**Содержание**

[Введение - ТЗ 2](#_Toc136988641)

[1.Среда разработки 4](#_Toc136988642)

[2.Метод решения задачи 5](#_Toc136988643)

[2.1 Шаг первый 5](#_Toc136988644)

[2.2 Шаг второй 5](#_Toc136988645)

[2.3 Шаг третий 5](#_Toc136988646)

[3.Реализация функциональных требований 6](#_Toc136988647)

[3.1 Создание обложки для приложения 6](#_Toc136988648)

[3.2 Создание кнопок навигации 6](#_Toc136988649)

[3.3 Создание функциональности 7](#_Toc136988650)

[A)Раздел «запчасти» и «сервис» 7](#_Toc136988651)

[B)Раздел «Гараж» и «Расходы» 11](#_Toc136988652)

[Список использованных источников 17](#_Toc136988653)

# Введение - ТЗ

«Лень — двигатель прогресса» - так говорил ещё Вознесенский Андрей Андреевич, русский поэт-шестидесятник, архитектор, прозаик, художник. Точно так же говорит и кандидат экономических наук, доцент, миллиардер, Play-Boy, филантроп и просто хороший человек Никифоров Дмитрий Вячеславович учащийся на кафедре КБ-4. Именно его стремление к оптимизации и определило выбор темы курсовой работы.

Ежегодно покупается миллионы транспорта для передвижения (ТРС) как на суше так и на воде, при этом еще больше запчастей и «расходников» требуется для обслуживания этих самых ТРС каждый год. Тем временем владельцы ТРС должны постоянно помнить что?, где ? и за сколько ? купить. При этом ещё желательно следить за своим бюджетом, чтобы понимать и анализировать свои расходы.

Целью данной работы будет автоматизация учёта расходов на свои ТРС и подбор сервисов или запчастей в одном приложении. Так как отдельно всё это вести проблематично, долго и не удобно.

**ТЗ**

1. Описание проекта:

Нужно создать DESCTOP-приложение для авто-владельцев, которое поможет им следить за своим авто, подбирать запчасти и анализировать расходы. Приложение будет предоставлять информацию о состоянии авто и т.д. . Пользователь сможет добавить новые транспорты, указывать их характеристики (пробег, состояние и т.д) и подбирать детали соответствующие.

1. Функциональные требования:

* Регистрация и авторизация пользователей;
* Создание списка транспорта;
* Добавление нового транспорта;
* Отметка уже выбранных деталей;
* Поиск запчастей по названию;
* Просмотр всех автомобилей в гараже;

1. Нефункциональные требования:

* Backend должен быть реализован на языке программирования Java с использованием фреймворка - Javafx .

1. Особенности реализации:

* Валидация входных данных;
* Пагинация списков;
* Пасринг сайтов

1. Необходимые ресурсы и технологии:

* Язык программирования - Java;
* Фреймворки - Javafx;
* БД -  SQLite;
* Frontend - Java .

1. Планируемый срок разработки:

2 месяца.

# 1.Среда разработки

При разработке данного приложения был использован язык Java, так как этот язык является кроссплатформенным.

В качестве системы контроля версий выбран Git. Также проект содержит несколько распространенных библиотек: JavaFx — для создания графической составляющей приложения, JSOUP — для парсинга данных с сайтов, ADOBE PHOTOSHOP — для создания компонентов для наполнения приложения.

# 2.Метод решения задачи

Решение задачи можно разбить на несколько подпунктов. Процесс записи и учета, состоит из нескольких разделов, каждый из которых отвечает за свою задачу:

2.1 Шаг первый – выбор или создания своего ТРС с указанием данных (Тип, гос. Номер, Год, vin, Название) во вкладке «Гараж».

2.2 Шаг второй – анализ того что сломано или нуждается в обновлении, поиск нужной запчасти и нужного обслуживания в разделе «запчасти» и «сервис» соответственно.

2.3 Шаг третий – после всего, записать все расходы к каждому ТРС

# 3.Реализация функциональных требований

# 3.1 Создание обложки для приложения

Для начала в классе HelloApplication создаем метод start который создаст окно для приложения (Листинг 1.1). Используя размеры под картинку с помощью scene, а так-же устанавливаем стиль окна через StageStyle. После чего подгружаем картинку и устанавливаем её в окне (Листинг 1.2)

public class HelloApplication extends Application {  
  
 int i=0;  
 @Override  
 public void start(Stage stage) throws IOException {  
 Group root = new Group();  
 Scene scene = new Scene(root, 1400, 700);  
 stage.initStyle(StageStyle.*DECORATED*);  
 stage.getStyle();  
 FileInputStream Url;  
 Image url;

Листинг 1.1

Url = new FileInputStream("png/front.png");  
url = new Image(Url);  
ImageView front = new ImageView(url);  
front.setX(0);  
front.setY(0);  
root.getChildren().add(front);

Листинг 1.2

# 3.2 Создание кнопок навигации

Загружаем картинки через чтение из файлов FileInputStream. После этого устанавливаем кнопки по координатам и на них накладываем картинки (Листинг 1.2). И так проделываем для каждой картинки которая используется как кнопка для своего действия. Не забывая про цвет setBackground.

