**Левин В.Е., Шульга Т.Э.**

**Создание и настройка среды для работы с онтологиями используя фреймворк «Jena»**

**Учебное пособие**

# Введение

После подготовительной работы, когда Вы спроектировали схему онтологии и создали её, возникает вопрос – «Что же дальше?». Вопрос правильный, ведь для последующей работы с онтологией её надо где-то хранить, куда-то надо слать запросы на выборку, на добавление. А онтология – вещь очень гибкая и, если вы планируете расширение в будущем, могут потребоваться программы-черви, которые будут заполнять онтологию и связывать её с другими.

Средства для подобного уже существуют, и Вы способны самостоятельно создать окружение для Вашей онтологии используя язык Java и библиотеку Jena.

Созданию такого окружения, разделённого на сервер, хранящий онтологию, а также веб-интерфейс для выборки данных, и посвящено это учебное пособие.

# Необходимые инструменты

Для создания среды потребуется следующие инструменты:

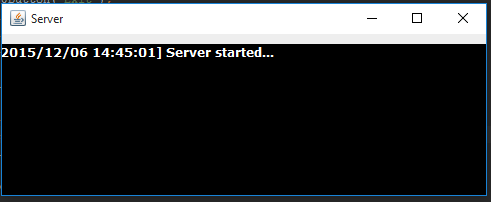
1. **Среда разработки Java приложений:** Eclipse, Intellij IDEA или NetBeans (в данном пособии используется Intellij IDEA)
2. **Runtime среда:** Java(<http://www.java.com/ru/>) и Java Development Kit (<http://www.oracle.com/technetwork/java/javase/downloads/index.html>)
3. **Фреймворк для работы с онтологиями:** для языка Java имеется Jena (<https://jena.apache.org/documentation/ontology/>)
4. **Вспомогательные библиотеки для облегчения и ускорения работы:**

* Simple-JSON (<https://github.com/fangyidong/json-simple>) – Легковесная и мощная библиотека для форматирования данных в формат JSON
* opa-core (<https://github.com/opaoz/opa-core>) – Библиотека с функциями для упрощения разворачивания клиент-серверного приложения и работы с Jena

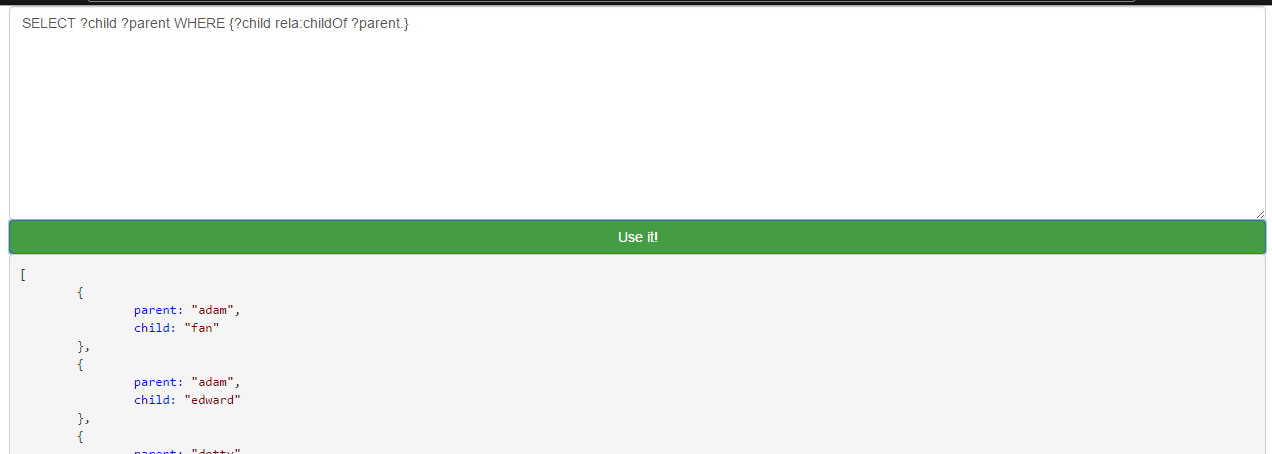
# Схема разрабатываемого приложения

Среда будет состоять из двух частей:

**Сервер**. Часть, написанная на языке Java, которая хранит в себе онтологию, обрабатывает запросы и производит выборку и добавление данных.



