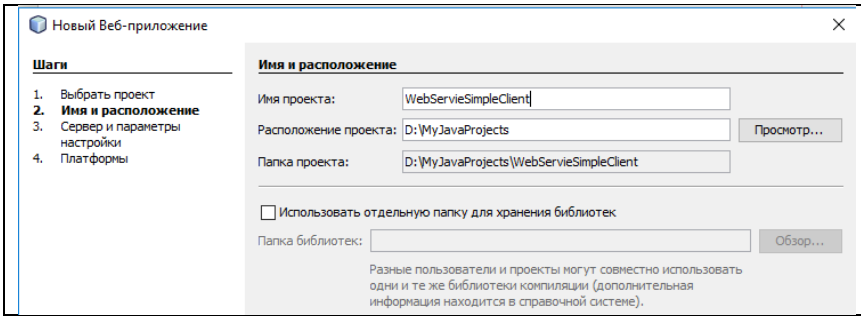
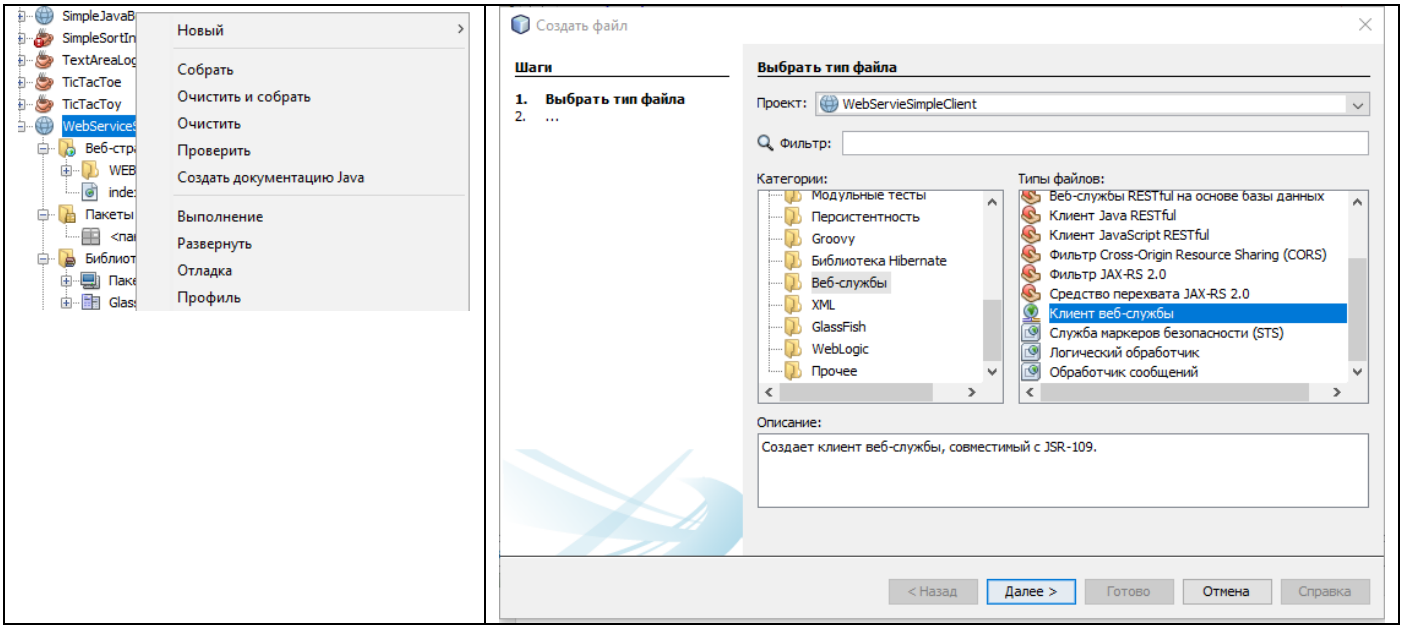
Прейдем к созданию клиента нашего сервера.