Url = new FileInputStream("png/add\_car.png");  
url = new Image(Url);  
ImageView add\_car\_img = new ImageView(url);  
Button addCar = new Button();  
addCar.setLayoutX(1100);  
addCar.setLayoutY(300);  
addCar.setGraphic(add\_car\_img);  
addCar.setBackground(null);

Листинг 1.3

**Итог**

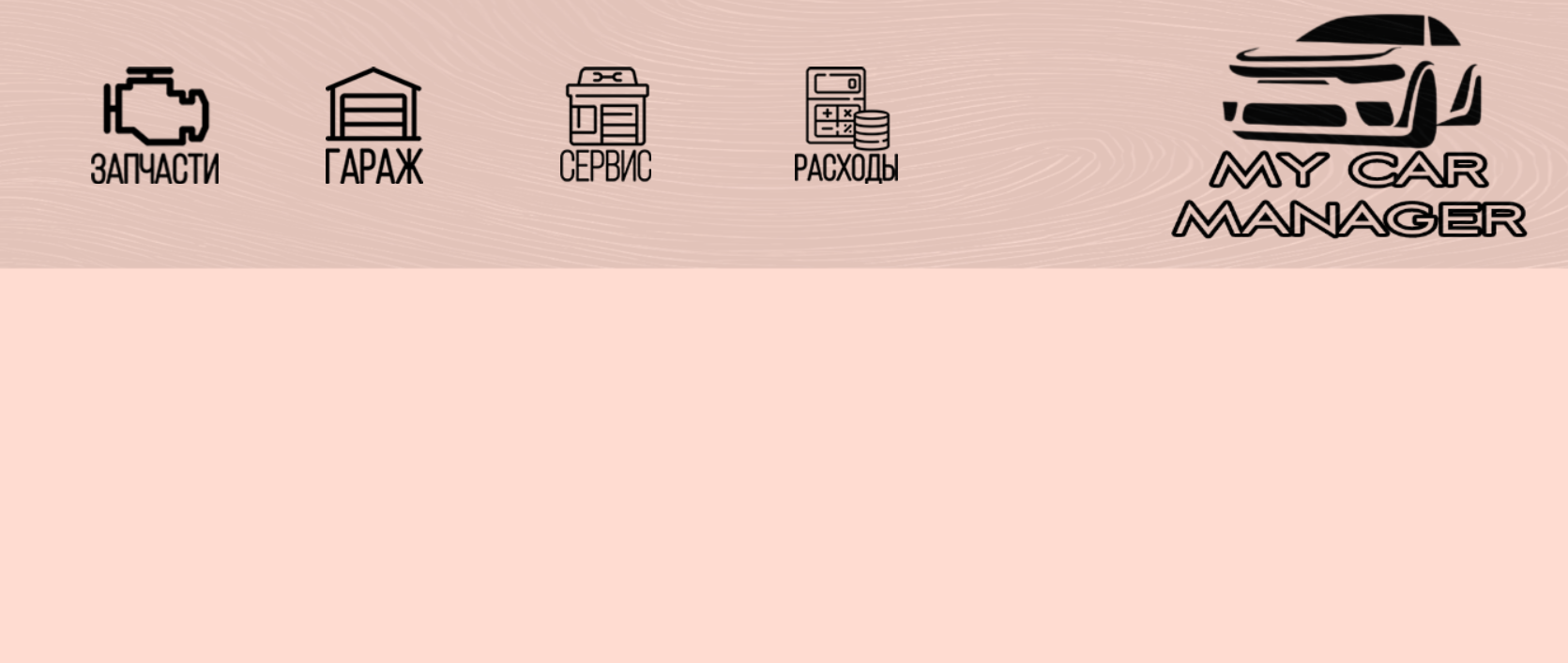


Рисунок 1

# 3.3 Создание функциональности

# A)Раздел «Запчасти» и «Сервис»

Для парсинга используем Jsoup и тащим данные с определеннго сайта (Листинг 2.1). Но для начала создаем метод getPane, в котором создаем поля меню для вывода парсинга с сайтов (Листинг 2.1)

public static Pane getPane (ModelRepairs[] models, int t) throws FileNotFoundException, MalformedURLException {  
 Pane res = new Pane();  
 int u = 10;  
 for(int i = 0; i<t; i++, u+=200){  
 *//System.out.println(models[i].path);* InputStream is = null;  
 ImageView photo = new ImageView();  
 String fileUrl = models[i].path;  
 try {  
 *// create a url object* URL url = new URL(fileUrl);  
 *// connection to the file* URLConnection connection = url.openConnection();  
 is = connection.getInputStream();  
 photo = new ImageView(new Image(is));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }

Листинг 2.1

Далее для раздела «запчасти» используя метод getRepairs, производим парсинг в сайта по ключевому слову(которое мы пишем в поиске) (Листинг2.2). Результат которого выводим в созданные выше поля. Так же добавляем кнопки для выхода назад.

