Федеральное государственное бюджетное образовательное учреждение   
высшего образования

|  |  |
| --- | --- |
| Gerb-BMSTU_01 | «Московский государственный технический университет  им. Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана) |

ФАКУЛЬТЕТ – Информатика и управления

КАФЕДРА – Информационные системы и телекоммуникации

РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по дисциплине

Разработка программного обеспечения

на тему

Разработка OSGI сервиса мониторинга сетевых Ethernet интерфейсов

Студент группы ИУ3-73 (подпись) 28.12.2017 А.М Петров

Руководитель курсовой работы (подпись) 28.12.2017 А.М Иванов

Москва, 2017

Реферат

В работе предложена архитектура OSGI компонента мониторинга сетевых интерфейсов. Архитектура реализована с использованием объектно-ориентированного языка программирования Java. В целях проверки работоспособности программного обеспечения разработана методика тестирования с помощью библиотеки JUnit. Результаты проведенных тестов доказывают функциональность OSGI компонента.

Содержание

[Введение 5](#_Toc506539821)

[1 Исследовательская часть 6](#_Toc506539822)

[1.1 Выбор и обоснование способа работы с сетевыми интерфейсами 6](#_Toc506539823)

[1.2 Обзор функциональных возможностей класса NetworkInetrface 6](#_Toc506539824)

[2 Конструкторская часть 8](#_Toc506539825)

[2.1 Техническое задание 8](#_Toc506539826)

[2.2 Выбор технических решений, удовлетворяющих интересам ЗС 8](#_Toc506539827)

[2.3 Структура проекта 8](#_Toc506539828)

[2.3.1 Плагин com.bmstu.coursework.ethernet.manager 9](#_Toc506539829)

[2.4 Диаграмма компонентов 12](#_Toc506539830)

[2.5 Диаграмма классов 12](#_Toc506539831)

[2.6 Алгоритм работы ПО 12](#_Toc506539832)

[3 Технологическая часть 13](#_Toc506539833)

[3.1 Запуск разработанного приложения 13](#_Toc506539834)

[3.2 Анализ искходного кода с помощью метрик качества 13](#_Toc506539835)

[3.3 Анализ зависимостей в коде системы 14](#_Toc506539836)

[3.4 Тестирование на корректность работы 15](#_Toc506539837)

[3.5 Реализация Junit тестов 16](#_Toc506539838)

[3.6 Оценка покрытия кода 17](#_Toc506539839)

[Заключение 19](#_Toc506539840)

[Список литературы 20](#_Toc506539841)

Введение

Вычислительные системы в наше время достигли определенного уровня сложности, при котором человеческих возможностей для их запуска и поддержания становится недостаточно. Решением данной проблемы стала автономность и автоматизация некоторых процессов. Одна из главных задач автономных вычислений – улучшение вычислительных систем при снижении фактора человеческого участия.

В самоуправляемых автономных системах человек, как оператор, приобретает совершенно новую роль: вместо прямого управления системой, он определяет общие политики и правила, которые определяют процесс самоконтроля.

В данной курсовой работе проектируется как раз система описанного выше типа, используемая для мониторинга состояний сетевых интерфейсов.

1. Исследовательская часть

Настоящий раздел является теоретическим и включает:

* Выбор способа работы с сетевыми интерфейсами;
* Обзор функциональных возможностей класса NetworkInetrface.
  1. Выбор и обоснование способа работы с сетевыми интерфейсами

Сетевой интерфейс является точкой соединения между компьютером и частной или общедоступной сетью. Сетевой интерфейс обычно является сетевой интерфейсной картой (NIC), но необязательно должен иметь физическую форму. Вместо этого сетевой интерфейс может быть реализован с помощью программного обеспечения. Например, интерфейс loopback (127.0.0.1 для IPv4 и :: 1 для IPv6) является не физическим устройством, а частью программного обеспечения, имитирующей сетевой интерфейс. Интерфейс loopback обычно используется в тестовых средах.

[java.net.NetworkInterface](https://docs.oracle.com/javase/8/docs/api/java/net/NetworkInterface.html) работает с обоими типами интерфейсов.

