МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий

(наименование института полностью)

Кафедра «Прикладная математика и информатика»

(наименование кафедры)

02.03.03 математическое обеспечение и администрирование

(код наименование направления подготовки, специальности)

информационных систем

Технология программирования

БАКАЛАВРСКАЯ РАБОТА

на тему «Разработка цифровой приборной панели для автомобиля»

Студент П.С. Дончук _____

(И.О. Фамилия) (личная подпись)

Руководитель В.С. Климов _____

(И.О. Фамилия) (личная подпись)

Допустить к защите

Заведующий кафедрой А.В. Очеповский _____

(ученая степень, звание, И.О. Фамилия) (личная подпись)

«_____» _____ 2018 г.

Тольятти 2018

АННОТАЦИЯ

Тема выпускной квалификационной работы – «Разработка цифровой приборной панели для автомобиля».

Объектом исследования в настоящей работе являются обьектно – ориентированные методы и их применение для разработки программного обеспечения.

Целью работы является повысить функционал и удобство использования автомобильной приборной панели, путем разработки её цифрового аналога.

Бакалаврскую работу можно разделить на несколько логически связанных частей:

В первой главе исследуется предметная область, формализуются требования к программному продукту, описывается используемый инструментарий и затрагиваемые технологии.

Во второй главе рассматриваются методы программирования и описывается непосредственный процесс конструирования ПО.

В третьей главе проводится тестирование разработанного приложения, с целью проверки работоспособности на различных устройствах, а также проверка работоспособности отдельных модулей.

В выпускной работе был презентуется программный продукт, выполняющий функции приборной панели автомобиля.

Выпускная квалификационная работа изложена на 40 страницах, содержит 27 графических изображений и 2 таблицы. Список литературы включает в себя 24 источника. К работе предоставляется 3 приложения.

Работа выполнена студентом Тольяттинского государственного университета, института математики, физики и информационных технологий, группы МОб – 1402, Дончуком П.С.

ABSTRACT

The title of current graduation work is «Development Digital Dashboard for a Car». This graduation work is about creating of an application for Android OS that will be able to read data from the adapter, connected to the car and displaying it on the device screen. Digital dashboard is more precise and has unlimited space for customization for more convenient display. The idea of digital instrument panels opens up great opportunities for developers, manufacturers and end users. You can change the unit of measure, appearance or output the specific sensor directly to the dashboard.

A united on-board system is needed, which combining all the functions required by a modern user, and giving the driver maximum information in a familiar and convenient format.

To realize this target, we were decided to transfer all the analog devices to a digital implementation, in order to display their readings on the screen of a multifunctional on-board system.

The graduation work may be divided into several logically connected parts which are: analysis of the subject area and conceptual requirements; development of program code; software testing; conclusion.

In conclusion, we would like to emphasize that this work has great potential for the development of the automotive industry. New opportunities for many interconnected industries are able to extend all automotive industry forward, to the introduction of "smart" technologies.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время основным источником вывода важных параметров, о текущем состоянии автомобиля, такими как, его скорость, количество оборотов двигателя или уровень оставшегося топлива является приборная панель. Информация на ней подается пользователю посредством аналоговых приборов, получающих информацию с электронных датчиков. Производители все чаще в автомобиль встраивают системы навигации, мощные бортовые компьютеры, предоставляющие множество информации, многофункциональные системы, включающие в себя мультимедийные возможности, навигацию, парковочные камеры и радары, видеорегистраторы и т.д.

Но все эти модули как правило встраиваются в переднюю панель, и водителю приходится отвлекаться от дороги, чтобы увидеть свой дальнейший маршрут или переключить станцию на радиоприемнике. Современное техническое оснащение автомобиля диктует свои требования, все возможности по управлению автомобилям и сопутствующими системами уже невозможно уместить в традиционную панель приборов.

Необходима единая бортовая система, сочетающая в себе все функции, требующиеся современному пользователю и дающая водителю максимум информации в привычном и удобном формате.

Для реализации этой цели было решено перенести все привычные приборы в цифровую реализацию, дабы выводить их показания на экран многофункциональной бортовой системы.

Объект бакалаврской работы: обьектно – ориентированные методы и их применение для разработки программного обеспечения.

Предмет бакалаврской работы: повышение функционала и удобства использования приборной панели, путем разработки её цифрового аналога.

Цель бакалаврской работы: разработка цифровой приборной панели для автомобиля.

Задачами бакалаврской работы, исходя из поставленной цели, являются:

— формирование представления о разрабатываемом программном продукте;

— разработка программы цифровой приборной панели;

— тестирование программы цифровой приборной панели.

Бакалаврская работа состоит из трех глав:

Первая глава работы посвящена анализу состояния вопроса в настоящее время, формированию требований к разрабатываемому ПО, выбору языка программирования и обзору используемого инструментария.

Вторая глава описывает проектирование и разработку программы цифровой приборной панели для автомобиля.

В третьей главе составляется план тестирования, разработанного ПО, проводится тестирование и анализируются полученные результаты.

1 ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И КОНЦЕПТУАЛЬНЫХ ТРЕБОВАНИЙ

1.1 Описание исследуемых задач

В наше время на дорогах появляется все больше автомобилей, что подстегивает эту индустрию к развитию. Для примера в США на 1000 человек приходится 809 автомобилей, в России 228, в Великобритании 525. В данной сфере постоянно вводятся инновации, меняются электроника в автомобилях и вид приборной панели, ведь каждый производитель хочет быть уникальным, и не похожим на остальных. Но, стоит задаться вопросом, нужно ли это среднестатистическому автовладельцу? Ведь эти обновления могут быть в ущерб функциональности и удобству восприятия, в угоду дизайну или же просто будут идти в разрез с привычками конкретного человека и его личными предпочтениями. Конечно, на многие модели автомобилей можно приобрести и установить нестандартные панели, которые будут отличаться внешним видом и функционалом от заводских аналогов. Но, стоит отметить что это дополнительные затраты времени и финансов для автовладельца, к тому же и предложенные варианты исполнения этих изделий могут не удовлетворять требованиям покупателя.

Поэтому темой этой работы была выбрана «Разработка цифровой приборной панели для автомобиля».

Для реализации функционала интегрированных бортовых систем обычно применяют системы, основанные на Linux, а для реализации мультимедийных возможностей применяют Android, который основан на Linux.

Приложение, написанное для Android можно использовать как на системах, использующих различные сборки Linux так и на любом устройстве под управлением Android, что делает данную платформу наиболее предпочтительной для разработки программы. В ходе данный работы будет

разработано приложение, способное полностью заменить собой все бортовые приборы в автомобиле с любой конфигурацией оборудования.

1.2 История появления и развития цифровых приборных панелей

Современный автомобиль – это сложное единство высокоточной механики и высокотехнологичной электроники. Сегодня микропроцессоры повсюду. Они небольшие по размеру, надежны в функционировании, эффективны при управлении различными параметрами. Поэтому они нашли свое место в автомобилестроении, где контроль над параметрами любого вида пользуется большим спросом.

Рассмотрим разработки, имеющиеся в данной сфере, из разных поколений, классов автомобилей и от разных производителей.

Первой в области цифровых приборных панелей стала компания Aston Martin с моделью Lagonda 2 – Series в далеком 1976 году. Перед водителем располагались три вакуум – люминесцентных экрана (рисунок 1.1), отображающих:

— скорость автомобиля;

— уровень топлива, текущий режим коробки передач и различные предупреждения;

— количество оборотов двигателя и текущее время.

Рисунок 1.1 – Приборная панель Aston Martin Lagonda

Первые попытки привнести в индустрию массового автомобилестроения цифровые приборные панели начались приблизительно в 1980– 1990 году. Тогда широкой публике были представлены Pontiac 6000STE (рисунок 1.2), Renault 11 (рисунок 1.3) и Buick Reatta (рисунок 1.4). И если Pontiac был довольно скромен в этом вопросе, и лишь часть приборов у него были реализованы цифровым способом, то в Renault 11 приборная панель была полностью цифровой. В Buick Reatta производитель развивал это, и цифровая приборная панель была дополнена сенсорным экраном, что по меркам той эпохи было крайне высокотехнологичным решением, однако неудобным для конечного пользователя, из – за низкого уровня развития сенсорных экранов в то время.

Рисунок 1.2 – Приборная панель Pontiac 6000STE

Рисунок 1.3 – Приборная панель Renault 11

Рисунок 1.4 – Приборная панель Buick Reatta

В следующем десятилетии все наработки прошлых лет из экспериментально – концептуальных эволюционировали в удобные для массового рынка пользователей. Цифровые приборные панели стали массово устанавливаться автомобили бюджетного сегмента. Также последовали улучшения эргономики и визуального качества приборов. Например, в Suzuki Liana (рисунок 1.5) была применена компоновка панели с использованием 2х дисплеев – шкал и 2х дисплеев выводящих символьную информацию. В Toyota Yaris (рисунок 1.6) панель состояла из одного дисплея, вмещающего в себя всю возможную для вывода информацию. Так же существовал отечественный «эксперимент» по внедрению цифровых приборных панелей в Lada Samara (рисунок 1.7). Предназначалась модификация автомобиля с этой панелью на экспорт и было выпущено 1500 экземпляров. Она была исполнена в стиле ретрофутуризм и состояла из 3х цифровых шкал, одного дисплея для вывода чисел, индикационных ламп и аналогово одометра.

Рисунок 1.5 – Приборная панель Suzuki Liana

Рисунок 1.5 – Приборная панель Toyota Yaris

Рисунок 1.6 – Приборная панель Lada Samara 43501.1 21093 – 385601

В актуальном поколении автомобилей применяют либо классические приборы, дополненные жидкокристаллическими экранами (рисунок 1.7), в

бюджетном сегменте, либо функцию приборной панели выполняет монитор с виртуальными приборами (рисунок 1.8).

Рисунок 1.7 – Приборная панель Lada Vesta

Рисунок 1.8 – Приборная панель Ford Mustang 2018

Многие из сохранившихся в наше время циферблатов – это цифровые реплики аналоговых приборов.

Сегодняшние аналоговые циферблаты, такие как часы, существуют больше для украшения, чем для выполнения функциональных задач.

1.3 Выявление требований к интерфейсу и разработка его концепции

Традиционная компоновка приборной панели предполагает равный размер спидометра и тахометра. За счет изменения размера приборов можно выразить характер автомобиля. Например, в спортивных автомобилях тахометр, как правило, крупнее, и иногда буквально затмевает спидометр, а в городском автомобиле главную роль играет спидометр, так как водителю важно соблюдать установленный скоростной режим. Разрабатываемая в рамках данной работы приборная панель не предназначена для какого– либо конкретного автомобиля, поэтому при разработке дизайна будут использоваться представления об эргономике и данных, обходимых к представлению максимально большому кругу пользователей. Вышеуказанные представления были сформированы на основе на основе данных, рассмотренных в пункте 1.2 «История появления и развития цифровых приборных панелей».