Для раздела «сервис» проделываем практически тоже самое, а так же добавляем ScrollPane, для панельки и прокрутки всего ассортимента с использованием CSS (Листинг 2.3).

public static Pane getRepairs() throws IOException {  
 Pane res = new Pane();  
 Elements all = *Doc*.getElementsByAttributeValue("class", "col-xxx");  
 int i = 0;  
 for(Element el: all){  
 String buyURL = "";  
 Elements title = el.getElementsByAttributeValue("class", "item-v-name");  
 Elements price = el.getElementsByAttributeValue("class", "price");  
 Elements path = el.getElementsByAttributeValue("class", "item\_img");  
 Element imageElement = path.select("img").first();  
 String absoluteUrl = imageElement.absUrl("src");  
 Elements status = el.getElementsByAttributeValue("class", "visible-sm visible-md visible-lg");  
 String st = "";  
 if(status.text().isEmpty()){  
 st = "Нет в наличии";  
  
 }  
 else{  
 st = status.text();  
 Elements buy\_url = el.getElementsByAttributeValue("class", "003 catalog-vitrina-3 fitem \t\t\t\t");  
 Elements buy\_url2 = el.getElementsByAttributeValue("class", "003 catalog-vitrina-3 fitem \twith\_old\_price\t\t\t");  
 Element buyElement = buy\_url.select("a").first();  
 Element buyElement2 = buy\_url2.select("a").first();  
 if(buyElement == null){  
 buyURL = buyElement2.absUrl("href");  
 }  
 else  
 buyURL = buyElement.absUrl("href");

Листинг 2.1

ScrollPane scroll\_garage = new ScrollPane();  
scroll\_garage.setLayoutX(30);  
scroll\_garage.setLayoutY(250);  
scroll\_garage.setMinHeight(0);  
scroll\_garage.setMinWidth(1000);  
scroll\_garage.setMaxHeight(440);  
scroll\_garage.setStyle("-fx-background: transparent; -fx-background-color: #FFffff;"); *// CSSSSSSSS*scroll\_garage.setHbarPolicy(ScrollPane.ScrollBarPolicy.*AS\_NEEDED*);  
scroll\_garage.setVbarPolicy(ScrollPane.ScrollBarPolicy.*AS\_NEEDED*);  
  
ScrollPane scroll\_service = new ScrollPane();  
scroll\_service.setLayoutX(30);  
scroll\_service.setLayoutY(250);  
scroll\_service.setMinHeight(0);  
scroll\_service.setStyle("-fx-background: transparent; -fx-background-color: #FFffff;"); *//CSSSSSSSS*scroll\_service.setMinWidth(1000);  
scroll\_service.setMaxHeight(440);  
scroll\_service.setHbarPolicy(ScrollPane.ScrollBarPolicy.*AS\_NEEDED*);  
scroll\_service.setVbarPolicy(ScrollPane.ScrollBarPolicy.*AS\_NEEDED*);