NetworkInterface так же полезен и для многосетевых систем, которые имею несколько сетевых карт. Используя NetworkInterface, можно определить, какую сетевую карту необходимо использовать для различных процессов в сети.

* 1. Обзор функциональных возможностей класса NetworkInetrface

С помощью данного класса и его методов (некоторые из которых приведены в таблице 1) можно узнать большое количество информации о сетевых интерфейсах. Для примера возьмем список IP адресов интерфейса. Эту информацию можно получить из экземпляра NetworkInterface, используя один из двух методов. Первый метод, getInetAddresses(), возвращает перечисление InetAddress. Другой метод, getInterfaceAddresses(), возвращает список экземпляров java.net.InterfaceAddress. Этот метод используется, когда вам нужна дополнительная информация об адресе интерфейса за пределами его IP-адреса. Например, может потребоваться дополнительная информация о маске подсети и широковещательном адресе, когда адрес является адресом IPv4, и длину префикса сети в случае адреса IPv6.

Таблица 1 – Некоторые методы класса NetworkInterface

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| [getDisplayName](https://docs.oracle.com/javase/7/docs/api/java/net/NetworkInterface.html#getDisplayName())() | Возвращает отображаемое имя интерфейса |
| [getHardwareAddress](https://docs.oracle.com/javase/7/docs/api/java/net/NetworkInterface.html#getHardwareAddress())() | Возвращает МАС адрес интерфейса |
| [getNetworkInterfaces](https://docs.oracle.com/javase/7/docs/api/java/net/NetworkInterface.html#getNetworkInterfaces())() | Возвращает все интерфейсы на устройстве |
| [isLoopback](https://docs.oracle.com/javase/7/docs/api/java/net/NetworkInterface.html#isLoopback())() | Проверяет, петлевой ли этот интерфейс |
| [getName](https://docs.oracle.com/javase/7/docs/api/java/net/NetworkInterface.html#getName())() | Возвращает имя интерфейса |
| isUp() | Проверяет, работающий ли это в данный момент интерфейс |

Сетевые интерфейсы могут быть иерархически организованы. Класс NetworkInterface включает два метода, getParent() и getSubInterfaces(), относящиеся к иерархии сетевого интерфейса. Метод getParent () вовзращает «родительский» NetworkInterface интерфейса. Если работа идет с субинтерфейсом, то getParent () возвращает ненулевое значение. Метод getSubInterfaces () возвращает все субинтерфейсы.

Класс NetworkInterfaces так же дает возможность разработчику программного обеспечения узнать сетевые параметры об интерфейсе. Например, можно выяснить, в каком из двух состояний (up/down) он находится. Методы isLoopback (), isPointToPoint (), isVirtual () позволяют узнать тип сетевого интерфейса:

1) петлевой;

2) физический;

3) виртуальный.

Рассмотрев возможности класса NetworkInterface , можно сделать вывод, что он полностью подходит нам для реализации необходимого сервиса.

1. Конструкторская часть
   1. Техническое задание

Изучить соответствующие системы, спроектировать интерфейс компонента, реализовать компонент, спроектировать JUnit тесты, провести тестирование, описать требования, конструкцию, особенности сборки проекта в документации.

* 1. Выбор технических решений, удовлетворяющих интересам ЗС

В таблице ниже представлены результаты выявления и начального анализа заинтересованных сторон (ЗС) и их интересов по отношению к системе.

Таблица 2 – Заинтересованные стороны и их интересы по отношению к системе

|  |  |
| --- | --- |
| **Интересы заинтересованных сторон** | **Технические решения** |
| Отслеживание событий изменения состояния параметров сетевых интерфейсов | При любом изменении параметров, а также состояния сетевого интерфейса, данное событие обрабатывается одним из классов разработанной системы.  Корректность работы будет проверяться в процессе разработки с помощью юнит тестов. |
| Быстрая и полная передача исходного кода, настроек, документов.  Возможность в дальнейшем совершенствовать систему, например, добавляя новые модули и возможности | Код и настройки разрабатываемого менеджера будут находиться в системе GitHub. Контроль версий будет производиться с использованием системы Git.  Для модульного тестирования будет использована система JUnit. |
| Быстрое внесение изменений и проверка, что они не нарушают работу существующего функционала. | Исходный код будет структурирован по пакетам. Повторно-используемые методы будут вынесены в родительские классы. |