Спроектируем интерфейс, исходя из сформированных представлений и требований. В центре панели будет расположен циферблат тахометра со стрелкой, а справа и слева от него расположим функциональные поля с различной информацией.

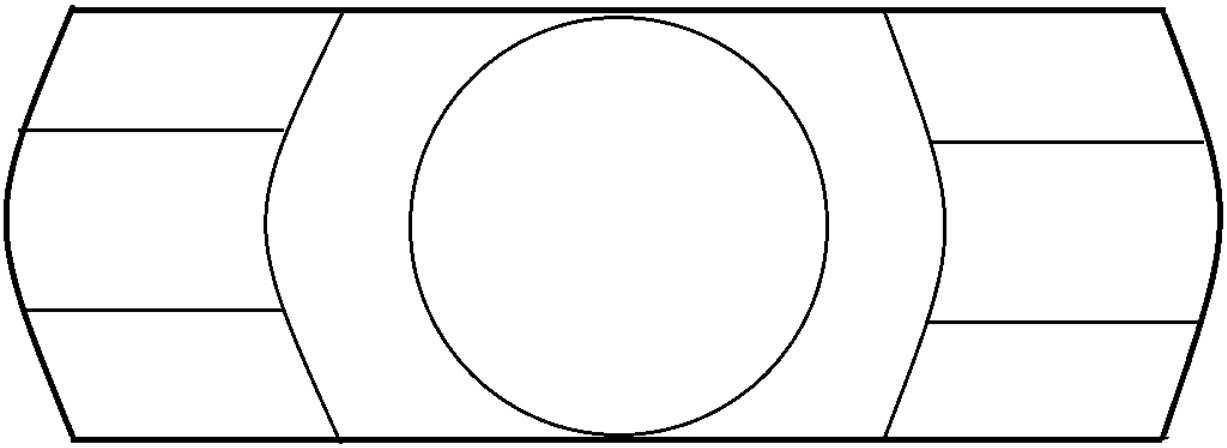

Рисунок 1.9 – Концептуальная модель дизайна интерфейса приборной панели

После разработки схематичного вида интерфейса (рисунок 1.9) выделим информацию, необходимую к выводу:

— текущая скорость автомобиля;

— температура охлаждающей жидкости;

— температура окружающего воздуха;

— текущие обороты двигателя;

— текущая мощность, вырабатываемая двигателем;

— расход топлива;

— уровень топлива в баке.

Нанесем на концептуальную модель дизайна интерфейса информацию, необходимую для вывода (рисунок 1.10).

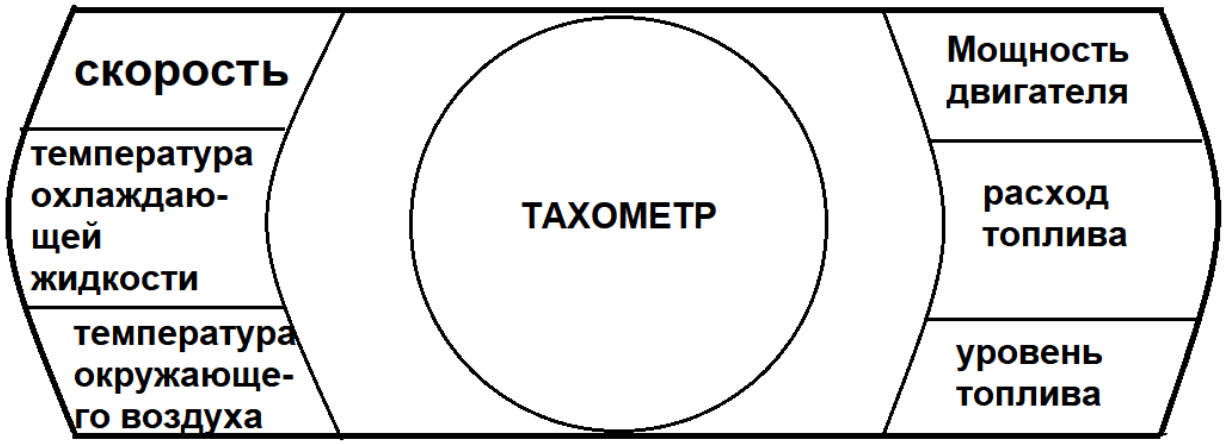

Рисунок 1.10 – Расположение блоков выводимой информации на интерфейсе

Показания, имеющие максимальные значения, таких как температура охлаждающей жидкости и уровень топлива, для наглядности выразим при помощи полукруглых шкал.

1.4 Формирование требований к программному продукту.

Исходя из рассмотренных примеров, существует минимум приборов, необходимых каждому пользователю, его необходимо реализовать в полном объёме, остальные приборы не критичны, и в автомобиле может не имеется необходимых подсистем для сбора соответствующих данных. Это спидометр, тахометр, и уровень топлива, данные три шкалы обязательны для любой приборной панели. Также, во всех современных автомобилях имеется индикатор «check engine», который загорается, в случае проблем с двигателем и сопутствующими системами, такими как топливный насос, система выхлопа и т.д. Пренебречь данным индикатором нельзя, так как вовремя не исправленная проблема может привести к более серьёзному ущербу для авто или же повлечь за собой катастрофические последствия, в случае отказа одной из важных систем во время движения.

Программа должна:

1. Осуществлять взаимодействие с пользователем, используя интерфейс.
2. Устанавливать соединение с блоком управлением автомобилем.
3. Получать и обрабатывать данные, полученные от адаптера.
4. Предоставлять полученные данные пользователю.
5. Иметь функциональную возможность предоставлять пользователю следующие данные, обновляемые в реальном времени:

5.1.Текущая скорость.

5.2.Текущее количество оборотов двигателя.

5.3.Количество оставшегося топлива в процентах.

5.4.Температура охлаждающей жидкости.

5.5. Температура внешнего воздуха.

5.6. Температура всасываемого воздуха.

5.7. VIN – код транспортного средства.

5.8. Иметь функциональный аналог «Check engine».

Основываясь на сформированных требованиях, необходимо составить схему запросов и взаимодействий (рисунок 1.3), исходя из которой, мы и будем проектировать и реализовывать программу.

1.5 Выбор языка для реализации программного продукта.

В программировании для Android используются обьектно – ориентированные технологии, поэтому в данном разделе мы приведем обзор объектных технологий.

В условиях постоянно растущего спроса на новые мощные программные продукты достаточно трудно сочетать такие требования, как быстрота разработки, правильность работы и экономичность. Объекты (а точнее классы, на основе которых создаются объекты) по сути представляют собой повторно используемые программные компоненты. В качестве объектов могут использоваться дата, время, видео, человек, автомобиль и другие предметы материального мира. Практически каждое существительное может быть адекватно представлено программным объектом в понятиях атрибутов (например, имя, цвет и размер) и поведений (например, вычисление, перемещение и передача данных). Разработчики программ видят, что использование модульной структуры и обьектно – ориентированного проектирования при разработке приложений повышает продуктивность работы. Этот подход пришел на смену применявшемуся ранее структурному программированию обьектно – ориентированный код проще понять и изменить.

Составим таблицу самых популярных языков обьектно – ориентированного программирования (таблица 1), по тем или иным

критериям подходящих для выполнения нашей задачи и выберем оптимальный, основываясь на выбранных критериях. Результаты будем оценивать от 0 до 5.

Язык Kotlin хоть и является крайне удачным для решения поставленной задачи, оценивать мы его не будем, так как в свободном доступе крайне мало информации и обучающих материалов по нему, ввиду того, что язык относительно «молодой». Как было выше упомянуто, целью данной работы было поставлено формирование идеи максимально простой для внедрения технологии, в первую очередь при оценке нужно исходить из распространённости языка, количество информации по разработки на нем, а также количеству подготовленных программистов, работающих с конкретным языком, и простоте обучения новых специалистов.

Таблица 1 – Сравнительная таблица языков программирования

	Java	C++	Python
Количество активных репозиториев на GitHub(по данным GitHut)	5	2	4
Популярность языка на StackOverflow и GitHub (RedMonk)	5	2	3
Анализ обсуждений в Twitter касательно работы связанной с программированием(Jobs Tractor)	5	1	3
Рейтинг по количеству квалифицированных инженеров, курсов и ранжированию в поисковых системах. (TIOBE)	5	4	1
Изучение языка в рамках образовательного курса 02.03.03 «Математическое обеспечение и администрирование информационных систем»	4	4	0
Имеющиеся навыки разработки на каждом языке	5	2	0
Удобство переноса на другие платформы	5	3	3
Итоговый результат	**34**	**18**	**14**

По итогам оценки для реализации программы был выбран язык Java, как самый востребованный, распространённый и, как следствие, обладающий поддержкой наибольшего, среди всех языков программирования, сообщества. Но главное преимущество языка состоит в его кроссплатформенности, за счет того, что программы, написанные на Java, компилируются всего один раз и после этого могут работать на любом устройстве, на котором установлена JVM, для остальных же языков необходима компиляция под конкретную ОС.

1.6 Обзор необходимого инструментария и используемых технологий.

Операционная система Android – операционная система, разработанная компанией Android, lnc., которая затем была приобретена Google. На сегодняшний день самая популярна мобильная ОС. Система основана на ядре Linux с использованием виртуальной машины Java. Система является полностью свободной и любой желающий может экспериментировать как с ее модификацией, так и с полной переработкой системы. Последняя на данный момент версия Android 8.1.0 Oreo.

Android studio – интегрированная среда разработки, предназначенная для написания приложений под ОС Android, разработанная корпорацией Google и выпущена в свободный доступ в 2013 году. Особенность этой IDE в том, что она поставляется сразу со всеми необходимыми для разработки пакетами, имеет встроенную виртуальную машину для эмуляции android устройства с настраиваемыми параметрами для тестирования и отладки приложений в различных условиях и на различных конфигурациях оборудования.

eXtensible Markup Language (расширяемый язык разметки) (XML) – Язык, состоящий из текста и описывающих его тегов, а расширяемым его называют из за отсутствующих ограничений разметки, которую волен

изменять разработчик под свои задачи (в рамках синтаксиса). XML используется в качестве средства для описания грамматики других языков и контроля за правильностью составления документов. В сущности, данный язык не содержит в себе тэгов, предназначенных для разметки, он просто определяет порядок их создания для дальнейшего использования.

Немаловажными особенностями XML является осуществление контроля за корректностью данных, хранящихся в документах, произведение проверок иерархических соотношений внутри документа и установление единого стандарта структуры документов, содержимым которых могут являться данные различных типов. Эти особенности позволяют использовать XML при построении сложных информационных систем, в которых обмен информацией между различными приложениями, работающими в одной системе, является необходимостью.

Java – Обьектно – ориентированный язык программирования, разработанный Sun Microsystems. Главная особенность языка заключается том, что созданные на Java программы транслируются в байт – код Java, в последствии исполняемый виртуальной машиной Java (JVM) – программой, обрабатывающей байт – код как интерпретатор и передающей оборудованию соответствующие инструкции.