Листинг 2.2

Итог

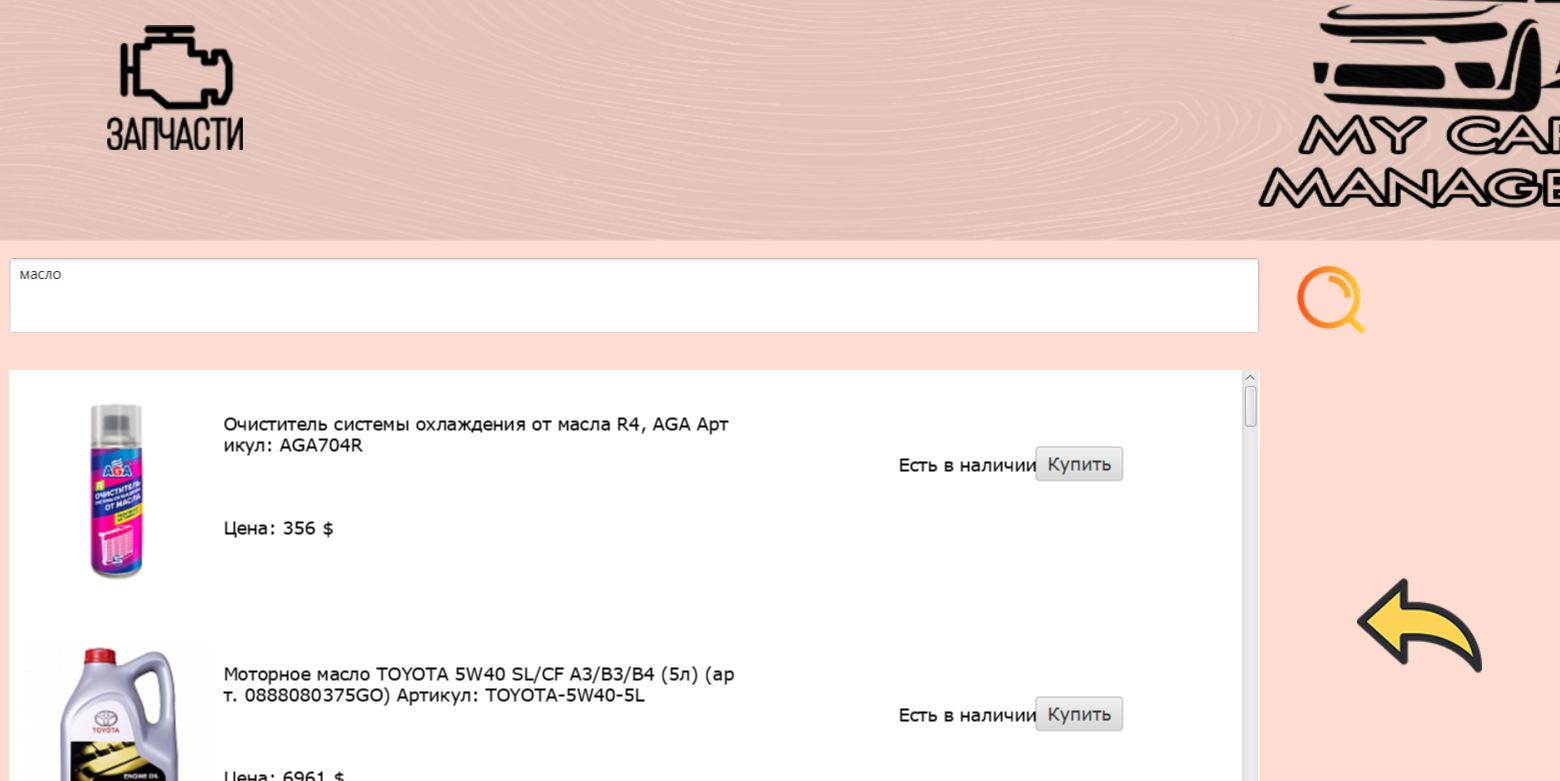


Рисунок 2

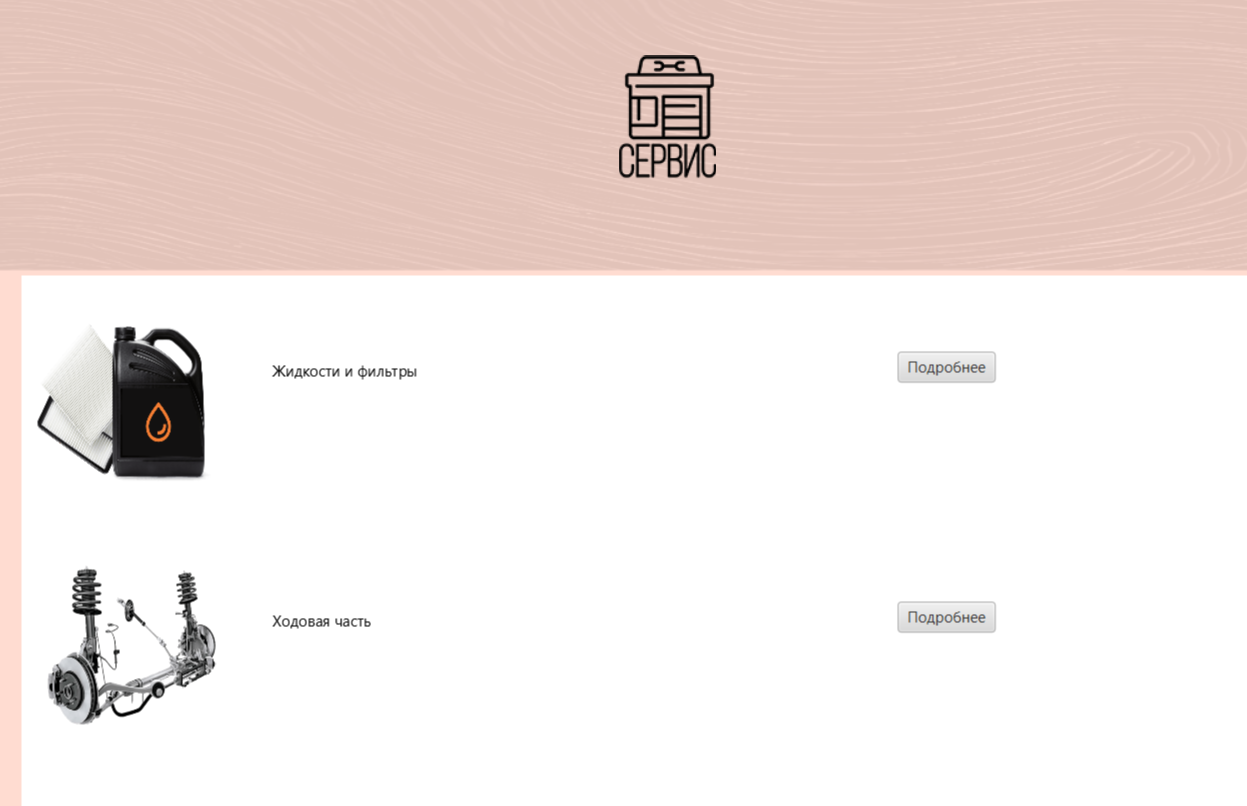


Рисунок 3

# B)Раздел «Гараж» и «Расходы»

Для добавления нового ТРС создаем ещё одно окно которое будет появляться при нажатии на кнопку (Листинг 3.1)