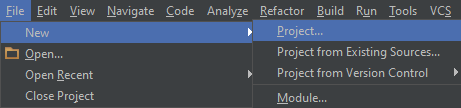
**Клиент**. Веб-интерфейс для приложения, позволяющий формировать и отсылать запросы к серверу, а также принимать результаты и выводить их в качестве наглядных данных.



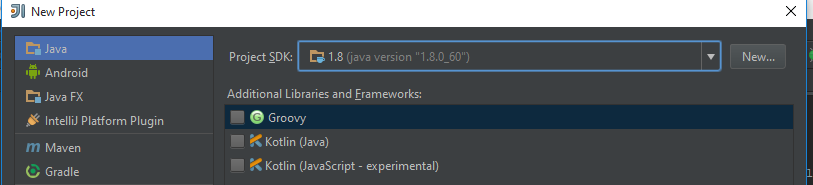
# Разработка

## Создание проекта и подключение требуемых библиотек

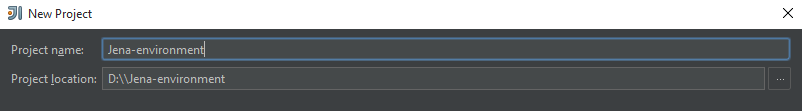
Командой меню «File > New > Project…» открываем диалоговое окно с выбором типа создаваемого проекта



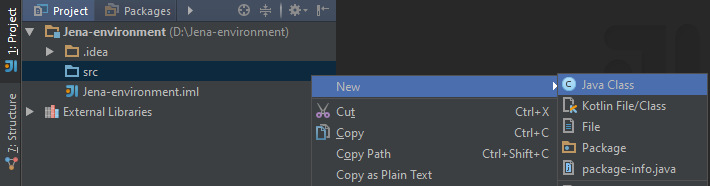
Выбираем тип проекта - «Java» и версию SDK из выпадающего списка и жмём кнопку «Next»

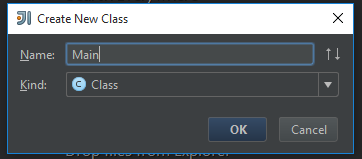


Пропускаем следующий экран с выбором темплейта, затем вводим название и расположение проекта и жмём «Finish»



Вот мы и создали проект, теперь добавим главный файл, настроим параметры запуска и выведем в консоль заветное «Hello, World!».

