = CommunicationService.*getTask*();  
 Logger.*log*(String.*format*("Клиенту (%s) выдан интервал от %f до %f%n", clientCommunication.getSocket(), clientTask.getUpperInterval(), clientTask.getDownInterval()));  
 pw.println(clientTask.serialize());  
 var clientResponse = CalculatingTask.*deserialize*(br.readLine());  
 Logger.*log*(String.*format*("Получен ответ от клиента (%s) для интервала %f до %f : %f %n", clientCommunication.getSocket(), clientTask.getUpperInterval(), clientTask.getDownInterval(), clientResponse.getLocalResult()));  
 TimeTracker.*update*(clientResponse.getElapsedTime());  
 CalculatingScheduler.*add*(clientResponse.getLocalResult());  
 }  
 pw.println(new CalculatingTask(0, 0).serialize(1));  
 Logger.*log*("Общий результат вычислений: " + CalculatingScheduler.*getResult*());  
 return;  
 } catch (Exception e){  
 CommunicationService.*pushTask*(clientTask);  
 CommunicationService.*removeClient*(clientCommunication);  
 Logger.*log*("Соединение с клиентом (" + clientCommunication.getSocket().getInetAddress().getHostAddress() + ") закрыто, его задача возвращена в очередь");  
 clientCommunication.getSocket().close();  
 }  
 }  
}

## Приложение А.5 Файл PrimeWindowController.java

public class PrimeWindowController {  
  
 @FXML  
 private TextField dUpperIntervalTextFiled;  
  
 @FXML  
 private TextField dDownIntervalTextField;  
  
 @FXML  
 private TextField dLoadPercent;  
  
 @FXML  
 private ListView<String> dLogList;  
  
 @FXML  
 private ListView<String> dClientList;  
  
 @FXML  
 private Label dEstText;  
 @FXML  
 private Label dSecLabel;  
  
 @FXML  
 private ProgressBar dProgressBar;  
  
 private CommunicationService communicationService;  
  
 @SneakyThrows  
 public void initialize() {  
 Logger.*setdLogger*(dLogList);  
 TimeTracker.*setTimeLabel*(dSecLabel);  
 TimeTracker.*setProgressBar*(dProgressBar);  
 dLoadPercent.focusedProperty().addListener((observable, oldValue, newValue) -> {  
 if(!newValue){  
 changeLoadPercent(null);  
 }  
 });  
 dUpperIntervalTextFiled.setText("2");  
 dDownIntervalTextField.setText("1");  
 new Thread(() -> {  
 try {  
 communicationService = new CommunicationService();  
 CommunicationService.*dClientList* = dClientList;  
 communicationService.runServiceFlow();  
 } catch (IOException ignore) {  
 }  
 }).start();  
 }  
  
 public void changeLoadPercent(KeyEvent keyEvent) {  
 var selectedIndex = dClientList.getSelectionModel().getSelectedIndex();  
 if (selectedIndex != -1) {  
 var client = CommunicationService.*clientSockets*.get(selectedIndex);  
 if(dLoadPercent.getText().equals("")){  
 client.setLoadPercent(1);  
 } else {  
 client.setLoadPercent(Integer.*parseInt*(dLoadPercent.getText()));  
 }  
 }  
 }  
  
 public void start(MouseEvent mouseEvent) {  
 dEstText.setVisible(true);  
 communicationService.start(Double.*parseDouble*(dUpperIntervalTextFiled.getText()), Double.*parseDouble*(dDownIntervalTextField.getText()));  
 }  
  
 public void selectClient(MouseEvent mouseEvent) {  
 var selectedIndex = dClientList.getSelectionModel().getSelectedIndex();  
 if (Objects.*nonNull*(selectedIndex) && selectedIndex != -1) {  
 var client = CommunicationService.*clientSockets*.get(selectedIndex);  
 dLoadPercent.setText(String.*valueOf*(client.getLoadPercent()));  
 }  
 }  
}