* 1. Структура проекта

Разберем работу написанной системы, для этого опишем состав каждого плагина, из которых состоит наша система:

Таблица 3 – Разработанные плагины и их описание

|  |  |
| --- | --- |
| **Плагин** | **Содержимое** |
| com.bmstu.coursework.ethernet.manager | содержит интерфейсы для сервисов и клиентов разрабатываемого ПО |
| com.bmstu.coursework.ethernet.manager.clients | содержит классы, являющиеся клиентами разрабатываемого ПО |
| com.bmstu.coursework.ethernet.manager.service | содержит класс, являющийся сервисом разрабатываемого ПО |

* + 1. Плагин com.bmstu.coursework.ethernet.manager

Данный плагин содержит 2 пакета:

* com.bmstu.coursework.ethernet.manager;
* com.bmstu. coursework.ethernet.manager.model.

Пакет com.bmstu.coursework.ethernet.manager содержит интерфейс IEthernetListener, который включает в себя 1 метод:

Таблица 4 – Методы интерфейса IEthernetListener и их описание

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| void adapterChanged(EthernetAdapterEvent event); | Добавляет событие, содержащее информацию о добавлении, изменении и удалении информации о сетевых интерфейсах. |

Пакет com.bmstu.coursework.ethernet.manager содержит интерфейс IEthernetManager,

который включает в себя 1 метод:

Таблица 5 – Методы интерфейса IEthernetManager и их описание

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| Collection<EthernetAdapterInfo> getCurrentAdaptersInfo(); | Возвращает действительную информацию о состоянии сетевых интерфейсов. |

Пакет com.bmstu.coursework.ethernet.manager.model содержит 3 класса:

* EthernetAdapterEvent.java
* EthernetAdapterInfo.java
* OldNewInfoPair.java

Таблица 6 – Классы пакета com.bmstu.coursework.ethernet.manager.model и их описание

|  |  |
| --- | --- |
| **Класс** | **Состав и описание** |
| OldNewInfoPair | public OldNewInfoPair – конструктор  public EthernetAdapterInfo getOldInfo() – возвращает старую информацию об интерфейсах;  public EthernetAdapterInfo getNewInfo() – возвращет обновленную информацию;  private EthernetAdapterInfo oldInfo; - имя обьекта класса EthernetAdapterInfo;  private EthernetAdapterInfo newInfo; – имя обьекта класса EthernetAdapterInfo; |
| EthernetAdapterEvent | public EthernetAdapterEvent() – конструктор;  public Collection<EthernetAdapterInfo> getAdded() – возвращает добавленную информацию об интерфейсах;  public Collection<EthernetAdapterInfo> getRemoved() – возвращает удаленную информацию об интерфейсах;  public Collection<OldNewInfoPair> getChanged() – возвращает замененную информацию об интерфейсах. |
| EthernetAdapterInfo | public EthernetAdapterInfo(String displayName, String name, Collection<String> addresses) – конструктор;  public String getDisplayName() – возвращает display имя интерфейса;  public String getName() – возвращает название интерфейса;  public Collection<String> getAddresses() – возвращает ip адреса сетевых интерфейсов;  public void setDisplayName(String displayName) – назначает displayname сетевому интерфейсу;  public void setName(String name) - назначает название сетевому интерфейсу;  public void setAddresses(Collection<String> addresses) – назначает ip адрес сетевому интерфейсу;  public String toString() – возвращает строку с displayname, название, ip адресом;  public int hashCode() – возвращает hash код объекта;  public boolean equals(Object object) – проверяет объекты на равнозначность. |

* + 1. Плагин com.bmstu.coursework.ethernet.manager.clients

Данный плагин содержит 1 пакет:

* com.bmstu.coursework.ethernet.manager.clients

Пакет com.bmstu.coursework.ethernet.manager.clients содержит 3 класса:

* AdapterAddListener;
* ChangeAdapterListener;
* RemoveAdapterListener.