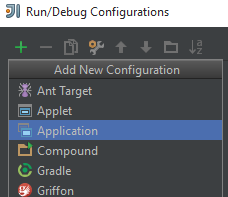




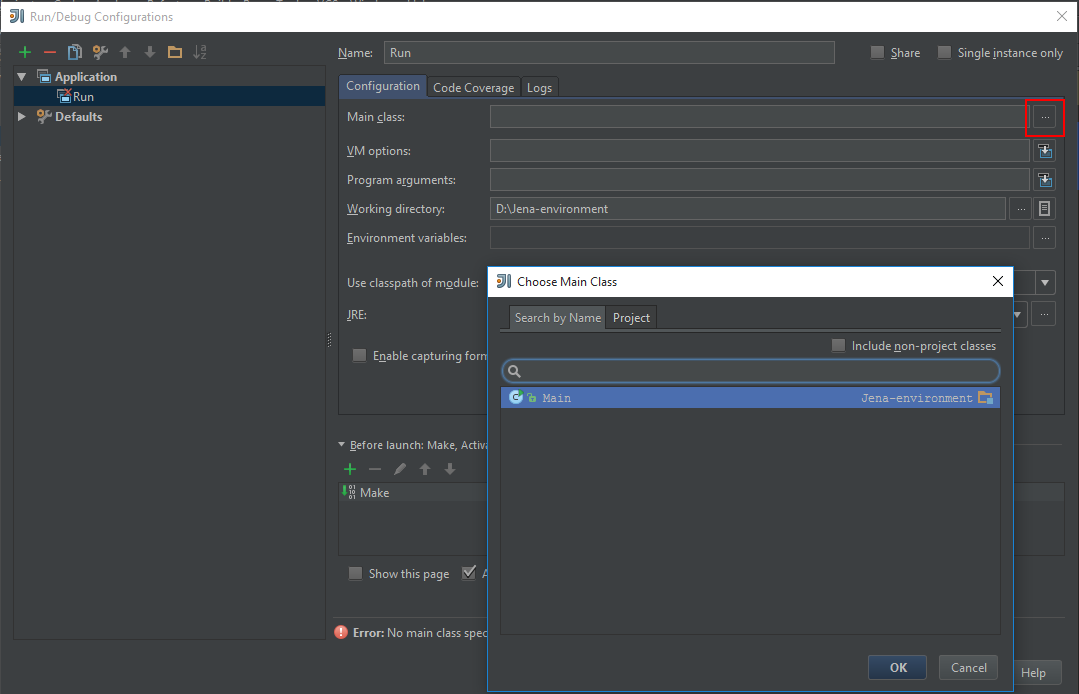
Содержимое класса:

public class Main {  
 public static void main(String[] args) {  
 System.*out*.println("Hello, world!");  
 }  
}

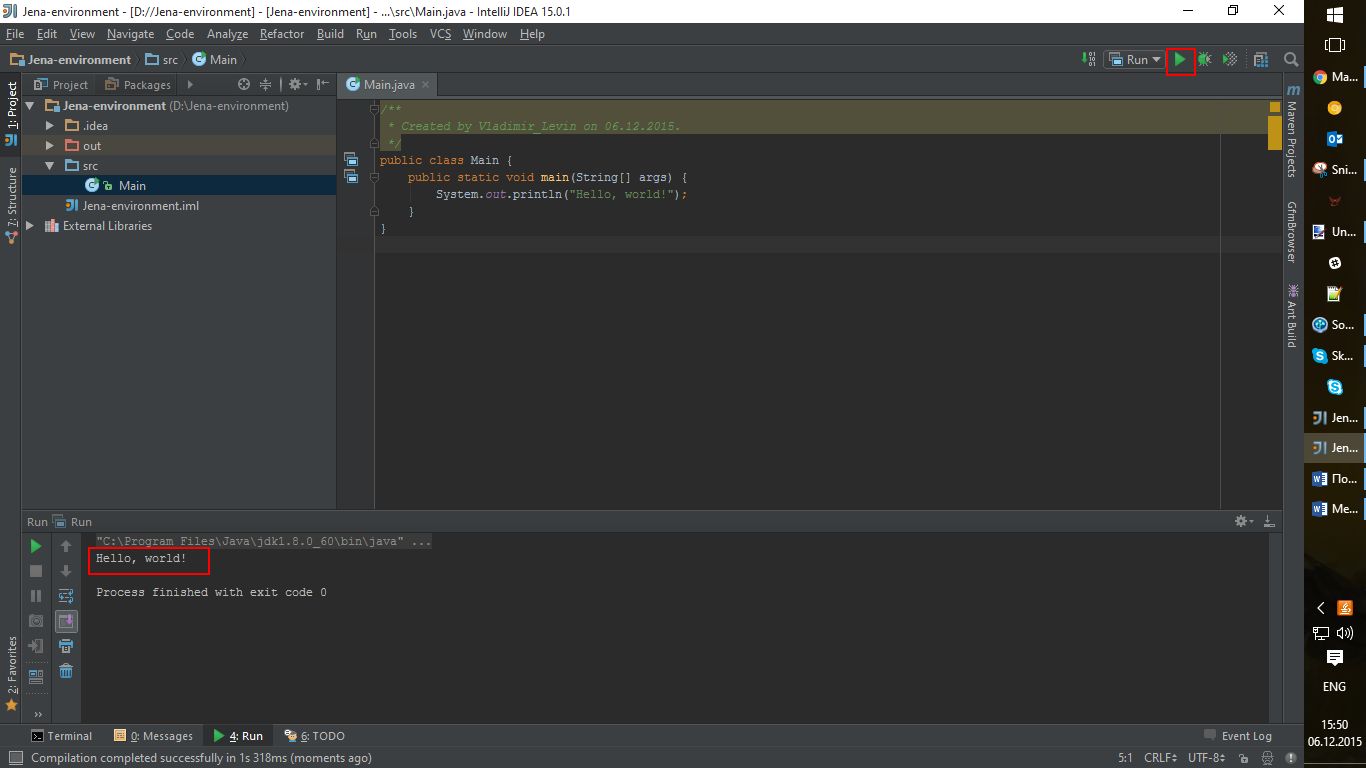
Далее настроим конфигурацию запуска. Для этого жмём выпадающее меню справа сверху и выбираем пункт «Edit Configurations…» , затем зелёный «+» и выбираем «Application»