## Приложение А.6 Файл CalculatingScheduler.java

public class CalculatingScheduler {  
  
 public static final int *SUBTASK\_COUNT* = 20;  
  
 @Setter  
 private static int *estimatedTaskCount* = 0;  
  
 @Getter  
 private static double *result*;  
  
 public static synchronized void reset(){  
 *result* = 0;  
 }  
  
 public static synchronized int decAndGet(){  
 return --*estimatedTaskCount*;  
 }  
  
 public synchronized static void add(double localResult) {  
 *result* += localResult;  
 }  
}

## Приложение А.7 Файл CalculatingTask.java

@RequiredArgsConstructor  
public class CalculatingTask {  
  
 @Setter  
 @Getter  
 private int status;  
  
 @NonNull  
 @Getter  
 private final double upperInterval;  
  
 @NonNull  
 @Getter  
 private final double downInterval;  
  
 @Getter  
 private double elapsedTime;  
  
 @Getter  
 private double localResult;  
  
 public String serialize(int status){  
 this.setStatus(status);  
 return new Gson().toJson(this);  
 }  
  
 public String serialize(){  
 return new Gson().toJson(this);  
 }  
  
 public static CalculatingTask deserialize(String data){  
 return new Gson().fromJson(data, CalculatingTask.class);  
 }  
}

## Приложение А.8 Файл Logger.java

public class Logger {  
 private static ListView<String> *dLogger*;  
  
 public static void setdLogger(ListView<String> dLogger) {  
 Logger.*dLogger* = dLogger;  
 }  
  
 public static void log(String text) {  
 Platform.*runLater*(() -> {  
 *dLogger*.getItems().add(text);  
 *dLogger*.refresh();  
 });  
 }  
}

## Приложение А.9 Файл TimeTracker.java

public class TimeTracker {  
 private static Label *timeLabel*;  
 private static ProgressBar *progressBar*;  
  
 public static void setTimeLabel(Label timeLabel) {  
 TimeTracker.*timeLabel* = timeLabel;  
 }  
  
 public static void setProgressBar(ProgressBar progressBar) {  
 TimeTracker.*progressBar* = progressBar;  
 }  
  
 public synchronized static void update(double elapsedTime){  
 Platform.*runLater*(()->{  
 *timeLabel*.setVisible(true);  
 *timeLabel*.setText((elapsedTime \* CommunicationService.*getTaskCount*()) / 1000 + " секунд");  
 *progressBar*.setProgress(1.0 - ((double) CommunicationService.*getTaskCount*() / CalculatingScheduler.*SUBTASK\_COUNT*));  
 });  
 }  
}