Таблица 7 – Классы пакета com.bmstu.coursework.ethernet.manager.clients и их описание

|  |  |
| --- | --- |
| **Класс** | **Состав и описание** |
| AdapterAddListener | public void adapterChanged(EthernetAdapterEvent event) – выводит информацию о добавлении сетевого интерфейса, если такая есть. |
| ChangeAdapterListener | public void adapterChanged(EthernetAdapterEvent event) – выводит информацию об изменении сетевого интерфейса, если такая есть. |
| RemoveAdapterListener | public void adapterChanged(EthernetAdapterEvent event) – выводит информацию об удалении сетевого интерфейса, если такая есть. |

* + 1. Плагин com.bmstu.coursework.ethernet.manager.service

Данный плагин содержит 1 пакет:

* com.bmstu.coursework.ethernet.manager.impl.

Пакет com.bmstu.coursework.ethernet.manager.clients содержит 2 класса:

* EthernetManager;
* AdapterListener.

Таблица 7 – Классы пакета com.bmstu.coursework.ethernet.manager.impl и их описание

|  |  |
| --- | --- |
| Класс | Состав и описание |
| EthernetManager | public EthernetManager() – конструктор;  public Collection<EthernetAdapterInfo> getCurrentAdaptersInfo() – возвращает действительную информацию о сетевых интерфейсах;  public void activate(Map<String, Object> properties) – запускает поток, активирует сервис;  public void deactivate() – останавливает работу потока, деактивирует сервис;  public void bindListener(IEthernetListener listener) – добавляет прослушивающего клиента;  public void unbindListener(IEthernetListener listener)- удаляет прослушивающего клиента;  private void throwEvent(EthernetAdapterEvent event) – сообщает клиентам о событии;  private Collection<EthernetAdapterInfo> createInfos() – создает информацию о сетевых интерфейсах;  private Collection<String> getAddresses(NetworkInterface networkInterface) – возвращает набор ip адресов; |
| AdapterListener | public void run() – выдает информацию сетвых интерфейсах, пока сервис находится в активном режиме работы;  private EthernetAdapterInfo getInfoByName(String name) – ищет информацию об интерфейсе по его названию |

* 1. Диаграмма компонентов

На рисунке 1 представлена диаграмма компонентов разработанного программного обеспечения.

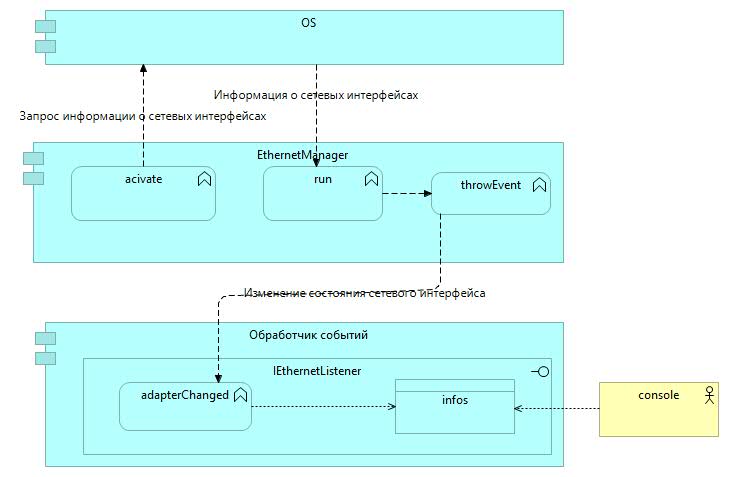


Рисунок 1 – Диаграмма компонентов

* 1. Диаграмма классов

Для наглядности, с помошью редактора UML Papirus была создана диаграмма классов. Весь проект состоит из четырех пакетов. Функционал всех пакетов и их компонентов был описан выше.