Нажимаем «…» возле поля «Main class» и выбираем ново созданный класс Main



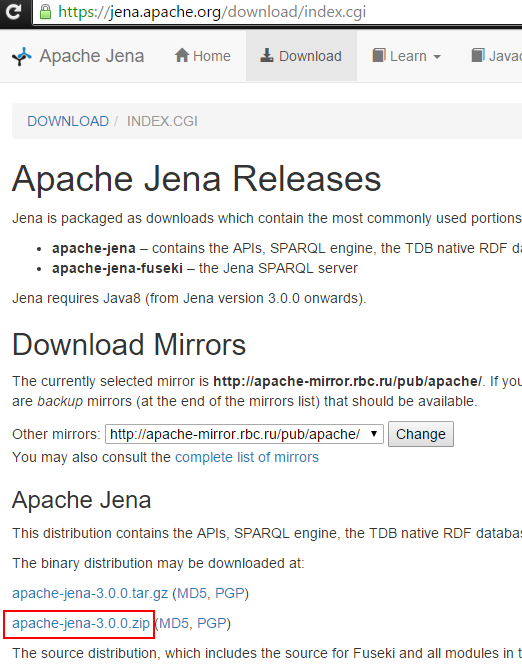
Теперь, по нажатию кнопки «Run» наш проект запустится



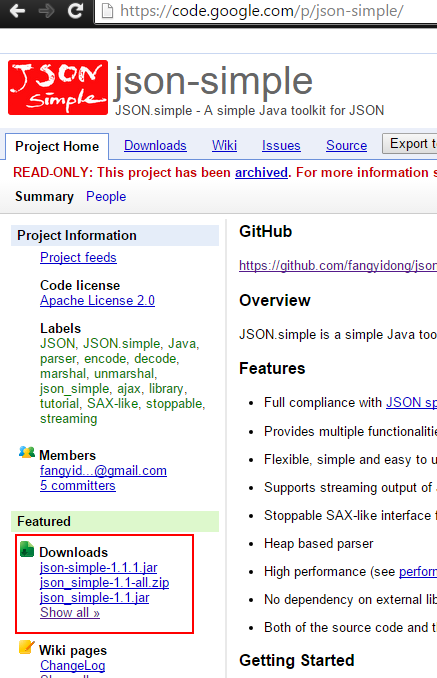
## Подключение библиотек

Скачиваем все библиотеки:

1. Jena - <https://jena.apache.org/download/index.cgi>

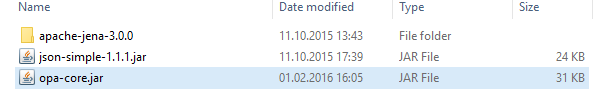


1. Simple-JSON - <https://code.google.com/p/json-simple/>

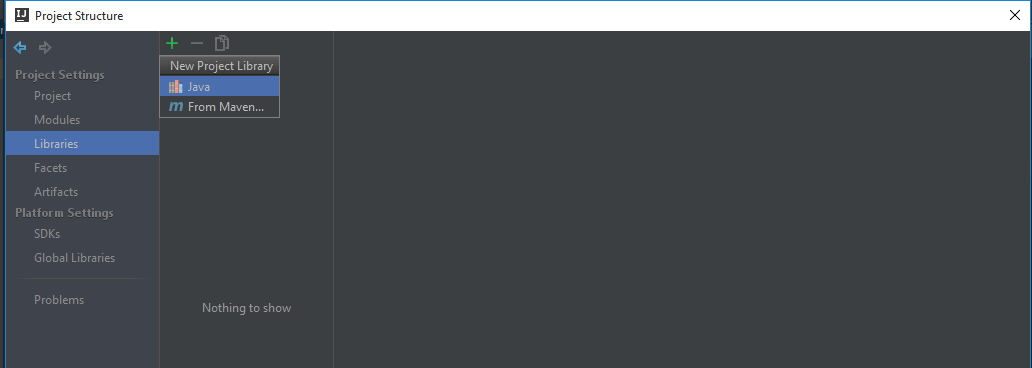


1. opa-core - <https://github.com/opaoz/opa-core/blob/master/dist/opa-core.jar?raw=true>

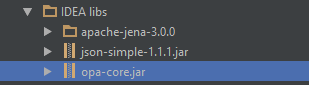
Складываем их в одну папку



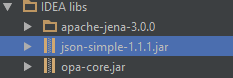
Открываем «File -> Project Structure -> Libraries»



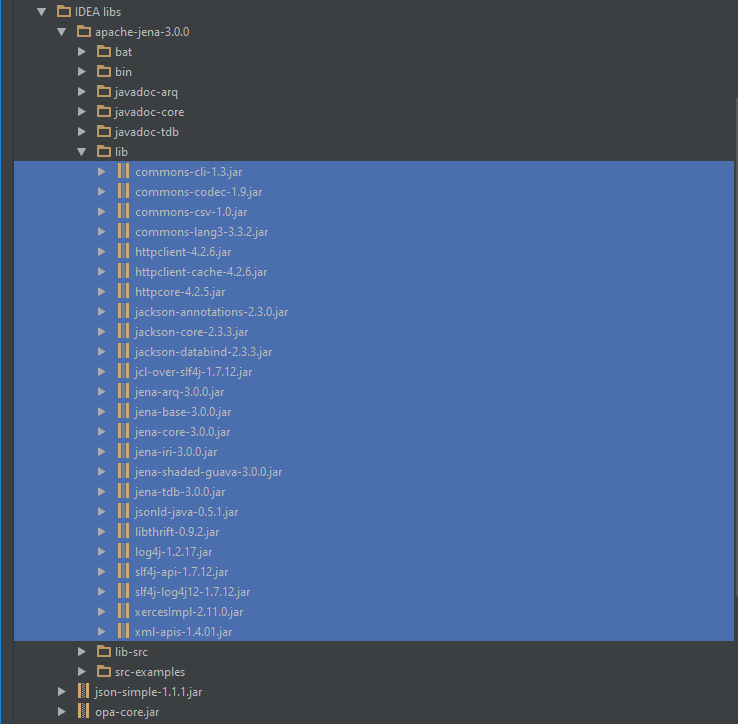
42Нажимаем на «+» и выбираем «Java», затем выбираем первый файл



Повторяем для второго



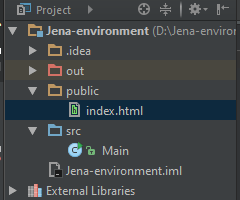
А для apache-jena выбираем всё в папке “lib”



## Структура проекта

Так как разрабатываемый проект является «клиент-серверным», возведём соответствующую структуру папок.