Достоинством подобного подхода к исполнению программ является независимость разработанного приложения от операционной системы и оборудования, что позволяет выполнять разработанные на языке Java приложения на любом устройстве, для которого существует соответствующая виртуальная машина.

Другой важной особенностью технологии Java является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. При попытке приложения выполнить операцию, которая превышает выданные программе программы (например,

попытка несанкционированного доступа к данным или соединения с другим устройством), вызывают немедленное прерывание исполнения программы.

Android Software developer kit (комплект для разработки программного обеспечения) (SDK) – это набор средств разработки, используемых для разработки приложений для платформы Android. Android SDK включает в себя следующее: Необходимые библиотеки, отладчик, эмулятор, соответствующая документация для интерфейсов прикладных программ Android (API), примеры исходного кода, учебники по ОС Android.

Android Debug Bridge – (Отладочный мост Android) (ADB) – инструмент, который устанавливается вместе с Android – SDK и позволяет управлять устройством на базе ОС Android. Работает на всех Android – устройствах, где данный функционал не был намеренно заблокирован производителем.

ADB – консольное приложение для ПК, с помощью которого производится отладка Android устройств, в том числе и эмуляторов. Работает по принципу клиент – сервер.

При первом запуске ADB с любой командой создается сервер в виде системной службы (демона), которая будет прослушивать все команды, посылаемые на порт 5037.

ADB позволяет:

1. Посмотреть какие устройства подключены и могут работать с ADB.
2. Просматривать логи.
3. Копировать файлы с/на аппарат.
4. Устанавливать/Удалять приложения.
5. Удалять (очищать) раздел data.
6. Прошивать (перезаписывать) раздел data.
7. Осуществлять различные скрипты управления.
8. Управлять некоторыми сетевыми параметрами.

Java developer kit (JDK) – Комплект разработчика ПО, разработанный Sun Microsystems(в данный момент Oracle Corporation) основанный на языке Java. Данный пакет включает в себя компилятор (javac), стандартные библиотеки классов, примеры, документацию, а так же различные утилиты и JRE. С JDK не поставляется среды разработки на Java, в связи с этим, если вы используете только этот комплект, вам придется компилировать программу при помощи консольных команд, а код программы писать в стороннем текстовом редакторе. Все современные интегрированные среды разработки приложений на Java, используют инструментарий из JDK и потребуют предварительной установки пакета или же поставляются с необходимым инструментарием из комплекта JDK

Электронное бортовое устройство (ЭБУ) – блок управления электронными системами, расположенный под панелью приборов, и представляющий собой центр электронной системы контроля двигателя. Он непрерывно обрабатывает информацию, поступающую от всевозможных подсистем и датчиков, и управляет системами, контролирующими эксплуатационные показатели автомобиля, диагностику ошибок и так далее.

ЭБУ включает выходные цепи (форсунки, различные реле и пр.) путем замыкания их на «массу» через выходные транзисторы контроллера. Единственное исключение – цепь реле топливного насоса. Электробензонасос запитывается через силовое реле. В свою очередь, обмоткой реле управляет ЭБУ посредством замыкания одного из выводов на «массу».

ЭБУ оснащен встроенной системой диагностики. Он может распознавать неполадки в работе электронной системы управления двигателем(ЭСУД), предупреждая о них водителя через контрольную лампу «Check engine» (проверьте двигатель). Также, электронное бортовое устройство хранит в оперативной (энергозависимой) памяти диагностические коды ошибок, указывающие на неисправность конкретного элемента

системы и характер этой неисправности, чтобы помочь специалистам в проведении диагностики и ремонта автомобиля.

1.7 Подключение к ЭБУ автомобиля и сбор данных

Подключится к бортовым системам автомобиля можно посредством диагностического разъёма, или же разъёма OBD. Мы можем реализовать как проводное подключение, с помощью кабеля, так и беспроводное, используя Bluetooth или Wi – Fi. Так как для реализации программы беспроводное подключение отличается от проводного только необходимостью в блоке кода, который управляет соответствующими модулями, а Bluetooth от Wi – Fi отличается только парой строк кода, мы будем использовать Bluetooth, как более экономичный к электроэнергии, в сравнении с Wi – Fi. На данный блок вывода завязаны все бортовые датчики, что позволяет получать информацию о всех системах автомобиля.

Вся информация, собираемая датчиками автомобиля, а также данные диагностики хранятся в памяти ЭБУ, и к ним можно получить доступ через диагностический разъем.

Выводы по главе 1: в главе 1 был проведен анализ предметной области, спроектированы модели программного продукта и описаны требования, предъявляемые к разрабатываемому программному продукту. В результате обзора существующих решений были выявлены их ключевые особенности, а также достоинства и недостатки. В конце главы были описаны и программные средства разработки.

2 РАЗРАБОТКА ПРОГРАММЫ ЦИФРОВОЙ ПРИБОРНОЙ ПАНЕЛИ

2.1 Подготовка к разработке ПО

Первое, с чего нам следует начать, это выбор и установка интегрированной среды разработки и пакетов разработчика. В нашем случае в качестве IDE выступает Android Studio, которая сама установит SDK и JDK, если необходимо. Далее нужно подготовить устройство для установки и тестирования разрабатываемого приложения. В данной работе мы рассмотрим два метода, один при помощи виртуальной машины на ПК, второй при помощи физического устройства, подключённого через ADB.

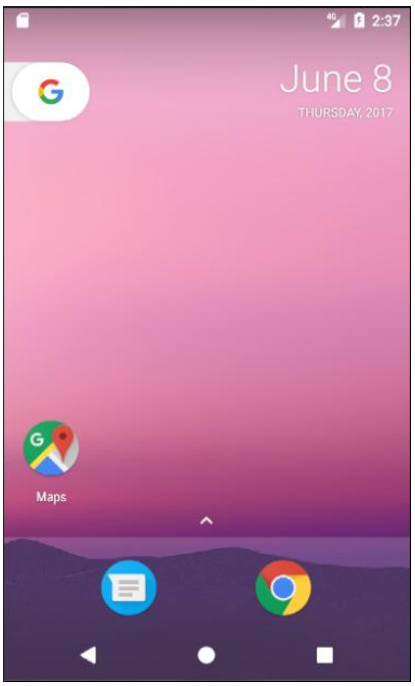

Рисунок 2.1 – Вид виртуального устройства (Nexus 4)

В меню настроек виртуального устройства (рисунок 2.1) можно совершить звонок или отправить смс, задать местоположение устройства, выбрать показания любых датчиков, таких акселерометр, магнитометр, датчик приближения и так далее, при этом можно использовать и крайне специфичные датчики, такие как датчик давления или термометр.

Виртуальное устройство является очень гибким инструментом для отладки, способном заменить множество реальных устройств простыми корректировками конфигурации. Можно выбрать любую версию ОС Android,

разрешения экрана и т.д. что позволяет ускорить разработку, отладку и тестирование во много раз и при этом не требует наличие у разработчика огромного количества устройств.

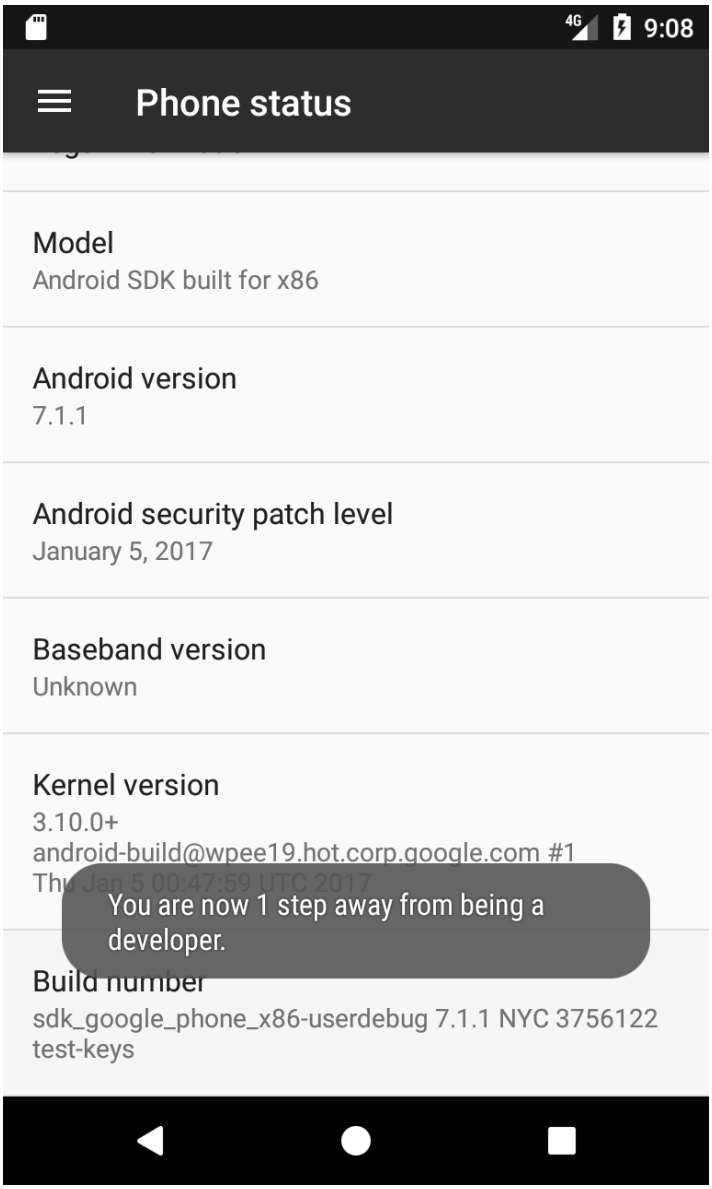

Рисунок 2.2 – Получение доступа к «меню разработчика»

При использовании физического устройства для тестирования необходимо активировать режим разработчика (рисунок 2.3). Для этого нужно найти в настройках пункт «Информация об устройстве» и быстро несколько раз нажать на строку «номер сборки». Начнется обратный отсчёт нажатий, после которого мы получим доступ в раздел для разработчиков. В этом разделе необходимо будет активировать пункты «отладка по USB» и «установка по USB», при помощи флажков(чек-боксов). Затем подключить

его по USB к компьютеру с установленным ADB и все, можно использовать его как отладочное устройство. При запуске приложения в среде Android Studio, сразу после сборки оно будет устанавливаться и запускаться на физическом устройстве, для последующей отладки и тестирования.

Разработаем блок схему работы основной функции программы (рисунок 2.3), для понимания этапов реализации.