## Приложение А.10 Файл PrimeWindow.fxml

*<?*xml version="1.0" encoding="UTF-8"*?>  
  
<?*import javafx.scene.control.\**?>  
<?*import javafx.scene.effect.\**?>  
<?*import javafx.scene.layout.\**?>  
<?*import javafx.scene.shape.\**?>*<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="462.0" prefWidth="710.0" xmlns="http://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/fxml/1" fx:controller="Controllers.PrimeWindowController">  
 <children>  
 <Label layoutX="14.0" layoutY="14.0" prefHeight="17.0" prefWidth="345.0" text="Вычисление интеграла ∫√x dx с шагом 0.0000005" />  
 <Label layoutX="14.0" layoutY="31.0" prefHeight="17.0" prefWidth="276.0" text="Параметры:" />  
 <Label layoutX="25.0" layoutY="48.0" prefHeight="17.0" prefWidth="351.0" text="1. Оставщееся время ожидание на основе текущей скорости" />  
 <Label layoutX="25.0" layoutY="65.0" prefHeight="17.0" prefWidth="345.0" text="2. Процент нагрузки вводится на серверной части" />  
 <Label layoutX="25.0" layoutY="82.0" prefHeight="17.0" prefWidth="345.0" text="3. Вычисления на основе метода средних прямоугольников" />  
 <Line endX="580.0" layoutX="113.0" layoutY="110.0" startX="-100.0" />  
 <Label layoutX="404.0" layoutY="38.0" prefHeight="17.0" prefWidth="118.0" text="Верхний предел" />  
 <TextField id="dUpperIntervalTextFiled" fx:id="dUpperIntervalTextFiled" layoutX="531.0" layoutY="34.0" />  
 <Label layoutX="405.0" layoutY="75.0" text="Нижний предел" />  
 <TextField id="dDownIntervalTextField" fx:id="dDownIntervalTextField" layoutX="531.0" layoutY="71.0" />  
 <TextField id="dLoadPercent" fx:id="dLoadPercent" layoutX="531.0" layoutY="132.0" onKeyPressed="#changeLoadPercent" />  
 <Label layoutX="404.0" layoutY="135.0" prefHeight="17.0" prefWidth="128.0" text="Процент нагрузки" />  
 <ListView id="dClientsList" fx:id="dClientList" layoutX="405.0" layoutY="194.0" onMouseClicked="#selectClient" prefHeight="190.0" prefWidth="276.0" />  
 <Label layoutX="403.0" layoutY="166.0" text="Подключенные клиенты" />  
 <Label layoutX="25.0" layoutY="135.0" text="Процесс выполнения" />  
 <ListView id="dLogList" fx:id="dLogList" layoutX="25.0" layoutY="166.0" prefHeight="218.0" prefWidth="345.0">  
 <effect>  
 <Blend />  
 </effect></ListView>  
 <Button fx:id="dStartButton" layoutX="587.0" layoutY="416.0" mnemonicParsing="false" onMouseClicked="#start" prefHeight="25.0" prefWidth="94.0" text="Начать" />  
 <Label fx:id="dEstText" layoutX="29.0" layoutY="416.0" prefHeight="25.0" prefWidth="201.0" text="Оставшееся время до выполнения:" />  
 <Label fx:id="dSecLabel" layoutX="230.0" layoutY="416.0" prefHeight="25.0" prefWidth="136.0" text="0 сек" />  
 <ProgressBar fx:id="dProgressBar" layoutX="26.0" layoutY="390.0" prefHeight="18.0" prefWidth="345.0" progress="0.0" />  
 </children>  
</AnchorPane>

# Приложение Б. Листинг клиента.

## Приложение Б.1 Файл Main.java

public class Main extends Application {  
 public static Stage *mPrimaryStage*;  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
  
 @Override  
 public void start(Stage primaryStage) throws Exception {  
 *mPrimaryStage* = primaryStage;  
 primaryStage.setOnCloseRequest(event -> {  
 System.*exit*(1);  
 });  
 *mPrimaryStage*.setTitle("Высокопроизводительные вычисления. Вариант №6. Клиент.");  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(getClass().getResource("/PrimeWindowFXML.fxml"));  
 Parent root = loader.load();  
 Scene scene = new Scene(root);  
 primaryStage.setScene(scene);  
 primaryStage.show();  
 }  
}

## Приложение Б.2 Файл CommunicationService.java

public class CommunicationService {  
  
 private static Socket *clientSocket*;  
 private boolean connected = false;  
  