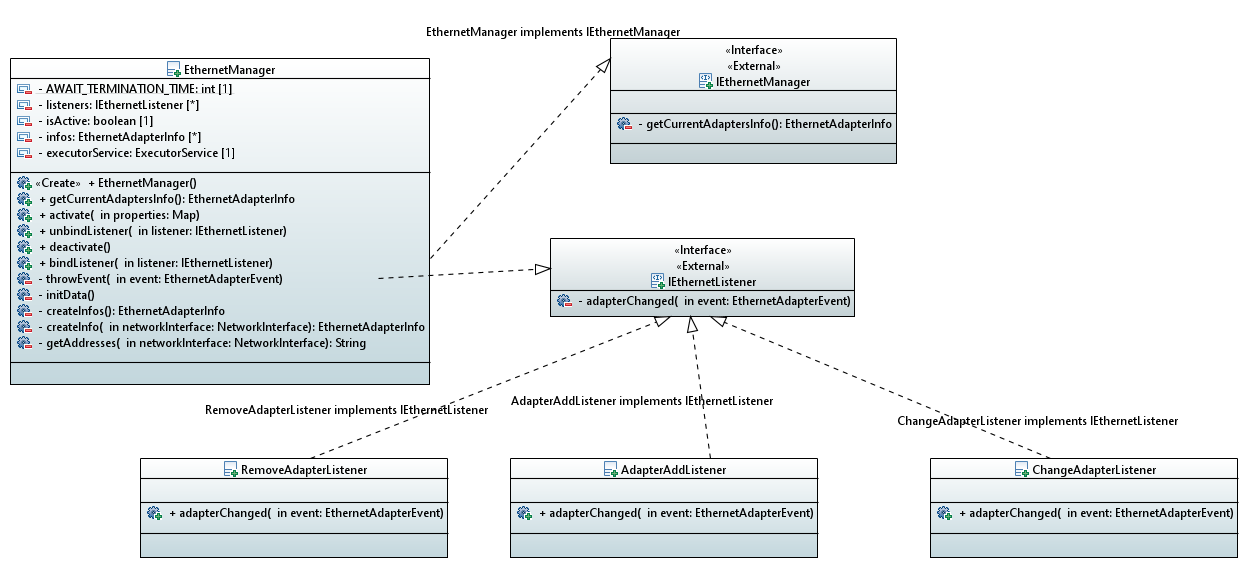


Рисунок 2 – Диаграмма классов

* 1. Алгоритм работы ПО

Окончательный алгоритм работы разработанного ПО следующий:  
1) Активация сервиса;  
2) Активация клиентов, их занесение менеджером в коллекцию клиентов;  
3) Запуск менеджером потока, в котором он через некоторые промежутки времени опрашивает порты;   
4) Полученная после запроса информация об интерфейсе сравнивается с последней сохраненной информацией об этом же интерфейсе по различным параметрам (состояние, ip адрес);  
5) Если два набора информации не идентичны, то создается событие, которое отправляется на обработку клиентам;  
6) В зависимости от типа поступившего события, выбирается нужный клиент.  
7) Сообщение обрабатывается и выводится в консоль  
8) Цикл повторяется

1. Технологическая часть
   1. Запуск разработанного приложения

Исходный код проекта доступен в репозитории GitHub [3].

Сервис EthernetManager взаимодействует с тремя клиентами AdapterAddListener, ChangeAdapterListener и RemoveAdapterListener. Запуск приложения осуществляется с помощью конфигурационного файла, который содержит настройки, необходимые для успешного запуска приложения.

* 1. Анализ искходного кода с помощью метрик качества

На рисунке 3 показано соотношение пакетов проекта по их размеру. Видно, что самый большой размер имеет пакет model.