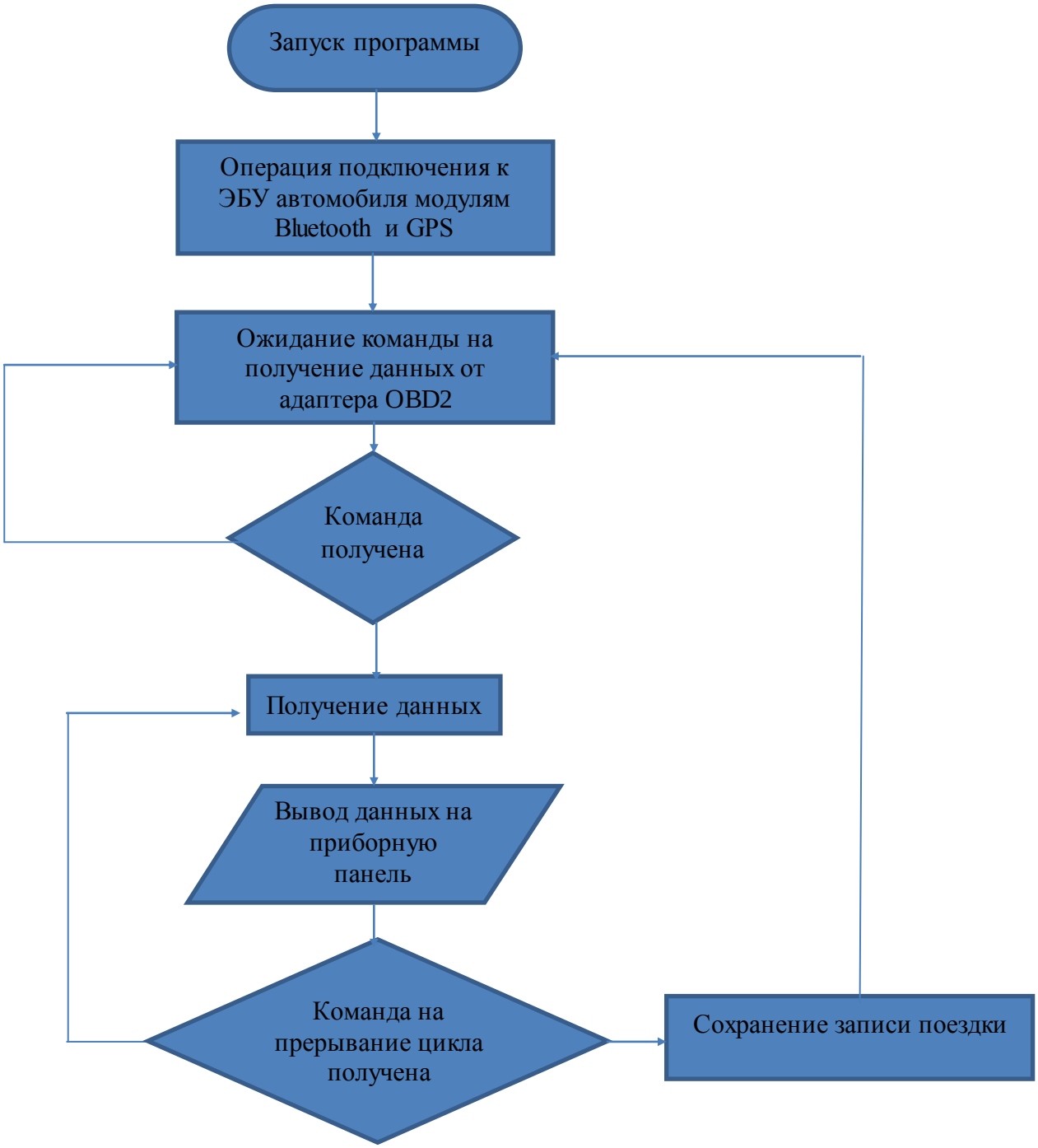

Рисунок 2.3 – Схема работы основного цикла программы

Рисунок 2.4 – Уровни программного обеспечения в Android

Любое устанавливаемое приложение начинает свою работу на устройстве с файла «AndroidManifest.xml». Этот файл содержит необходимую информацию о конфигурации оборудования, которая позволяет правильно установить приложение на устройстве. В нем описываются классы и события, которые может использовать приложение, а также все разрешения, необходимые для его работы. Так, если приложению необходим доступ к передачи данных – например, чтобы загрузить файл, – соответствующее разрешение должно быть явно указано в файле – манифесте. Это конкретное разрешение могут иметь многие приложения. Подобная защита, подразумевающая описание всех допустимых разрешений, допускаемых для приложения, снижает риск ошибок ввиду неправильно написанного кода или же чтобы оградить устройство от несанкционированного доступа, так как в процессе установки приложения

пользователь видит, к каким службам оно будет иметь возможность обращаться.

Сама программа строится из файлов– классов и табличных– xml файлов, а также содержит несколько версий иконки приложения, для разных разрешений экрана. В java – файлах содержится основной код программы, в xml – файлах – описание интерфейса, хранятся параметры программы, переменные, текстовые данные и т.д.

В файле «build.gradle» хранятся инструкции по автоматической сборке проекта, что очень удобно работе с проектами, имеющими в своей структуре несколько файлов.

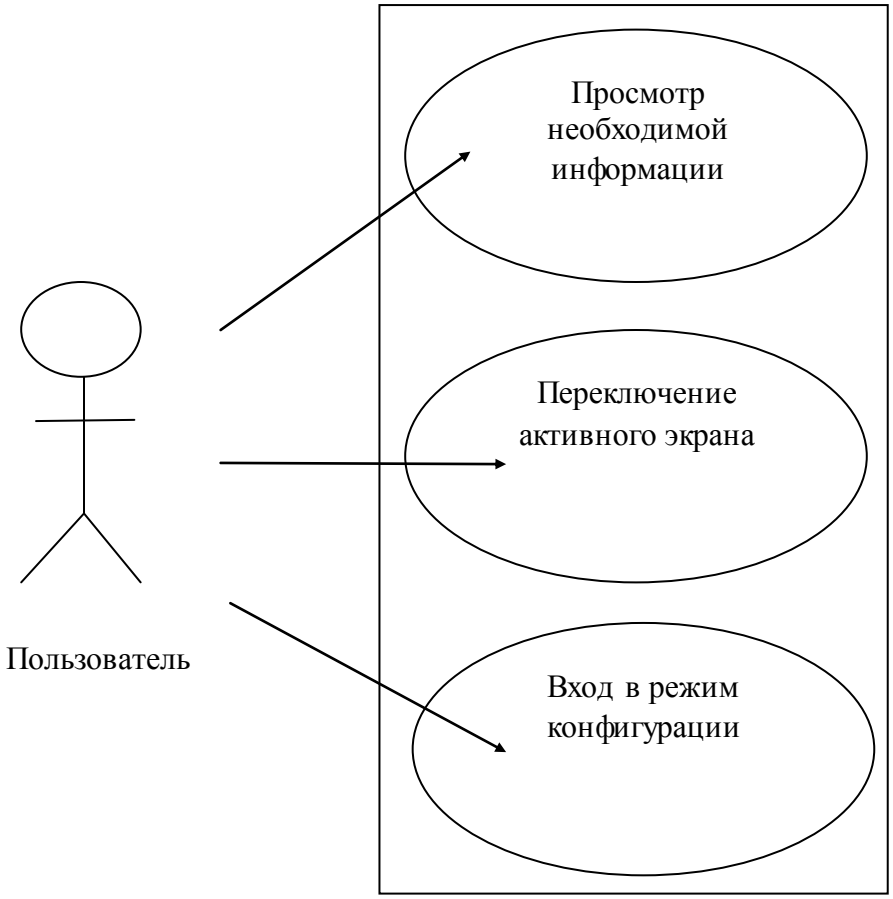

Рисунок 2.5 – Диаграмма вариантов использования

2.3 Обзор использованных методов

Абстрактные классы – это классы, содержащие шаблоны методов. Таким образом производные классы могут наследовать методы абстрактного класса, что очень удобно при создании нескольких классов схожего функционала. Например, у нас есть абстрактный класс «TemperatureCommand.java» (рисунок 2.6), который описывает методы, используемые во всех классах, используемых для обработки информации, получаемой с датчиков температуры.

```
public abstract class TemperatureCommand extends ObdCommand implements
        SystemOfUnits {

    private float temperature = 0.0f;

    public TemperatureCommand(String cmd) {
        super(cmd);
    }

    public TemperatureCommand(TemperatureCommand other) {
        super(other);
    }

    @Override
    protected void performCalculations() {
        // ignore first two bytes [hh hh] of the response
        temperature = buffer.get(2) - 40;
    }

    @Override
    public String getFormattedResult() {
        return useImperialUnits ? String.format("%.1f%s", getImperialUnit(), getResultUnit())
                : String.format("%.0f%s", temperature, getResultUnit());
    }

    @Override
    public String getCalculatedResult() {
        return useImperialUnits ? String.valueOf(getImperialUnit()) : String.valueOf(temperature);
    }

    @Override
    public String getResultUnit() {
        return useImperialUnits ? "F" : "C";
    }

    public float getTemperature() {
        return temperature;
    }

    public float getImperialUnit() {
        return temperature * 1.8f + 32;
    }

    public float getKelvin() {
        return temperature + 273.15f;
    }

    public abstract String getName();

}
```

Рисунок 2.6 – Реализация абстрактного класса «TemperatureCommand»

Создав абстрактный класс «TemperatureCommand», мы можем использовать описанные в нем методы во всех подклассах («AmbientAirTemperatureCommand», «EngineCoolantTemperatureCommand»,

«AirIntakeTemperatureCommand»). Благодаря тому, что в абстрактном классе были описаны методы конвертации температуры в различные единицы измерения, информация о температуре, полученная любым из вышеуказанных запросов, может быть автоматически конвертирована в выбранную единицу измерения.

Запрос от приложения к ЭБУ автомобиля на примере запроса информации о температуре окружающего воздуха:

```
public class AmbientAirTemperatureCommand extends TemperatureCommand {

    public AmbientAirTemperatureCommand() {
        super("01 46");
    }

    public AmbientAirTemperatureCommand(TemperatureCommand other) {
        super(other);
    }
    @Override
    public String getName() {
        return AvailableCommandNames.AMBIENT_AIR_TEMP.getValue();
    }

}
```

Рисунок 2.7 – Реализация класса «AmbientAirTemperatureCommand»

В данном классе, наследующем методы от абстрактного класса «TemperatureCommand» мы ассоциируем строку «AmbientAirTemperatureCommand» с командой «01 46» для ЭБУ, тем самым, каждый раз, когда будет производиться запрос значения строки «AmbientAir–TemperatureCommand», программа будет предоставлять информацию о температуре окружающего воздуха, полученную от бортовых систем автомобиля.