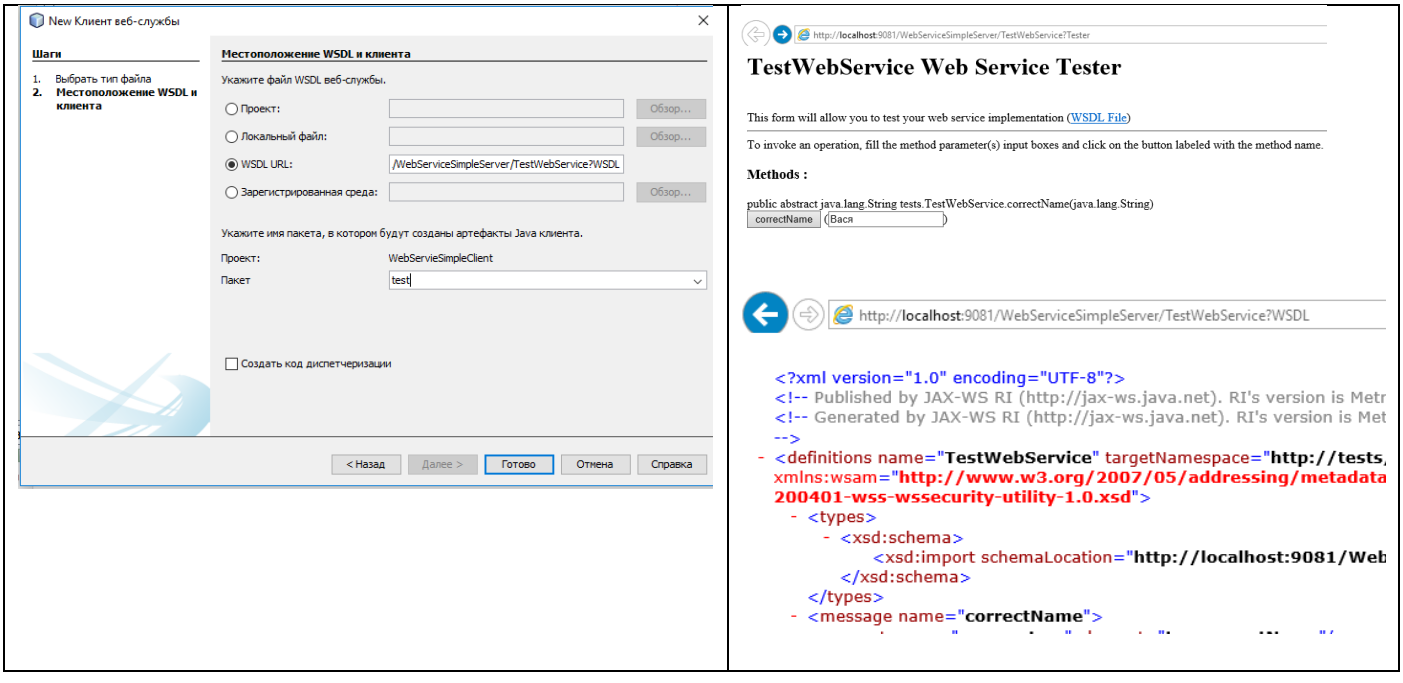
Создаем обычное консольное Java application (File->New) – так же как создавали ранее.