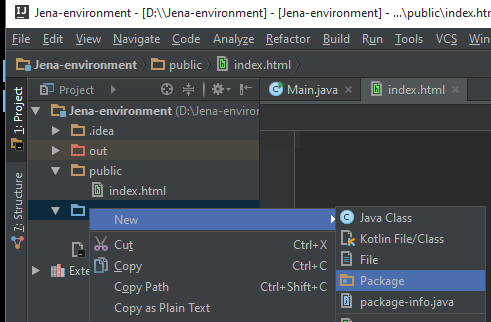
Сначала создаём папку “public” в корне проекта. Здесь будет находится вся клиентская часть проекта. Чтобы обозначить это, создадим файл “index.html”.



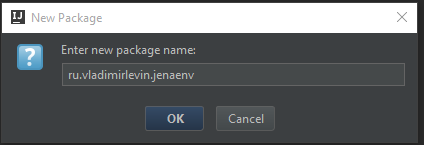
Вся серверная часть будет располагаться в папке “src”. Внутри будет возводится структура из пэкэджей (*[Package (Java)](https://ru.wikipedia.org/wiki/Package_(Java)))* в соответствии с популярным [стайлгайдом](https://docs.oracle.com/javase/tutorial/java/package/namingpkgs.html). Суть его состоит в том, что имя корневого пакета составляется из 3 частей:

1. **Домен** (org, ru, com, ua)
2. **Автор**
3. **Название проекта**

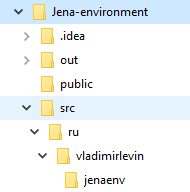
В итоге получаем *«ru.vladimirlevin.jenaenv»*



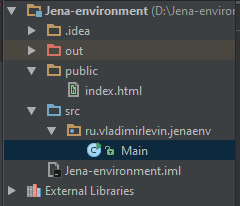
При создании можно писать наименование сразу с «.»



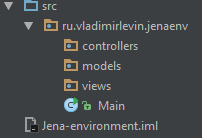
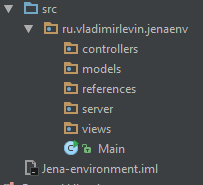
В итоге создастся соответствующая структура папок:



В созданный package и перенесём наш класс “Main”



Продолжим структурирование проекта, добавим MVC составляющую и пару вспомогательных пакетов

Суть такого подхода состоит в следующем:

1. **Controllers** – папка, содержащая логику конкретных запросов к серверу.
2. **Models** – хранилище RDF-моделей
3. **References** – папка с константами, используемыми в проекте
4. **Server** – двигатель приложения, обработка всех входящих запросов и подключений
5. **Views** – UI-компоненты, для визуализации действий сервера.
6. **Main**.java – стартер проекта

## Создание окружения

Начнём с файла конфигурации. Создадим класс **References.java** в пэкэдже «references», и добавим в него нужные константы.

Содержимое файла:

package ru.vladimirlevin.jenaenv.references;  
  
import java.awt.\*;  
import java.text.DateFormat;  
import java.text.SimpleDateFormat;  
  
public final class References {  
 /\*Model\*/  
 public static final String *FAMILY\_PATH* = "D:/Family/family.rdf"; // Путь сохранения rdf-модели  
  
 /\*Server\*/  
 public static final int *DEFAULT\_SERVER\_PORT* = 8080; // Порт по умолчанию для сервера  
 public static final int *SERVER\_PORT* = 8080; // Порт для сервера (если не указал используются дефолтный)  
  
 /\*Log\*/  
 public static final Font *LOG\_FONT* = new Font("Verdana", Font.*BOLD*, 12); // Стиль текста в консоли сервера  
 public static final DateFormat *DATE\_FORMAT* = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss"); // Формат даты в консоли  
}

Традиционно сервер запускается из-под какой-либо оболочки, так что проблемы с логами не возникает. Мы же создадим свой собственный интерфейс, для логов сервера.