С помощью ключевого слова «super» мы можем определить класс, от которого текущий класс будет наследовать методы. Вызов метода «super» всегда должен быть первым оператором, выполняемым внутри конструктора подкласса.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

Рисунок 2.8 – Блок кода, обьявляющего класс как суперкласс

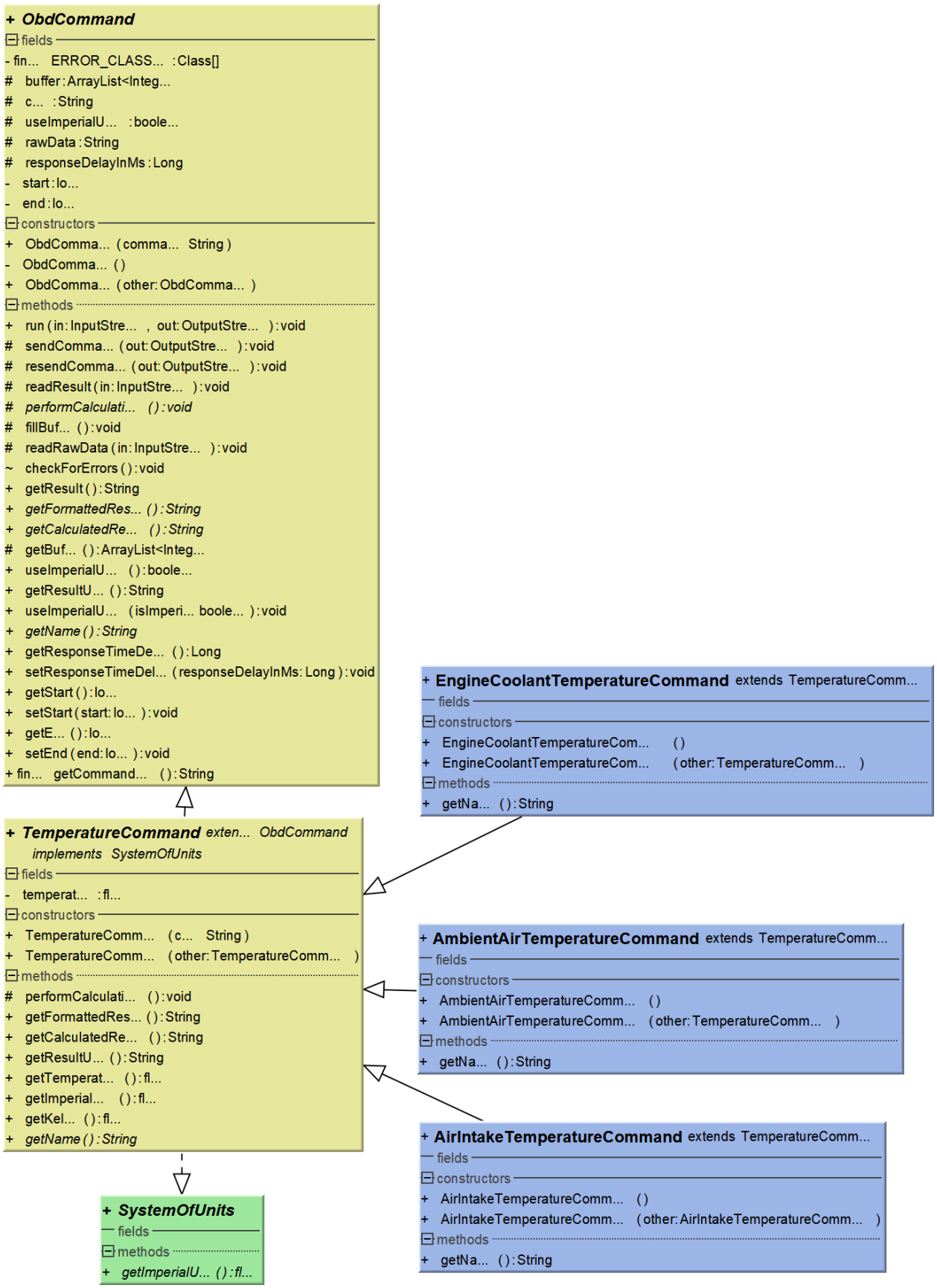

Рисунок 2.9 – Диаграмма классов, взаимодействующих с температурой

Конвертация данных из шестнадцатеричной системы исчисления, на которой работает ЭБУ в десятеричную систему исчисления на примере считывания VIN – кода автомобиля (рисунок 2.8)

```
public String convertHexToString(String hex) {
    StringBuilder sb = new StringBuilder();
        for (int i = 0; i < hex.length() - 1; i += 2) {

        //проход по шестнадцатеричной строке
        String output = hex.substring(i, (i + 2));
        //преобразование шестнадцатеричных единиц в десятеричные
        int decimal = Integer.parseInt(output, 16);
        // преобразование десятеричных единиц в символы
        sb.append((char) decimal);
    }
    return sb.toString();
}
```

Рисунок 2.10 – Фрагмент кода, выполняющего конвертацию из шеснадцатеричной системы в строку

Также, при работе программы могут возникнуть нештатные ситуации и если программа не способна на них корректно реагировать, то работа программы попросту завершится. Для предотвращения подобных ситуаций существуют исключения. Операторы программы, которыенербходимо отслеживать, помещаются в блок «try». Если исключение произошло, то оно создаётся и передаётся дальше. Программный код способен перехватить исключение при помощи блока «catch» и обработать его. Системные исключения автоматически передаются самой системой. Чтобы передать исключение вручную, используется «throw». Любое исключение, созданное и передаваемое внутри метода, должно быть указано в его интерфейсе ключевым словом «throws». Любой код, который следует выполнить обязательно после завершения блока «try», помещается в блок «finally».

```java
public static float getGpsDistanceUpdatePeriod(SharedPreferences prefs) {
    String periodString = prefs
            .getString(ConfigActivity.GPS_DISTANCE_PERIOD_KEY, "5"); // 5 as in meters
    float period = 5; // by default 5 meters

    try {
        period = Float.parseFloat(periodString);
    } catch (Exception e) {
    }

    if (period <= 0) {
        period = 5;
    }

    return period;
}
```

Рисунок 2.11 – Пример использования исключений

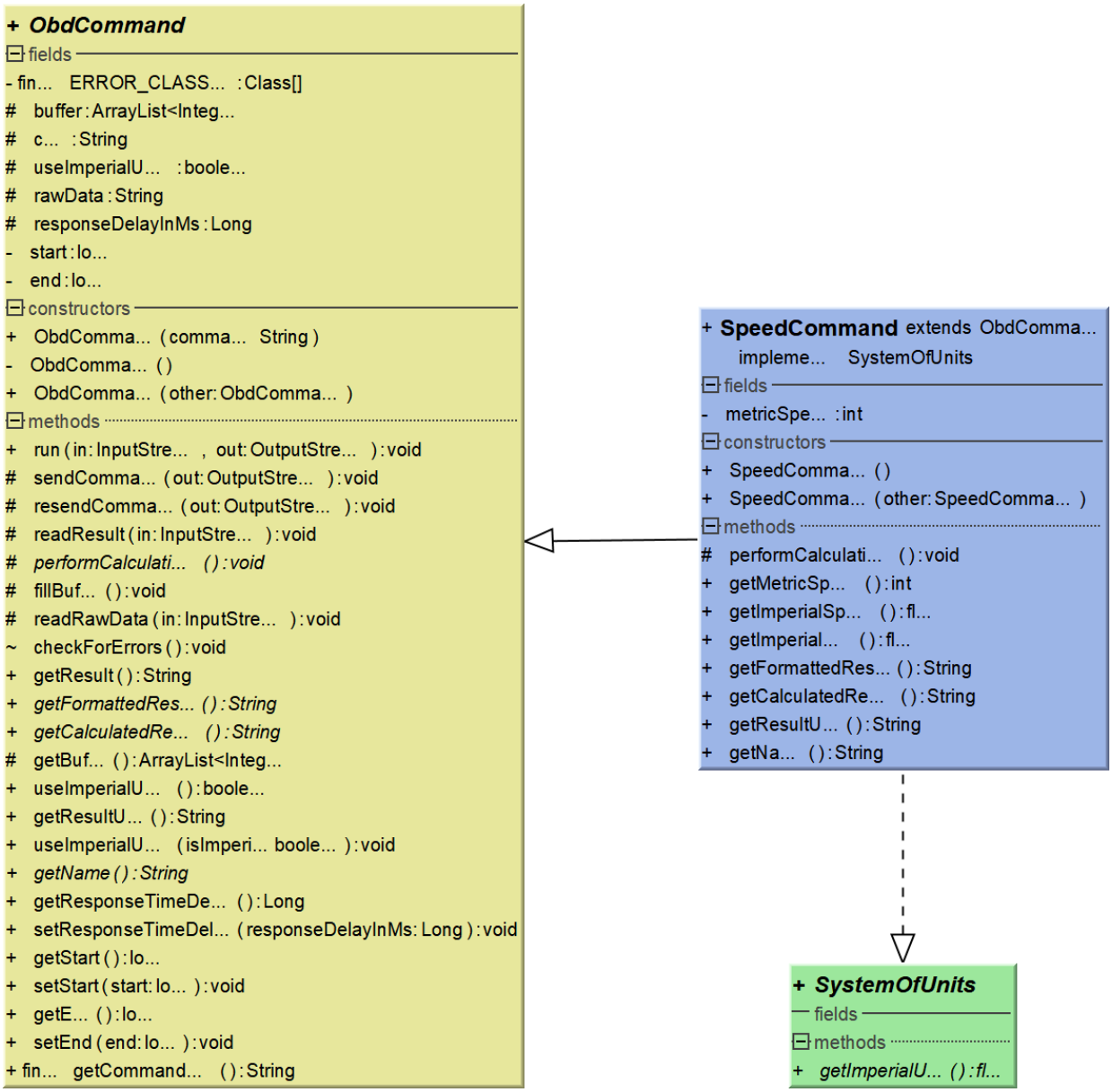

Рисунок 2.12 – Диаграмма классов, взаимодействующих с информацией о скорости

```
switch (msg.what) {
                case NO_BLUETOOTH_DEVICE_SELECTED:
                    makeToast(getString(R.string.text_bluetooth_nodevice));
                    finish();
                    break;
                case CANNOT_CONNECT_TO_DEVICE:
                    makeToast(getString(R.string.text_bluetooth_error_connecting));
                    finish();
                    break;

                case OBD_COMMAND_FAILURE:
                    makeToast(getString(R.string.text_obd_command_failure));
                    finish();
                    break;
                case OBD_COMMAND_FAILURE_IO:
                    makeToast(getString(R.string.text_obd_command_failure) + " IO");
                    finish();
                    break;
                case OBD_COMMAND_FAILURE_IE:
                    makeToast(getString(R.string.text_obd_command_failure) + " IE");
                    finish();
                    break;
                case OBD_COMMAND_FAILURE_MIS:
                    makeToast(getString(R.string.text_obd_command_failure) + " MIS");
                    finish();
                    break;
                case OBD_COMMAND_FAILURE_UTC:
                    makeToast(getString(R.string.text_obd_command_failure) + " UTC");
                    finish();
                    break;
                case OBD_COMMAND_FAILURE_NODATA:
                    makeToastLong(getString(R.string.text_noerrors));
                    //finish();
                    break;

                case NO_DATA:
                    makeToast(getString(R.string.text_dtc_no_data));
                    ///finish();
                    break;
                case DATA_OK:
                    dataOk((String) msg.obj);
                    break;

            }
            return false;
```

Рисунок 2.13 – Пример кода с использованием оператора «switch»

Если программа должна в зависимости от входных данных выбирать ответное действие, можно применить оператор «switch». Он работает следующим образом: вычисленное значение выражения сравнивается со всеми значениями, указанными в операторах «case», если при этом находится оператор «case» со значением, которое совпадает со значением выражения, управление передается стоящему за ним (после двоеточия) коду. Если же значению выражения не соответствует ни один из операторов «case», управление передается коду, расположенному после ключевого слова «default». Стоит отметить, что оператор «default» необязателен. В случае, когда ни один из операторов «case» не соответствует значению выражения и

при этом в «switch» отсутствует оператор «default», выполнение программы продолжается с оператора, следующего за оператором «switch».