И уже в это проект TestWebServiceClient добавляем клиент вэб-службы

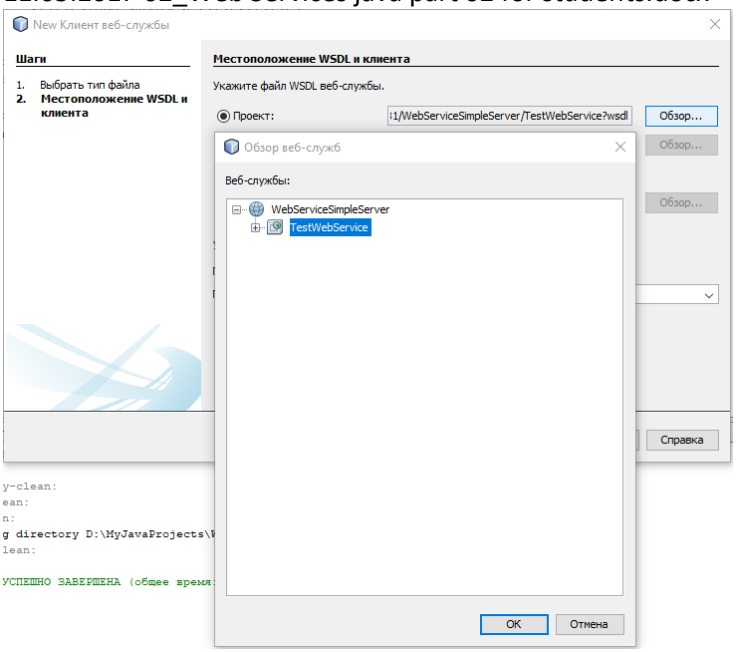


Нам необходимо создать клиента нашей вэб-службы на основании проекта или файла WSDL нашей службы. Для этого на следующем шаге необходимо выбрать проект или ввести адрес к нашему файлу WSDL. Адрес файла бы берем из окна тестирования нашей вэб-службы из проекта:

http://localhost:9081/WebServiceSimpleServer/TestWebService?WSDL



или



Структура консольного проекта	Итоговый вид функции main использующей наш web-srvice

TestWebService
- Пакеты исходных кодов
  - META-INF
    - jax-ws-catalog.xml
  - META-INF.wsdl.localhost_9081.WebServiceSimpleServer
    - TestWebService.wsdl
    - TestWebService.xsd_1.xsd
  - testwebservice
    - TestWebService.java
- Тесты
- Созданные исходные файлы (jax-ws)
  - test
    - CorrectName.java
    - CorrectNameResponse.java
    - ObjectFactory.java
    - TestWebService.java
    - TestWebService_Service.java
    - package-info.java
- Ссылки на веб-службы
  - TestWebService
    - TestWebService
      - TestWebServicePort
        - correctName
- Библиотеки
- Библиотеки тестов

```java
package testwebservice;
import test.TestWebService_Service;

public class TestWebService {

    public static void main(String[] args) {
        TestWebService_Service service = new TestWebService_Service();
        test.TestWebService port = service.getTestWebServicePort();
        String currentName = port.correctName("Вася");
        System.out.println(currentName);
    }

}
```

Дополнительные сведения:

https://ru.wikipedia.org/wiki/REST

http://www.restapitutorial.ru/lessons/whatisrest.html

Для веб-служб, построенных с учётом REST (то есть не нарушающих накладываемых им ограничений), применяют термин «**RESTful**».

В отличие от веб-сервисов (веб-служб) на основе SOAP, не существует «официального» стандарта для RESTful веб-API. Дело в том, что REST является **архитектурным стилем**, в то время как SOAP является протоколом. Несмотря на то, что REST не является стандартом сам по себе, большинство RESTful-реализаций используют стандарты, такие как HTTP, URL, JSON и XML.

Существует **шесть** обязательных ограничений для построения распределённых REST-приложений по Филдингу. Если сервис-приложение нарушает *любое* из этих ограничительных условий, данную систему нельзя считать REST-системой.