После чего вносим данные которые вставляем в окно (Листинг 3.2) и сохраняем, данные так же сохраняются в базе данных.

addCar.setOnAction(value ->{  
 Group root\_add = new Group();  
 Scene scene\_add = new Scene(root\_add, 400, 600);  
 Stage newWindow = new Stage();  
 newWindow.initStyle(StageStyle.*TRANSPARENT*);  
  
 try {  
 FileInputStream Url1 = new FileInputStream("png/front3.png");  
 Image url1 = new Image(Url1);  
 ImageView front1 = new ImageView(url1);  
 front1.setX(0);  
 front1.setY(0);  
 root\_add.getChildren().add(front1);  
 } catch (FileNotFoundException e) {  
 throw new RuntimeException(e);  
 }

Листинг 3.1

Text title = new Text("Новый транспорт");  
title.setLayoutX(140);  
title.setLayoutY(20);  
title.setFont(Font.*font*("Verdana",16));  
  
Text name\_text = new Text("Название машины:");  
name\_text.setLayoutX(10);  
name\_text.setLayoutY(55);  
name\_text.setFont(Font.*font*("Verdana",13));  
  
TextArea name = new TextArea();  
name.setPromptText("Название машины:");  
name.setLayoutX(160);  
name.setLayoutY(30);  
name.setMaxHeight(13);  
name.setMaxWidth(200);

Листинг 3.2 часть

В разделе «Расходы» аналогично создаем доп. окно в которое вставляем нужные нам строки для записи (Листинг 3.3).

ScrollPane scrollPane = new ScrollPane();  
scrollPane.setLayoutX(10);  
scrollPane.setLayoutY(130);  
scrollPane.setMinWidth(580);  
scrollPane.setMinHeight(10);  
scrollPane.setMaxHeight(390);  
scrollPane.setMaxWidth(590);  
scrollPane.setHbarPolicy(ScrollPane.ScrollBarPolicy.*AS\_NEEDED*);  
scrollPane.setVbarPolicy(ScrollPane.ScrollBarPolicy.*AS\_NEEDED*);  
  
Text title = new Text("Все расходы");  
title.setLayoutX(250);  
title.setLayoutY(30);  
title.setFont(Font.*font*("Verdana",16));  
  
Text car\_text = new Text("Название машины: " + editModel.name);  
car\_text.setLayoutX(10);  
car\_text.setLayoutY(55);  
car\_text.setFont(Font.*font*("Verdana",13));  
  
Text num = new Text("#");  
num.setLayoutX(30);  
num.setLayoutY(100);  
num.setFont(Font.*font*("Verdana",13));

Листинг 3.3 часть

# 4 Важные методы

Метод check, (Листинг 4.1) – который создает окно чека для учета

public static void check(String k) throws SQLException, ClassNotFoundException, FileNotFoundException {  
 String q = String.*valueOf*(k);  
 ModelTransport editModel = DataBase.*getModel\_Tra*(q);  
 String[][] mas = DataBase.*getAll\_byID*(q);  
 Group root\_add = new Group();  
 Scene scene\_add = new Scene(root\_add, 600, 400);  
 Stage newWindow = new Stage();  
 newWindow.initStyle(StageStyle.*TRANSPARENT*);  
  
 FileInputStream Url = new FileInputStream("png/front2.png");  
 Image url = new Image(Url);  
 ImageView front = new ImageView(url);  
 front.setX(0);  
 front.setY(0);  
 root\_add.getChildren().add(front);  
  
 ScrollPane scrollPane = new ScrollPane();  
 scrollPane.setLayoutX(10);  
 scrollPane.setLayoutY(130);  
 scrollPane.setMinWidth(580);  
 scrollPane.setMinHeight(10);  
 scrollPane.setMaxHeight(390);  
 scrollPane.setMaxWidth(590);  
 scrollPane.setHbarPolicy(ScrollPane.ScrollBarPolicy.*AS\_NEEDED*);  
 scrollPane.setVbarPolicy(ScrollPane.ScrollBarPolicy.*AS\_NEEDED*);

Листинг 4.1

# 

Рисунок 4.0 – создание чека для учета

# 5. Анализ памяти в VisualVM

Так же, для анализа потребления памяти приложение проверено в VisualVM - это визуальный инструмент, объединяющий инструменты командной строки JDK и легкие возможности профилирования. Предназначен как для разработки, так и для производства.