Внутри оператора «switch» (а также внутри циклических конструкций) «break» без метки приводит к передаче управления на код, стоящий после оператора «switch». Если «break» отсутствует, после текущего раздела «case» будет выполняться следующий. Иногда бывает удобно иметь в операторе «switch» несколько смежных разделов «case», не разделенных оператором «break».

Для вывода круговых шкал, отображающих параметры, имеющие максимальные значения, необходимо получить выходные данные в процентах.

```
public abstract class PercentageObdCommand extends ObdCommand {

    protected float percentage = 0f;

    public PercentageObdCommand(String command) {
        super(command);
    }

    public PercentageObdCommand(PercentageObdCommand other) {
        super(other);
    }

    @Override
    protected void performCalculations() {
        percentage = (buffer.get(2) * 100.0f) / 255.0f;
    }

    @Override
    public String getFormattedResult() {
        return String.format("%.1f%s", percentage, getResultUnit());
    }

    public float getPercentage() {
        return percentage;
    }

    @Override
    public String getResultUnit() {
        return "%";
    }

    @Override
    public String getCalculatedResult() {
        return String.valueOf(percentage);
    }

}
```

Рисунок 2.14 – Фрагмент кода, формирующего вывод значения параметра в процентах

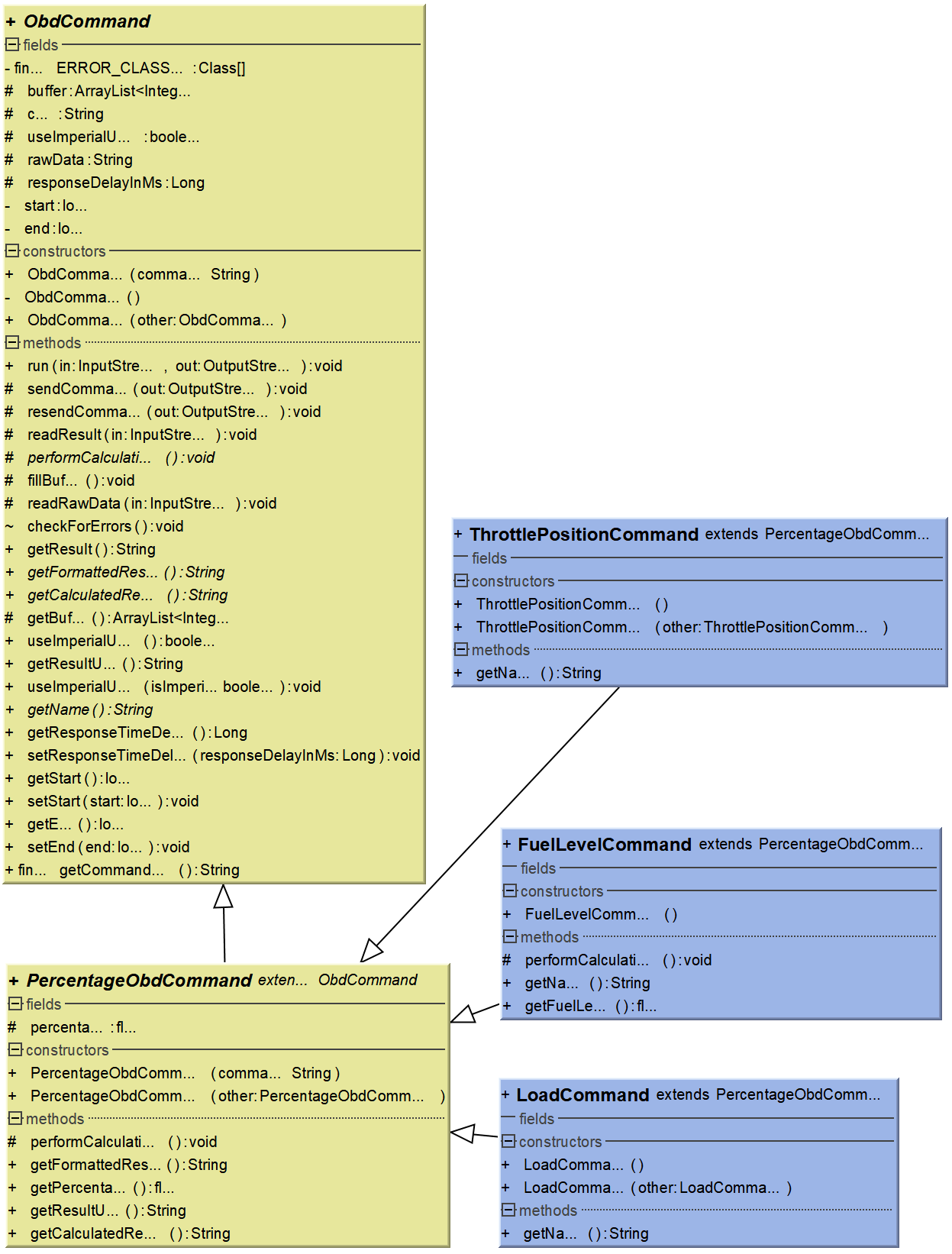

Рисунок 2.15 – Диаграмма классов, наследующих методы вывода показаний в процентах

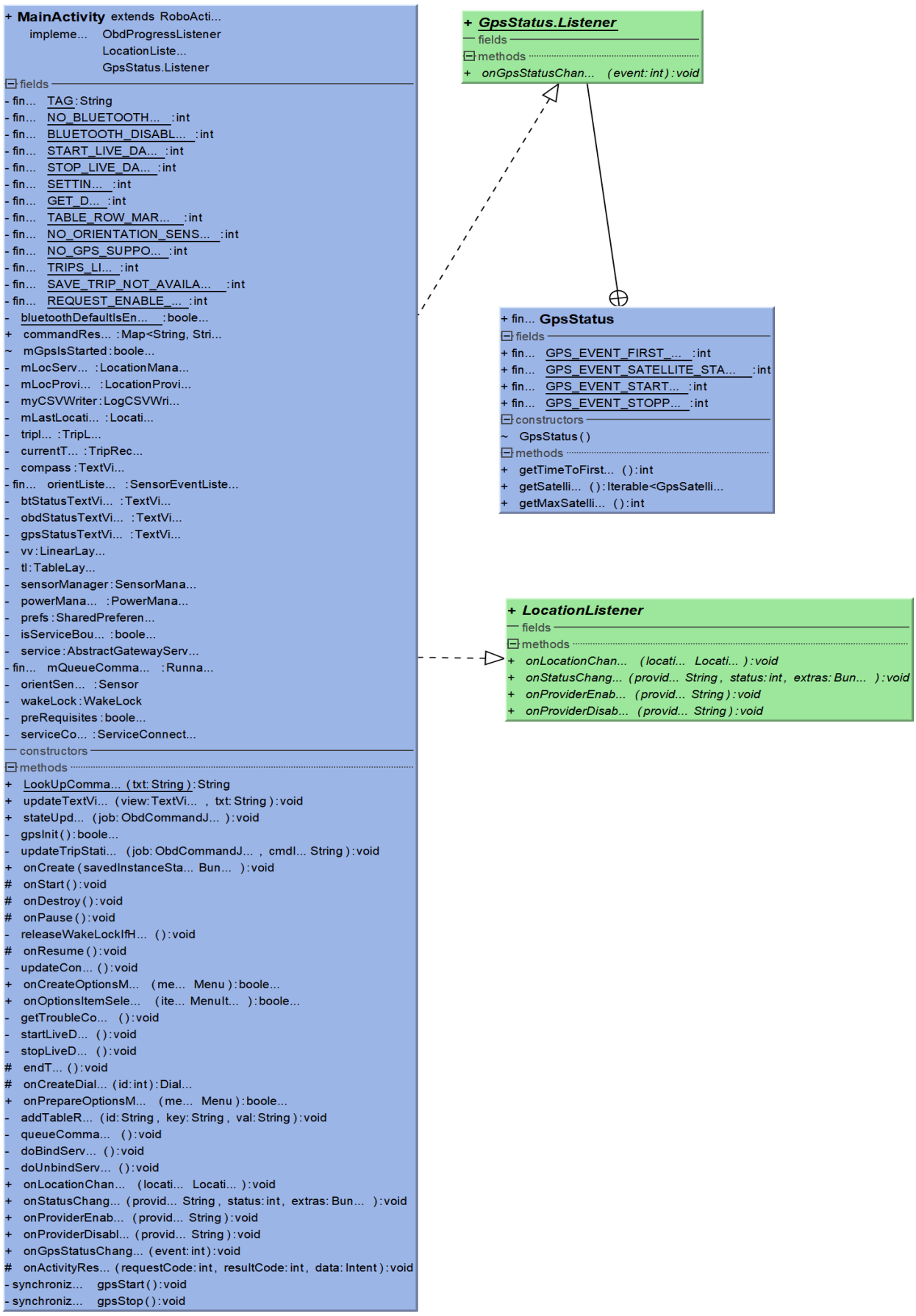

Рисунок 2.16 – Диаграмма классов, взаимодействующих с модулем GPS считыванием координат

2.4 Разработка интерфейса приложения

Следующей задачей было разработать интерфейс, максимально простой и удобный для конечного пользователя. Для этого ознакомимся с требованиями к интерфейсу. В пункте «1.3 Анализ актуальных решений и выявление их достоинств и недостатков. Представление основных отличий и разработка концепции интерфейса» был разработан концептуальный макет интерфейса, через который необходимо предоставить пользователю информацию, требования к которой установлены в пункте «1.2 Формирование требований к программному продукту». В качестве инструмента для реализации этих целей мною был использован язык xml и редактор, входящий в состав Android Studio, что сэкономило мне время и сильно упростило выполнение поставленных задач, так как параллельно писать код и подстраивать под него интерфейс в одном приложении, всего лишь переключаясь между вкладками оказалось невероятно удобно. Так же в Android Studio по умолчанию встроены типовые элементы интерфейса, такие как кнопки, переключатели, флажки, поля для пользовательских данных (например, номер телефона или e – mail) и т.д.