 @SneakyThrows  
 public void runClientFlow(String ip, int port) {  
 try {  
 *clientSocket* = new Socket(ip, port);  
 BufferedReader br = new BufferedReader(new InputStreamReader(*clientSocket*.getInputStream()));  
 PrintWriter pw = new PrintWriter(*clientSocket*.getOutputStream(), true);  
 Logger.*log*("Соединение с сервером установлено");  
 connected = true;  
 while (connected) {  
 if (br.ready()) {  
 var task = CalculatingTask.*deserialize*(br.readLine());  
 if(task.getStatus() == 1) {  
 Logger.*log*("Сервер завершил расчеты");  
 continue;  
 }  
 Logger.*log*(String.*format*("Поступило задание для вычисление интеграла от %f до %f%n", task.getUpperInterval(), task.getDownInterval()));  
 var timeStart = System.*currentTimeMillis*();  
 var result =  
 new CalculatingScheduler()  
 .init(task)  
 .start()  
 .join();  
 task.setLocalResult(result);  
 var elapsedTime = System.*currentTimeMillis*() - timeStart;  
 task.setElapsedTime(elapsedTime);  
 Logger.*log*(String.*format*("Интервал от %f до %f был вычислен за %d милисекунд%n", task.getUpperInterval(), task.getDownInterval(), elapsedTime));  
 pw.println(task.serialize());  
 }  
 }  
 } finally {  
 Logger.*log*("Осуществляется закрытие соединения с сервором");  
 *clientSocket*.close();  
 Logger.*log*("Соединение успешно закрыто");  
 }  
 }  
}

## Приложение Б.3 Файл PrimeWindowController.java

public class PrimeWindowController {  
 @FXML  
 private TextField dIPTextField;  
  
 @FXML  
 private TextField dPortTextField;  
  
 @FXML  
 private ListView<String> dLogList;  
  
 private CommunicationService communicationService;  
  
 public void initialize() {  
 Logger.*setdLogger*(dLogList);  
 dIPTextField.setText("localhost");  
 dPortTextField.setText("4004");  
 }  
  
 public void connect(MouseEvent mouseEvent) {  
 communicationService = new CommunicationService();  
 new Thread(()->{  
 communicationService.runClientFlow(dIPTextField.getText(), Integer.*parseInt*(dPortTextField.getText()));  
 }).start();  
 }  
}

## Приложение Б.4 Файл CalculatingScheduler.java

public class CalculatingScheduler {  
  
 private static final int *THREAD\_COUNT* = Runtime.*getRuntime*().availableProcessors();  
 public static int *loadPercent* = 20;  
  
 @Getter  
 private double result = 0;  
  
 private final List<Thread> processingThreads = new ArrayList<>();  
  
 public CalculatingScheduler init(CalculatingTask task) {  
 Logger.*log*("Количество потоков: " + *THREAD\_COUNT*);  
 double calculatedPeriod = (task.getUpperInterval() - task.getDownInterval()) / *THREAD\_COUNT*;  
 double taskInterval = task.getDownInterval();  
 for (var i = 0; i < *THREAD\_COUNT*; i++) {  
 processingThreads.add(new Thread(new CalculatingTask(taskInterval, taskInterval + calculatedPeriod, this)));  
 taskInterval = taskInterval + calculatedPeriod;  
 }  
 return this;  
 }  
  
 public CalculatingScheduler start() {  
 processingThreads.forEach(Thread::start);  
 return this;  
 }  
  
 public synchronized void add(double localResult) {  
 result += localResult;  
 }  
  
 public double join() {  
 processingThreads.forEach(processingThread -> {  
 try {  
 processingThread.join();  
 } catch (InterruptedException ignore) {  
 }  
 });  
 Logger.*log*("Результат вычислений :" + result);  
 return result;  
 }  
}

## Приложение Б.5 Файл CalcuatingTask.java

@RequiredArgsConstructor  
public class CalculatingTask implements Runnable{  
  
 @Setter  
 @Getter  
 private int status;  
  
 @NonNull  
 @Getter  
 private final double upperInterval;  
  
 @NonNull  
 @Getter  
 private final double downInterval;  
  
 @Setter  
 private double elapsedTime;  
  
 @NonNull  
 private final CalculatingScheduler scheduler;  
  
 @Setter  
 private double localResult;  
  
 public String serialize(){  
 return new Gson().toJson(this);  
 }  
  
 public static CalculatingTask deserialize(String data){  
 return new Gson().fromJson(data, CalculatingTask.class);  
 }  
  