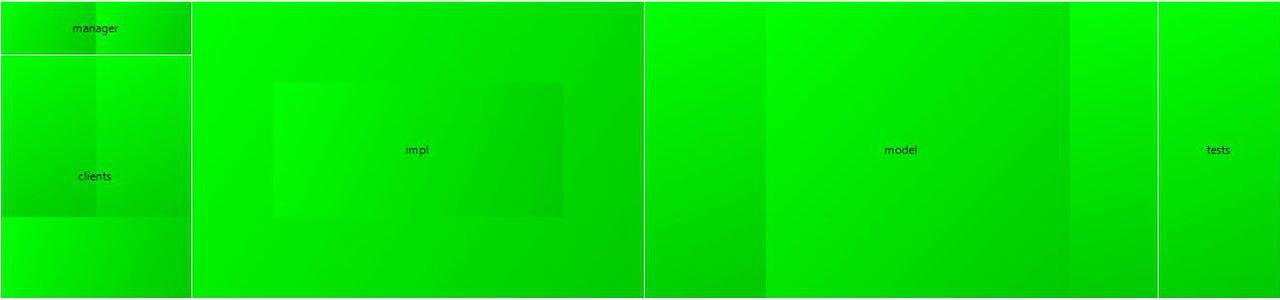


Рисунок 3 – Соотношение пакетов по размеру

Далее на рисунке 4 отображен список всех метрик по разделам. Всего имеется четыре раздела:

* метрики количества (Count);
* метрики сложности (Complexity);
* метрики Роберта Мартина (Robert C. Martin);
* метрики Чидамбера-Кемерера (Chidamber & Kermerer).
* Первый раздел с метриками количества (Count) содержит следующие метрики:
* количество классов верхнего уровня (Unit);
* среднее число внутренних классов на класс (Classes / Class);
* среднее число методов в классе (Methods / Class);
* среднее число полей в классе (Fields / Class);
* число строчек кода (ELOC);
* число строчек кода на модуль (ELOC / Unit).

Второй раздел с метриками сложности (Complexity) содержит всего три различных метрики:

* средняя циклическая сложность (CC);
* метрика Fat (Fat);
* средняя зависимость компонентов между модулями (ACD - Unit).

Третий раздел с метриками Роберта Мартина содержит следующие метрики:

* нормализованное расстояние от основной последовательности (D);
* абстрактность (A);
* нестабильность (I);
* число афферентных соединений (Ca);
* число эфферентных соединений (Ce).

Четвертый раздел с метриками Чидамбера-Кемерера содержит следующие метрики:

* средняя длина метода на класс (WMC);
* средняя глубина наследования (DIT);
* среднее количество классов-наследников (NOC);
* среднее число соединений класса (CBO);
* среднее число методов, которые потенциально могут быть выполнены в ответ на сообщение, полученное объектом этого класса (RFC);
* отсутствие единства методов (LCOM).

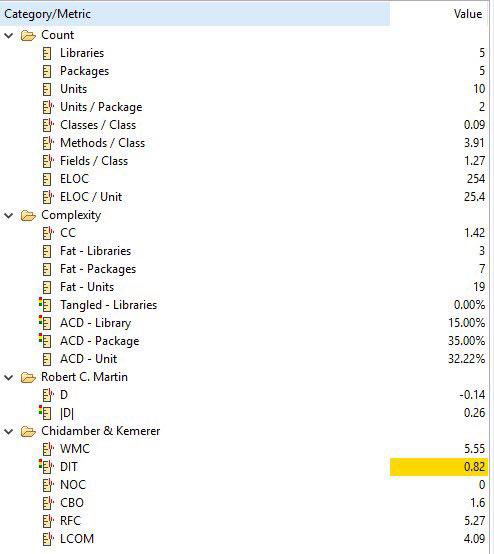


Рисунок 4 – Значения метрик

* 1. Анализ зависимостей в коде системы

Рисунок 5 описывает зависимости между пакетами разработанного программного обеспечения:

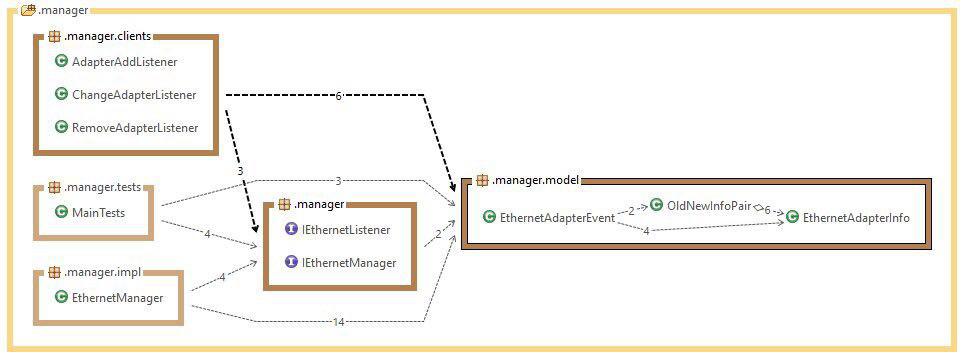


Рисунок 5 – Зависимости между пакетами

* 1. Тестирование на корректность работы

Работоспособность разработанного программного обеспечения проверялась на проводных и беспроводных подключениях к сети интернет.

На рисунке 6 показан вывод ПО при отключении и подключении сетевого кабеля к разъему eth1:

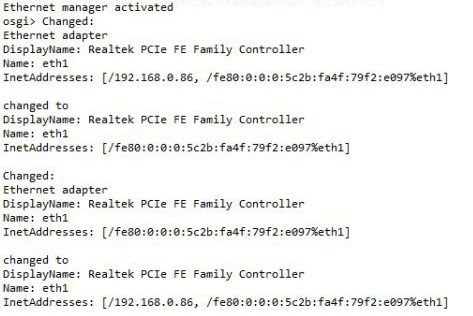
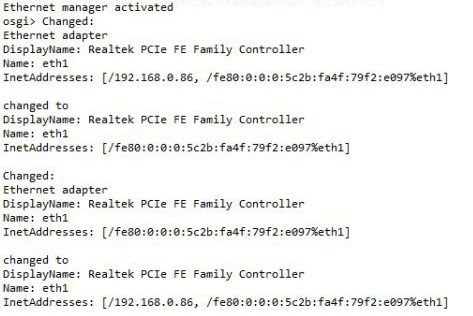


Рисунок 6– Работоспособность программы при физическом подключении

На рисунке 7 показан вывод ПО при отключении и подключении ПК к беспроводной сети:

  
Рисунок 7 – Работоспособность программы при подключении к беспроводной сети

* 1. Реализация Junit тестов

Для последующего тестирования разработанного программного обеспечения была поставлена задача написания JUnit тестов.

Интеграция JUnit с OSGI происходит с помощью ServiceTracker. ServiceTracker позволяет получить сервисы из OSGI. Для его использования тест необходимо запускать, как plugin test, иначе система не будет подгружать необходимые плагины.

В ходе выполнения курсовой работы было написано 3 теста, проверяющих следующий функционал ПО:

Доступность сервиса, используется метод assertNotNull, в случае ошибки в поиске сервиса выводится сообщение “Ethernet manager not found”

Возможность получения информации о сетевых интерфейсах, используется метод assertNotNull, в случае отсутствия какой-либо получаемой информации выводится сообщение “Infos is null”

Отсутствие ошибок при использовании клиентов.

Перед тестирующими методами используется аннотация Test.

Результаты тестирования JUnit приведены на рисунке 8:

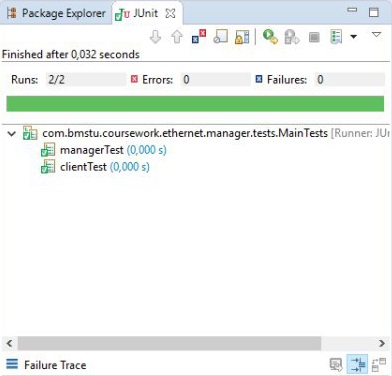


Рисунок 8 – Результат работы тестов для проверки на корректность работы

* 1. Оценка покрытия кода

Покрытие кода - мера, используемая при [тестировании программного обеспечения](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F). Она показывает процент [исходного кода](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) программы, который был выполнен в процессе тестирования.

На рисунке 9 приведена статистика покрытия инструкций в разработанном ПО в результате проведения вышеупомянутых тестов.