После принятия решения об визуальном оформлении, интерфейсе и предоставляемой пользователю информации, разработаем графический интерфейс программы:

Рисунок 2.15 – Экран (№1) «Приборная панель»

1. «Приборная панель» (рисунок 2.15) – Экран с выводом информации, необходимой для контроля над автомобилем во время движения. В центре тахометр, справа и слева от него блоки с информацией. Отображает пользователю такие данные как:

1.1. Скорость (текущая скорость автомобиля);

1.2. Температура (температура жидкости, охлаждающей двигатель);

1.3. Внешняя температура (температура воздуха, попадающего в воздухозаборник);

1.4. Мощность двигателя (сила, передающаяся от двигателя к колесам);

1.5. Расход топлива (Объём топлива, затрачиваемого на 100 километров пути);

1.6. Уровень топлива (Процент заполнения топливного бака);

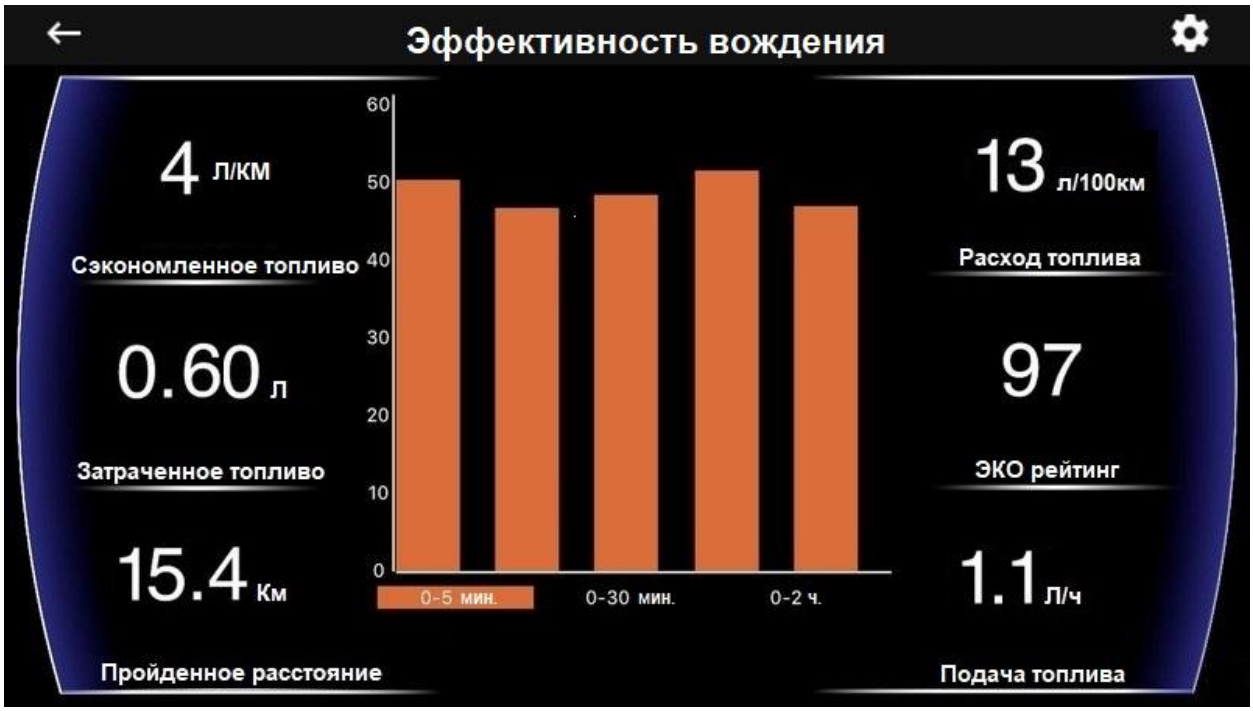

Рисунок 2.16 – Экран (№2) «Эффективность вождения»

2. «Эффективность вождения» (рисунок 2.16) – Экран с выводом информации, необходимой для просмотра статистики расходов на езду и экономичности вождения. В центре диаграмма расхода топлива, справа и слева от него блоки с информацией. Отображает пользователю такие данные как:

2.1. Сэкономленное топливо (снижение среднего расхода топлива, относительно прошлой поездки);
2.2. Затраченное топливо (объём топлива, затраченного на поездку);
2.3. Пройденное расстояние (расстояние, пройденное с начала поездки);
2.4. Расход топлива (объём топлива, затрачиваемого на 100 километров пути);
2.5. ЭКО рейтинг (сравнительная оценка стиля вождения);
2.6. Подача топлива (объём топлива, подающегося в единицу времени, из топливного бака в двигатель);

Рисунок 2.17 – Экран (№3) «Экономичность вождения»

3. «Экономичность вождения» (рисунок 2.17) – Экран с выводом информации, необходимой для просмотра оценки экономичности и экологичности вождения. В центре оценка вождения, справа и слева от него блоки с информацией. Отображает пользователю такие данные как:

3.1. Кол– во топлива (заполненный объём в топливном баке);

3.2. Пройденное расстояние (расстояние, пройденное с начала поездки);

3.3. Подача топлива (объём топлива, подающегося в единицу времени, из топливного бака в двигатель);

3.4. Затраченное топливо (объём топлива, затраченного на поездку);

3.5. Оставшийся путь (расстояние, доступное для преодоления, при текущем количестве топлива баке и текущем среднем расходе топлива);

3.6. Выброс CO2 (масса углекислого газа, выбрасываемого автомобилем);

Так как при разработке мы не располагали необходимыми аппаратными средствами, переключение между режимами приложения

осуществляется пролистываниями экрана, а доступ к отладочным настройкам были реализованы через контекстное меню. В реальном автомобиле переключение между экранами приборной панели осуществлялось бы при помощи внешних органов управления, например, кнопок на руле.

Так же имеется возможность доступа в режим конфигурации, но реальной приборной панели доступ туда осуществлялся бы какой– либо сложной последовательностью действий, или же вообще был бы заблокирован для рядового пользователя. В нем можно выбрать систему мер, используемую для вывода данных, выбрать команды для OBD и так далее.

Выводы по главе 2: в главе был проведен обзор необходимого инструментария, рассмотрены использованные методы разработки, на основе концептуальной модели, спроектированной в главе 1, был разработан пользовательский интерфейс.

3 ТЕСТИРОВАНИЕ ЦИФРОВОЙ ПРИБОРНОЙ ПАНЕЛИ

3.1 Обоснование технологии тестирования

Тестирование программного обеспечения – это оценка разрабатываемого программного обеспечения, чтобы проверить его возможности, способности и соответствие ожидаемым результатам. Существуют различные типы методов, используемые в области тестирования и обеспечения качества.

Широко используемыми методами тестирования являются модульное тестирование, интеграционное тестирование, приемочное тестирование, и тестирование системы.

Поскольку элементы разработанного программного продукта тесно связаны между собой и постоянно передают друг – другу различные данные, наилучшими методам тестирования будет системное тестирование.

Таким образом, сначала необходимо будет проверить работоспособность каждого модуля программы – Подключения к модулям GPS и Bluetooth и адаптеру OBD. После этого необходимо будет проверить то, насколько корректно передается информация от одного модуля к другому.

Приложение было предназначено для получения информации с диагностического разъёма и выводу данных на экран в реальном времени. Так как целью своей разработки я ставил саму идею, а предложенные мною функции можно реализовать другими методами, будет использован принцип черного ящика.

Тестирование методом черного ящика осуществляется без каких – либо знаний внутренней работы системы. Этот тест также известен как Black – box, closed – box тестирование или функциональное тестирование.

3.2 Разработка плана тестирования

Список тестируемых функций:

1. Функция установления подключения к GPS.
2. Функция установления подключения к ЭБУ.
3. Работоспособность контекстного меню.
4. Работоспособность переходов между окнами программы.
5. Функция смены системы мер на имперскую и обратно.
6. Функция вывода текущих оборотов двигателя.
7. Функция отсчета времени поездки.

Разработав план тестирования можно приступать непосредственно к тестированию и составлению отчета о тестировании.

3.3 Проведение тестирования и составление отчета о тестировании

Таблица 2 – Протокол тестирования

Тестируемая функция	Действие	Ожидаемый результат	Полученный результат
Функция установления подключения к GPS	Установление подключения с включенным GPS	Подключение установлено	выполнено
Функция установления подключения к GPS	Установление подключения с выключенным GPS	Вывод сообщения: «ваше устройство не поддерживает GPS, или GPS выключен	выполнено
Функция установления подключения к Bluetooth	Установление подключения с включенным Bluetooth	Подключение установлено	выполнено
Функция установления подключения к Bluetooth	Установление подключения с выключенным Bluetooth	Вывод запроса за подключение Bluetooth	выполнено
Функция установления подключения к Bluetooth	Установление подключения с выключенным Bluetooth и последующем отказе в подключении.	Вывод сообщения: У вас выключен Bluetooth. Пожалуйста, включите его	выполнено

Таблица 2 – Протокол тестирования(продолжение)

Тестируемая функция	Действие	Ожидаемый результат	Полученный результат
Тест корректности компаса	Вращение телефона вокруг вертикальной оси	Индикация верных сторон света	выполнено
Тест работоспособности контекстного меню	Вызов контекстного меню	Вывод контекстного меню	выполнено
Работоспособность переходов между окнами программы	Смена окон при помощи контекстного меню и возвращение на главный экран кнопкой «назад»	Переход между связанными окнами происходит корректно	выполнено

Таблица 2 – Протокол тестирования(продолжение)

Тестируемая функция	Действие	Ожидаемый результат	Полученный результат
Функция смены системы мер на имперскую и обратно	Запрос кодов ошибок из памяти ЭБУ	Получение и вывод списка ошибок	выполнено
Функция отсчета времени поездки	Старт системы	После активации записи поездки, на главный экран должен выводится отсчет времени	выполнено
Функция вывода текущих оборотов двигателя	Изменение оборотов двигателя, при включенной записи поездки	главный экран должны выводится данные о текущих оборотах двигателя	выполнено

Исходя из полученных результатов тестирования можно сделать вывод, что программа работает так, как задумано. Программный продукт корректно выполняет все функции, которые были протестированы, ошибок при тестировании не обнаружено.

Таким образом, все заявленные функции работают без ошибок.

Тестирование работоспособности приложения проводилось на устройствах:

— ZTE V975 (платформа intel, экран 720p, ОС Android 4.2);

— Xiaomi MI5 (платформа Qualcomm, экран 1080p. ОС Android 7.0);

LG Nexus 4 (платформа Qualcomm, экран 720p. ОС Android 7.1) (эмуляция);

— ASUS Zenfone 5 (платформа intel, экран 720p, ОС Android 4.4).