Для начала добавим компонент **Log.java** (ru.vladimirlevin.jenaenv.views.components) и унаследуем его от класса *JPanel*. Log - это часть интерфейса, наглядно демонстрирующая что сейчас происходит с сервером и стилизованная под консоль.

package ru.vladimirlevin.jenaenv.views.components;  
  
import ru.vladimirlevin.jenaenv.references.References;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.text.DateFormat;  
import java.util.Date;  
  
*/\*\*  
 \* Created by Vladimir\_Levin on 14.02.2016.  
 \*/*public class Log extends JPanel {  
 private static final long *serialVersionUID* = -7968405633747657642L; // Автоматически генерируемый UID  
 private JTextArea text; // Тестовое представление логов  
 private DateFormat dateFormat = References.*DATE\_FORMAT*; // Паттерн форматирования даты из файла References  
  
 // Конструктор класса  
 public Log() {  
 text = new JTextArea(); // Добавляем текст на панель  
 text.setAlignmentX(*LEFT\_ALIGNMENT*); // Позиционируем её налево  
 text.setAlignmentY(*TOP\_ALIGNMENT*); // и прижимаем к верху  
 text.setBackground(Color.*BLACK*); // Задний фон - чёрный  
 text.setForeground(Color.*WHITE*); // Цвет текста - белый  
 text.setFont(References.*LOG\_FONT*); // Стиль шрифта из файла References  
 text.setLineWrap(true); // Устанавливаем перенос строк  
 text.setWrapStyleWord(true); // И слов  
  
 JScrollPane scrollPane = new JScrollPane(text); // Создаём ScrollBar  
 scrollPane.setPreferredSize(new Dimension(500, 200)); // Ограничиваем размеры области вывода  
 this.add(scrollPane); // Добавляем к нашей импровизированной консоли  
 this.setVisible(true); // Отрисовываем  
 }  
  
 // Метод добавления сообщений в консоль  
 public void info(String message) {  
 text.append("[" + dateFormat.format(new Date()) + "] " + message + " \n");  
 }  
  
 // Размеры окна консоли  
 @Override  
 public Dimension getPreferredSize() {  
 return new Dimension(500, 200);  
 }  
}

После добавления компонента, нам требуется класс, который будет непосредственно показывать окно. Создадим для этого класс **ServerUI.java** (ru.vladimirlevin.jenaenv.views) и унаследуем его от класса *JFrame*

package ru.vladimirlevin.jenaenv.views;  
  
import ru.vladimirlevin.jenaenv.views.components.Log;  
  
import javax.swing.\*;  
import java.awt.\*;  
  
*/\*\*  
 \* Created by Vladimir\_Levin on 14.02.2016.  
 \*/*public class ServerUI extends JFrame {  
 private static final long *serialVersionUID* = 6182239340196405650L;  
 private Log log; // Окно лога, созданное в предыдущем шаге  
  
 public ServerUI() {  
 log = new Log(); //Инициализируем окно  
  
 this.setLayout(new FlowLayout()); // Т.к. у нас всего один компонент в окне, установим данный тип заполнения  
 this.add(log); // Добавляем окно лога на фрейм  
 this.setTitle("Server"); // Устанавливаем название  
 this.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*); // При закрытии окна, выключаем сервер  
 this.pack(); // Устанавливаем размер по контенту  
 this.setVisible(true); // Отрисовываем  
 }  
  
 // Метод добавления сообщения в лог  
 public void println(String message) {  
 log.info(message);  
 }  
}

Т.к. сервер будет принимать Sparql запросы и обрабатывать их, создадим контроллер **RequestManager.java** (ru.vladimirlevin.jenaenv.controllers). Оставим его пока пустым, до создания модели.

package ru.vladimirlevin.jenaenv.controllers;  
  
*/\*\*  
 \* Created by Vladimir\_Levin on 14.02.2016.  
 \*/*public class RequestManager {  
 public RequestManager(String request) {  
 // Обаратываем запрос  
 }  
}

Обработку подключений на сервере следует осуществлять в отдельном потоке, создадим для этого класс **ClientThread.java** (ru.vladimirlevin.jenaenv.server), и реализуем в нём интерфейс *Runnable*