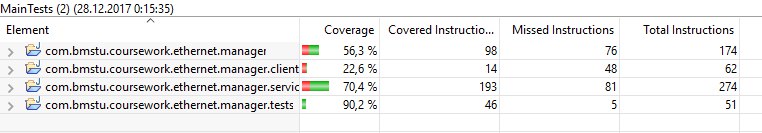


Рисунок 9 – Оценка покрытия кода

На рисунке 10 так же приведен пример статистического анализа покрытого кода, путем его выделения различными цветами:  
1) красный – код не был пройден при тестировании;

2) желтый – условные и прочие операторы;

3) зеленый – код был пройден.

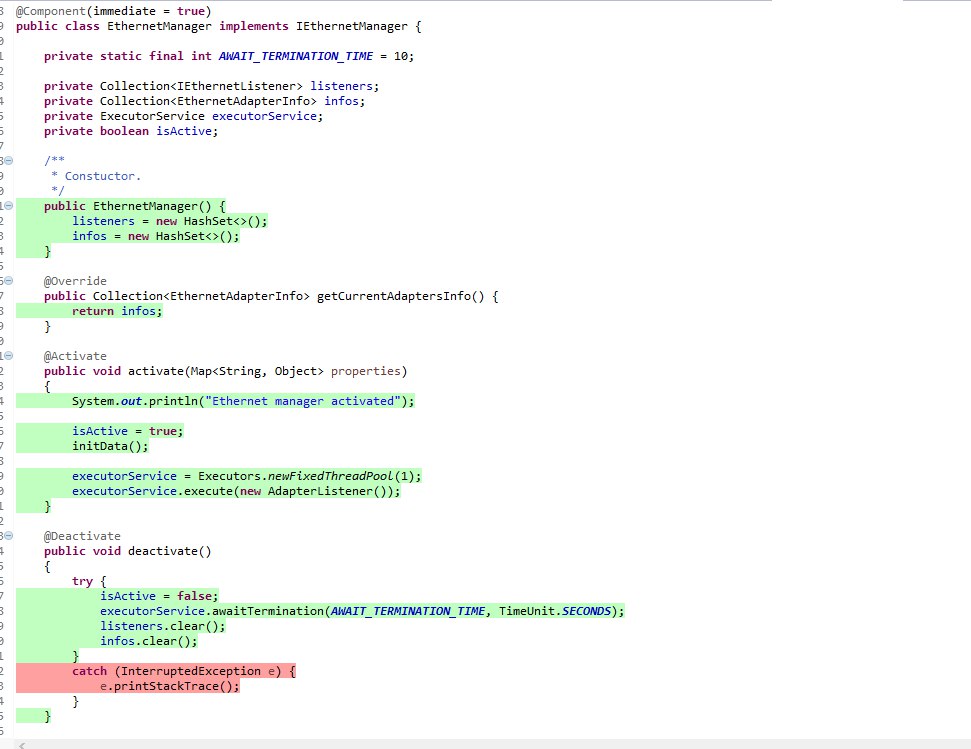


Рисунок 10 – Графическое отображение пройденных участков кода

По рисункам видно, что разработанное программное обеспечение функционирует в требуемом режиме.

Заключение

В ходе выполнения курсовой работы был разработан OSGI сервис мониторинга сетевых Ethernet интерфейсов. Работа была выполнена с помощью технологий OSGI и JUnit в среде разработки Eclipse.

В результате выполнения проекта были решены задачи, поставленные в начале работы. Были изучены основные принципы работы необходимых технологий; разработано ПО, соответствующее требованиям ТЗ; проведены необходимые тесты работоспособности программного продукта. Результаты тестирования свидетельствуют правильное функционирование ПО.

Список литературы

[1]  Lesson: Programmatic Access to Network Parameters : [Электронный ресурс]. Режим доступа: https://docs.oracle.com/javase/tutorial/networking/nifs/index.html

[2] OSGI modulation – Tutorial : [Электронный ресурс]. Режим доступа: http://www.vogella.com/tutorials/OSGi/article.html

[3] OSGI in Action: Creating modular applications in JAVA : Richard Hall 2011. – 416 с.

[4] OSGI and Equinox : J. McAffer, 2010. – 328 с.

[5] OSGI services – Tutorial : [Электронный ресурс]. Режим доступа: http://www.vogella.com/tutorials/OSGiServices/article.html