На всех устройствах в составе тестовой группы, разработанное в ходе данной работы, приложение исполнялось без нареканий и проходило все сценарии проверок без ошибок.

Выводы по главе 3: в настоящей главе были составлены критерии для тестирования приложения, и на их основе была протестирована работа этого приложения. По итогам тестирования была дана оценка результатам тестирования.

ЗАКЛЮЧЕНИЕ

Темой работы была выбрана: «разработка цифровой приборной панели для автомобиля». В ходе исследования предметной области была сформулированная и установлена цель работы. Цель данной работы заключалась в разработке идеи адаптируемого, модульного ПО для использования в качестве приборной панели автомобиля.

Для достижения поставленной цели было необходимо изучить технологии, применяемые для установления связи между автомобилем и устройством пользователя, методы разработки приложений для Android, способы тестирования и применить все полученные знания на практике, спроектировав и написав приложение, выполняющее требуемые функции.

По итогам выполненной работы можно сказать что все поставленные задачи были выполнены в полном объёме, а тема ВКР полностью раскрыта. Разработанная программа выполняет все необходимые задачи, полностью работоспособна, готова к использованию конечным пользователем.

Эта концепция является перспективным направлением для дальнейших исследований.

«Честно говоря, я вижу, что аппаратные циферблаты исчезают и мне это нравится». – Гордон Вагенер, руководитель проекта Mercedes – Benz.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Souza B., Yanaga E. Open Source Tools for Java Deployment, 2014
2. Griffiths, D. Head First Android Development / Dawn Griffiths, David Griffiths. – O'Reilly Media, 2015. – 734 p.
3. Lee, W– M. Beginning Android Tablet Application Development / Wei– Meng Lee. – WROX, 2011. – 288 p.
4. Eder L. 10 Things You Didn't Know About Java, 2014
5. Mednieks Z., Dornin L, Meike G. B., Nakamura M. Programming Android – O'Reilly, 2011.
6. Delessio, C. Hello, Android Application Development in 24 Hours, Sams Teach Yourself / Carmen Delessio. – Sams Publishing, 2013. – 448 p.
7. Peter Lawrey //Java Lambdas and Low Latency, 2015
8. Raoul– Gabriel Urma // Alternative Languages for the JVM, 2014
9. Smyth N. Android Studio Development Essentials – Android 6 Edition – Payload Media, 2015.
10. Chan, J. Java: Learn Java in One Day and Learn It Well. Java for Beginners with Hands– on Project / Jamie Chan. – LCF Publishing. 2016. –236 p.
11. Oaks, S. Java Performance: The Definitive Guide: Getting the Most Out of Your Code 1st Edition / Scott Oaks. – O'Reilly Media, 2014. – 426 p.
12. Jinseong K. Efficient Protection of Android Applications through User Authentication Using Peripheral Devices, 2018.
13. Lindholm T., Yellin F., Bracha G., Buckley A. Inside Java: The Java Virtual Machine, 2015.
14. The #1 open – source Android GNSS/GPS test program, GitHub [Электронный ресурс]: https://github.com/barbeau/gpstest (дата обращения: 19.05.18)

15. Управление данными [Электронный ресурс]: учебное пособие / Ю.Ю. Громов, О.Г. Иванова, А.В. Яковлев, В.Г. Однолько; Министерство образования и науки Российской Федерации, Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Тамбовский государственный технический университет». – Тамбов: Издательство ФГБОУ ВПО «ТГТУ», 2014. – 192 с.: ил – Библиогр. в кн. – ISBN 978– 5 – 8265 – 1374 – 3

16. Android Developers [Электронный ресурс] // Google – Электрон. дан. – [Б. м.], 2016.– URL – https://developer.android.com/index.html (дата обращения 02.05.2018). Библиотека программиста [Электронный ресурс]: https://proglib.io/ (дата обращения:21.06.18)

17. Android OBD– II Reader application that uses pure OBD– II PID's Java API, GitHub [Электронный ресурс]: https://github.com/pires/android– obd– reader (дата обращения: 20.05.18)

18. Сорокин, А.А. Объектно – ориентированное программирование [Электронный ресурс]: учебное пособие (курс лекций) / А.А. Сорокин; Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Северо – Кавказский федеральный университет», 47 Министерство образования и науки Российской Федерации. – Ставрополь: СКФУ, 2014. – 174 с.

19. Дейтел П. Андроид для разработчиков, 3– е издание/П. Дейтел, Х. Дейтел, А. Уолд. – Питер, 2016. – 512 с.

20. Дентон Т. Автомобильная электроника / Дентон Т. – НТ Пресс, 2008. – 576 с.

21. Лафоре Р. Структуры данных и алгоритмы Java; Пер. с англ. – Е. Матвеев – СПб.: Питер, 2016. – 704 с.

22. Управление данными [Электронный ресурс]: учебник / Ю.Ю. Громов, О.Г. Иванова, А.В. Яковлев, В.Г. Однолько; Министерство образования и науки Российской Федерации, Федеральное государственное

бюджетное образовательное учреждение высшего профессионального образования «Тамбовский государственный технический университет». – Тамбов: Издательство ФГБОУ ВПО «ТГТУ», 2015. – 192 с.: ил., табл., схем. – Библиогр. в кн.. – ISBN 978 – 5 – 8265– 1385 – 9

23. Смирнов Ю. Электронные и микропроцессорные системы управления автомобилей / Ю. Смирнов, А. Муханов. – Лань 2012. – 624 с.

24. Учебник и энциклопедия для разработчика [Электронный ресурс]: developer.alexanderklimov.ru/android/ (дата обращения: 28.02.18)

25. Управление данными [Электронный ресурс]: учебное пособие / Ю.Ю. Громов, О.Г. Иванова, А.В. Яковлев, В.Г. Однолько; Министерство образования и науки Российской Федерации, Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Тамбовский государственный технический университет». – Тамбов: Издательство ФГБОУ ВПО «ТГТУ», 2014. – 192 с.: ил – Библиогр. в кн. – ISBN 978– 5 – 8265 – 1374 – 3

26. Шматко А.В., Федорченко В.Н. Обзор и анализ инструментов разработки мобильных приложений для ОС Android // Инновации в науке: сб. ст. по матер. LVII междунар. науч. – практ. конф. № 5(54). Часть I. – Новосибирск: СибАК, 2016. – С. 59 – 73.

ПРИЛОЖЕНИЕ А

Содержимое файла «RPMCommand.java»

```
public class RPMCommand extends ObdCommand {

    private int rpm = -  1;

    public RPMCommand() {
        super("01 0C");
    }

    public RPMCommand(RPMCommand other) {
        super(other);
    }

    @Override
    protected void performCalculations() {
        // ignore first two bytes [41 0C] of the response((A*256)+B)/4
        rpm = (buffer.get(2) * 256 + buffer.get(3)) / 4;
    }

    @Override
    public String getFormattedResult() {
        return String.format("%d%s", rpm, getResultUnit());
    }

    @Override
    public String getCalculatedResult() {
        return String.valueOf(rpm);
    }

    @Override
    public String getResultUnit() {
        return "RPM";
    }

    @Override
    public String getName() {
        return AvailableCommandNames.ENGINE_RPM.getValue();
    }

    public int getRPM() {
        return rpm;
    }

}
```

ПРИЛОЖЕНИЕ Б

Содержимое файла «SpeedCommand.java»

```
public class SpeedCommand extends ObdCommand implements SystemOfUnits {

    private int metricSpeed = 0;


    public SpeedCommand() {
        super("01 0D");
    }

    public SpeedCommand(SpeedCommand other) {
        super(other);
    }

    @Override
    protected void performCalculations() {

        metricSpeed = buffer.get(2);
    }
    public int getMetricSpeed() {
        return metricSpeed;
    }


    public float getImperialSpeed() {
        return getImperialUnit();
    }

    public float getImperialUnit() {
        return metricSpeed * 0.621371192F;
    }

    public String getFormattedResult() {
        return useImperialUnits ? String.format("%.2f%s", getImperialUnit(),
getResultUnit())
                : String.format("%d%s", getMetricSpeed(), getResultUnit());
    }

    @Override
    public String getCalculatedResult() {
        return useImperialUnits ? String.valueOf(getImperialUnit()) :
String.valueOf(getMetricSpeed());
    }

    @Override
    public String getResultUnit() {
        return useImperialUnits ? "mph" : "km/h";
    }

    @Override
    public String getName() {
        return AvailableCommandNames.SPEED.getValue();
    }

}
```

ПРИЛОЖЕНИЕ В

Содержимое файла «main.xml»

```
      <?xml version="1.0" encoding="utf-  8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/vehicle_view"
    android:layout_with="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="@dimen/activity_horizontal_margin"
    android:orientation="vertical">

    <TableRow
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">

        <TextView
            android:id="@+id/textView"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="2"
            android:tag="SPEED"
            android:text="@string/text_zero"
            android:textSize="@dimen/abc_text_size_display_3_material" />

        <TextView
            android:id="@+id/compass_text"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:gravity="end"
            android:text="@string/text_orientation_default"
            android:textSize="@dimen/abc_text_size_display_3_material" />
    </TableRow>

    <TableRow
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">

        <TextView
            android:id="@+id/textView4"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:gravity="center"
            android:tag="FUEL_CONSUMPTION"
            android:text="@string/text_consumption_default" />

        <TextView
            android:id="@+id/textView3"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:gravity="center"
            android:tag="ENGINE_RUNTIME"
            android:text="@string/text_runtime_default" />

        <TextView
            android:id="@+id/textView2"
            android:layout_width="0dp"
```

```
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:gravity="center"
        android:tag="ENGINE_RPM"
        android:text="" />

</TableRow>

<ScrollView
    android:id="@+id/data_scroll"
    android:layout_width="fill_parent"
    android:layout_height="0dp"
    android:layout_gravity="top"
    android:layout_margin="10dp"
    android:layout_weight="1"
    android:elevation="2dp"
    android:outlineProvider="bounds">

    <TableLayout
        android:id="@+id/data_table"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:stretchColumns="*"></TableLayout>
</ScrollView>

<TableRow
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">

    <TextView
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:gravity="center"
        android:text="@string/text_gps"
        android:textSize="@dimen/abc_text_size_medium_material" />

    <TextView
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:gravity="center"
        android:text="@string/text_bluetooth"
        android:textSize="@dimen/abc_text_size_medium_material" />

    <TextView
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:gravity="center"
        android:text="@string/text_obd"
        android:textSize="@dimen/abc_text_size_medium_material" />


</TableRow>

<TableRow
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">

    <TextView
        android:id="@+id/GPS_POS"
        android:layout_width="0dp"
```

```
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:gravity="center"
            android:text="" />

        <TextView
            android:id="@+id/BT_STATUS"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:gravity="center"
            android:text="" />

        <TextView
            android:id="@+id/OBD_STATUS"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:gravity="center"
            android:text="" />
    </TableRow>

</LinearLayout>
```