 @SneakyThrows  
 @Override  
 public void run() {  
 Logger.*log*(String.*format*("Поток начал свое выполнение от: %f до: %f%n", upperInterval, downInterval));  
 var timeStart = System.*currentTimeMillis*();  
 localResult = GFG.*trapezoidal*(upperInterval, downInterval, GFG.*GLOBAL\_STEP*);  
 var timeEnd = System.*currentTimeMillis*();  
 var workTime = timeEnd - timeStart;  
 Thread.*sleep*((workTime \* CalculatingScheduler.*loadPercent*) - workTime);  
 scheduler.add(localResult);  
 }  
}

## Приложение Б.6 Файл GFG.java

public class GFG {  
  
 static double y(double x){  
 return Math.*sqrt*(x);  
 }  
  
 public static double trapezoidal(double a, double b, double h){  
 int n = (int)((b - a) / h);  
 double s = *y*(a) + *y*(b);  
  
 for (int i = 1; i < n; i++)  
 s += 2 \* *y*( a + i \* h);  
  
 return (h / 2) \* s;  
 }  
}

## Приложение Б.7 Файл Logger.java

public class Logger {  
 private static ListView<String> *dLogger*;  
  
 public static void setdLogger(ListView<String> dLogger) {  
 Logger.*dLogger* = dLogger;  
 }  
  
 public static void log(String text) {  
 Platform.*runLater*(()->{  
 *dLogger*.getItems().add(text);  
 *dLogger*.refresh();  
 });  
 }  
}

## Приложение Б.8 Файл PrimeWindow.java

*<?*xml version="1.0" encoding="UTF-8"*?>  
  
<?*import javafx.scene.control.\**?>  
<?*import javafx.scene.layout.\**?>  
<?*import javafx.scene.shape.\**?>*<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="424.0" prefWidth="710.0" xmlns="http://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/fxml/1" fx:controller="Controllers.PrimeWindowController">  
 <children>  
 <Label layoutX="14.0" layoutY="14.0" prefHeight="17.0" prefWidth="345.0" text="Вычисление интеграла ∫√x dx с шагом 0.0000005" />  
 <Label layoutX="14.0" layoutY="31.0" prefHeight="17.0" prefWidth="276.0" text="Параметры:" />  
 <Label layoutX="25.0" layoutY="48.0" prefHeight="17.0" prefWidth="351.0" text="1. Оставщееся время ожидание на основе текущей скорости" />  
 <Label layoutX="25.0" layoutY="65.0" prefHeight="17.0" prefWidth="345.0" text="2. Процент нагрузки вводится на серверной части" />  
 <Label layoutX="25.0" layoutY="82.0" prefHeight="17.0" prefWidth="345.0" text="3. Вычисления на основе метода средних прямоугольников" />  
 <Line endX="580.0" layoutX="113.0" layoutY="110.0" startX="-100.0" />  
 <Label layoutX="404.0" layoutY="38.0" prefHeight="17.0" prefWidth="118.0" text="Адрес сервера" />  
 <TextField id="dUpperIntervalTextFiled" fx:id="dIPTextField" layoutX="531.0" layoutY="34.0" />  
 <Label layoutX="405.0" layoutY="75.0" text="Порт сервера" />  
 <TextField id="dDownIntervalTextField" fx:id="dPortTextField" layoutX="531.0" layoutY="71.0" />  
 <Label layoutX="14.0" layoutY="121.0" text="Процесс выполнения" />  
 <ListView id="dLogList" fx:id="dLogList" layoutX="26.0" layoutY="144.0" prefHeight="218.0" prefWidth="655.0" />  
 <Button fx:id="dConnectTextField" layoutX="568.0" layoutY="385.0" mnemonicParsing="false" onMouseClicked="#connect" prefHeight="25.0" prefWidth="112.0" text="Подключиться" />  
 </children>  
</AnchorPane>