Обязательными условиями-ограничениями являются:

1. Модель клиент-сервер

Первым ограничением применимым к нашей гибридной модели является приведение архитектуры к модели клиент-сервер, описанной в параграфе 3.4.1. Разграничение потребностей является принципом, лежащим в основе данного накладываемого ограничения. Отделение потребности интерфейса клиента от потребностей сервера, хранящего данные, повышает переносимость кода клиентского интерфейса на другие платформы, а упрощение серверной части улучшает масштабируемость. Наибольшее же влияние на всемирную паутину, пожалуй, имеет само разграничение, которое позволяет отдельным частям развиваться независимо друг от друга, поддерживая потребности в развитии интернета со стороны различных организаций.[2]

2. Отсутствие состояния

Протокол взаимодействия между клиентом и сервером требует соблюдения следующего условия: в период между запросами клиента никакая информация о *состоянии* клиента на сервере не хранится. (Stateless protocol (англ.)русск.. Все запросы от клиента должны быть составлены так, чтобы сервер получил всю необходимую информацию для выполнения запроса. *Состояние* сессии при этом сохраняется на стороне клиента.[2]. Информация о состоянии сессии может быть передана сервером какому-либо другому сервису (например в службу базы данных) для поддержания устойчивого состояния, например с целью, и на период установления аутентификации. Клиент инициирует отправку запросов, когда он готов (возникает необходимость) перейти в новое состояние.

Во время обработки клиентских запросов считается, что клиент находится в *переходном состоянии*. Каждое отдельное *состояние* приложения представлено связями, которые могут быть задействованы при следующем обращении клиента.

3. Кэширование

Как и во Всемирной паутине, клиенты а также промежуточные узлы могут выполнять кэширование ответов сервера. Ответы сервера в свою очередь должны иметь явное или неявное обозначение как кэшируемые или некэшируемые с целью предотвращения получения клиентами устаревших или неверных данных в ответ на последующие запросы. Правильное использование кэширования способно полностью или частично устранить некоторые клиент-серверные взаимодействия, ещё более повышая производительность и расширяемость системы.

4. Единообразие интерфейса

Наличие унифицированного интерфейса является фундаментальным требованием дизайна REST-сервисов.[2] Унифицированные интерфейсы позволяют каждому из сервисов развиваться независимо.

К унифицированным интерфейсам предъявляются следующие четыре ограничительных условия[9][10]:

Идентификация ресурсов

- Все ресурсы идентифицируются в запросах, например, с использованием URI в интернет-системах. Ресурсы концептуально отделены от представлений, которые возвращаются клиентам. Например, сервер может отсылать данные из базы данных в виде HTML, XML или JSON, ни один из которых не является типом хранения внутри сервера.

Манипуляция ресурсами через представление

- Если клиент хранит представление ресурса, включая метаданные — он обладает достаточной информацией для модификации или удаления ресурса.

«Самоописываемые» сообщения

- Каждое сообщение содержит достаточно информации, чтобы понять каким образом его обрабатывать. К примеру, обработчик сообщения (parser) необходимый для извлечения данных может быть указан в списке MIME-типов.[2]

Гипермедиа, как средство изменения состояния приложения (HATEOAS (англ.)русск.)

- Клиенты изменяют состояние системы только через действия, которые динамически определены в гипермедиа на сервере (к примеру, гиперссылки в гипертексте). Исключая простые точки входа в приложение, клиент не может предположить что доступна какая-то операция над каким-то ресурсом, если не получил информацию об этом в предыдущих запросах к серверу. Не существует универсального формата для предоставления ссылок между ресурсами, RFC 5988 и JSON Hypermedia API Language являются 2мя популярными форматами предоставления ссылок в REST HYPERMEDIA сервисах.

5. Слои

Клиент обычно не способен точно определить взаимодействует ли он напрямую с сервером, или же с промежуточным узлом в связи с иерархической структурой сетей (слои). Применение промежуточных серверов способно повысить масштабируемость за счет балансировки нагрузки и распределенного кэширования. Промежуточные узлы также могут подчиняться политике безопасности с целью обеспечения конфиденциальности информации.[11]

6. Код по требованию (необязательное ограничение

REST может позволить расширить функциональность клиента за счёт загрузки кода с сервера в виде апплетов или сценариев. Филдинг утверждает, что дополнительное ограничение позволяет проектировать архитектуру, поддерживающую желаемую функциональность в общем случае, но возможно за исключением некоторых контекстов.