Результат представлен на Рисунках 4.1 - 4.3

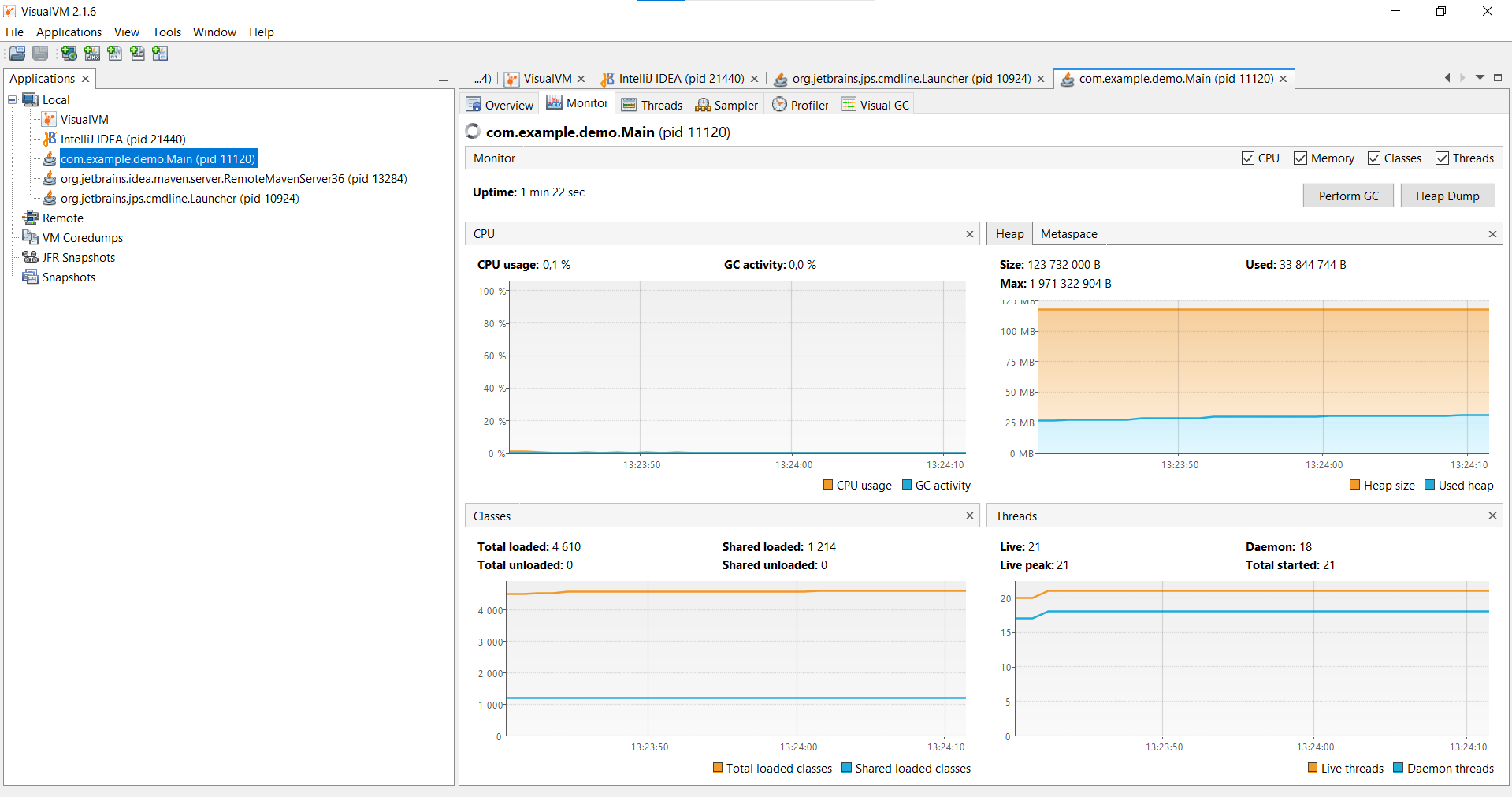


Рисунок 4.1 - Потребление памяти через 1 минуту 22 секунду

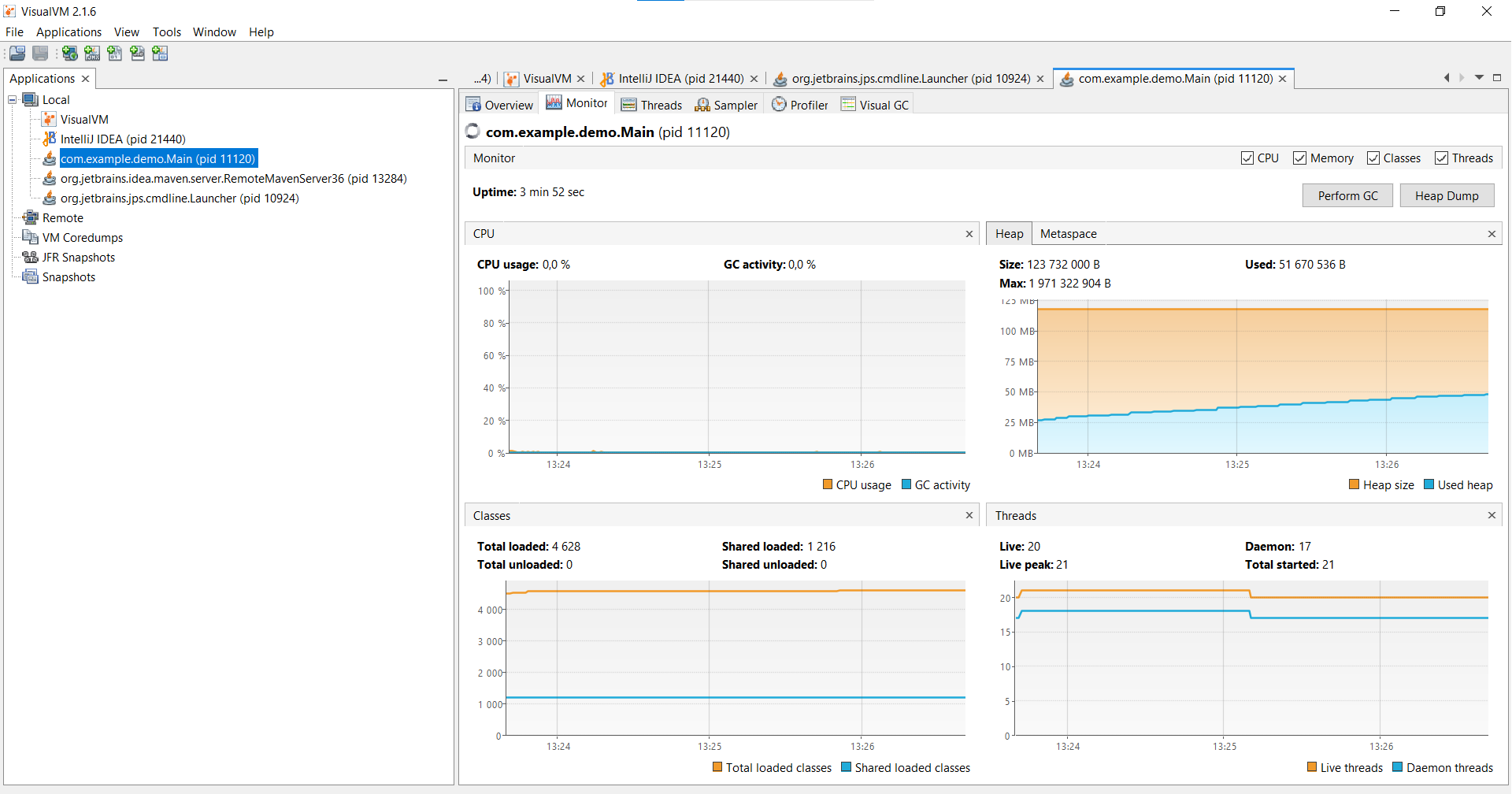


Рисунок 4.2 - Потребление памяти через 3 минуты 52 секунды

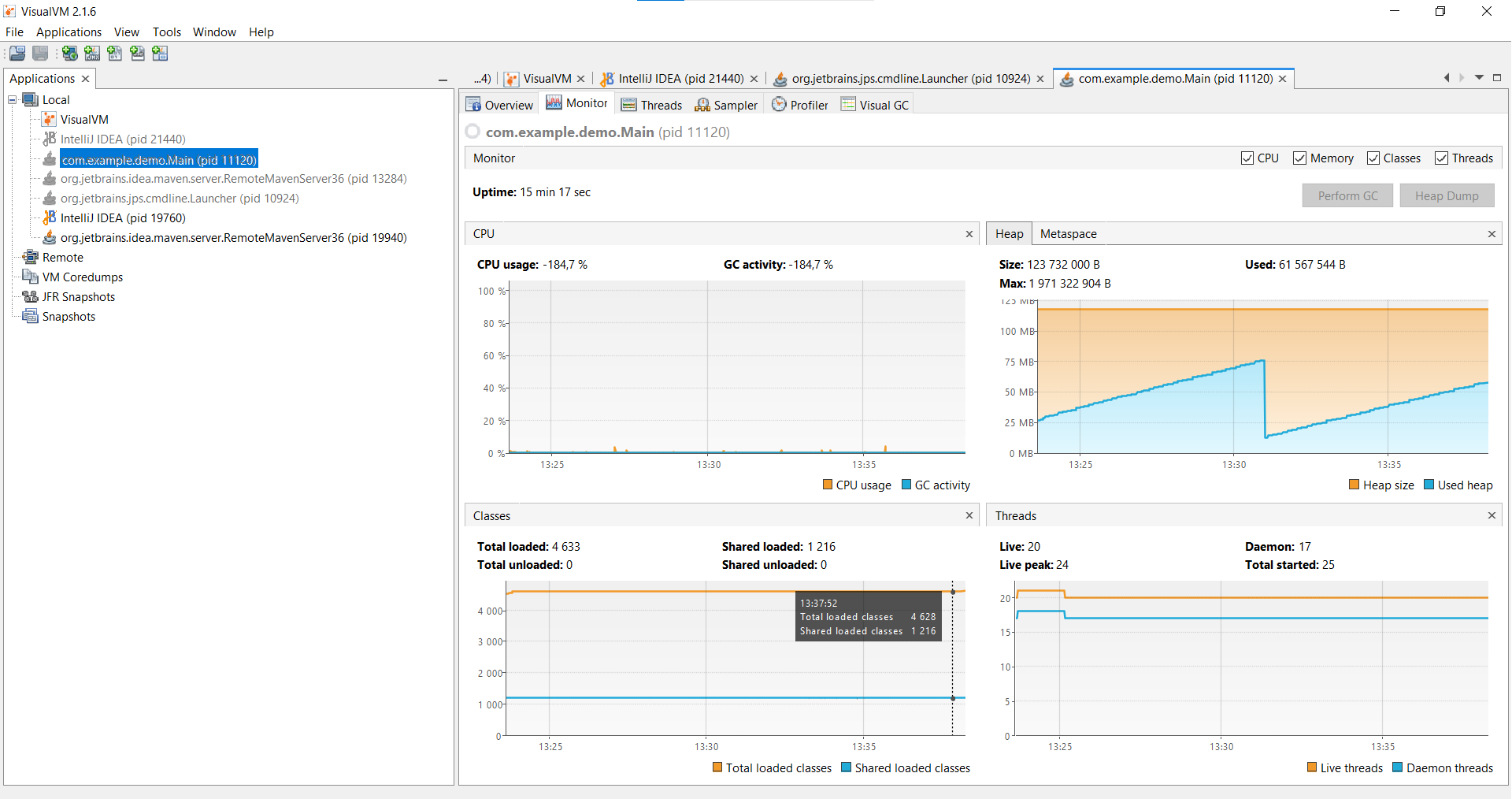


Рисунок 4.3 - Потребление памяти через 15 минут 17 секунд

# 6. Анализ программного кода

Код программы был проанализирован при помощи статического анализатора SonarLint (это бесплатный плагин для IDEA с открытым исходным кодом, которой идентифицирует и помогает устранить проблемы качества и безопасности при написании кода). В ходе проверки, никакой критической ошибки не обнаружено.

Результат, приведен на Рисунке 5

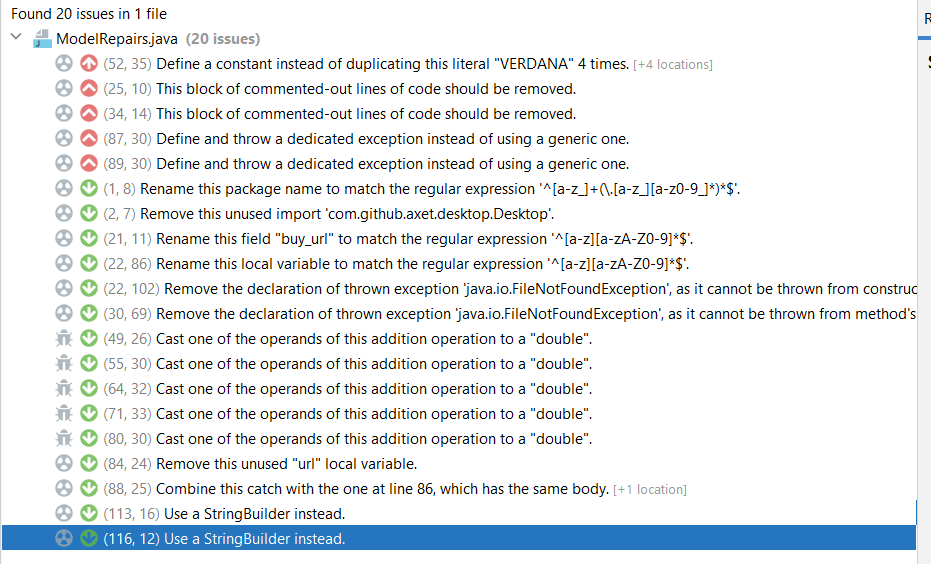


Рисунок 5.

# 

# Список использованных источников

1. JavaFX [Электронный ресурс]: сайт с документацией по использованию библиотеки. – режим доступа: https://openjfx.io/openjfx-docs/#install-java (дата обращения: 10.05.23)
2. Habr [Электронный ресурс]: сайт с документацией по использованию frontend – режим доступа: https://qna.habr.com/q/882895 (дата обращения: 10.05.23)
3. SQLite.org [Электронный ресурс]: сайт с документацией по использованию SQL – режим доступа: https://www.sqlite.org/index.html (дата обращения: 10.05.23)