**Лекция 13**

Теперь, после того как необходимые программные средства, такие как сервер приложений, инсталлированы, а необходимые сведения о технологии разработки получены, можно приступать к реализации нашего тестового примера с использованием технологии Web -сервисов.

Начнем, как обычно, с реализации серверной части, т.е. собственно Web -сервиса.

**Первый пример**

**Серверный класс Billing**

Ниже представлен соответствующий серверный класс ([пример 12.1](https://www.intuit.ru/studies/courses/633/489/lecture/11090?page=1#example.12.1)).

1 package com.asw.ws.ex1.endpoint;

2

3 import javax.jws.WebMethod;

4 import javax.jws.WebService;

5 import java.util.Hashtable;

6

7

8 @WebService()

9 public class Billing {

10 private Hashtable hash;

11

12 public Billing(){

13 hash = new Hashtable();

14 }

15 @WebMethod()

16 public void addNewCard(Card[] cards) {

17 for (int i=0;i<cards.length;i++)

18 hash.put(cards[i].cardNumber, cards[i]);

19 }

20 @WebMethod()

21 public void addMoney(String card, double money) {

22 Card c = (Card)hash.get(card);

23 if (c==null) {

24 System.out.println("Bad Card number\n");

25 return;

26 };

27 c.balance+=money;

28 hash.put(card,c);

29 }

30 @WebMethod()

31 public void processOperation(CardOperation[] co) {

32 for (int i=0;i<co.length;i++){

33 Card c = (Card)hash.get(co[i].card);

34 if (c==null) {

35 System.out.println("Bad Card number\n");

36 };

37 c.balance+=co[i].amount;

38 hash.put(co[i].card,c);

39 }

40 }

41

42 @WebMethod()

43 public Card getCard(String card){

44 return (Card)hash.get(card);

45 }

46 }

Листинг 12.1. Реализация серверного класса Billing

В первых строчках импортируются необходимые классы, в том числе аннотации, уже знакомые по предыдущему примеру. Сам класс аннотирован как Web -сервис (аннотация **@WebService)**,а его методы - как методы Web -сервиса (аннотация **@WebMethod)**. Код самих методов обсуждать не имеет смысла - он в точности повторяет код примеров из предыдущего раздела. Обратим только внимание на тот факт, что в качестве передаваемых в методы и возвращаемых методами аргументов используются сложные типы данных - классы. Дело в том, что используемая нами технология позволяет передавать и возвращать в качестве параметров методов не только переменные базовых типов, но и объекты созданных пользователем классов - чем мы и воспользуемся. В данном случае определено два таких "транспортных" класса: **Card**,хранящий информацию о карте и использующийся в операциях типа заведения карты и просмотра атрибутов карты, и **CardOperation**,хранящий информацию об операции изменения баланса карты. Соответствующие определения классов будут приведены ниже.

Таким образом, у разрабатываемого Web -сервиса будут определены следующие методы:

* **addNewCard** - метод, принимающий массив объектов **Card** и заносящий их в хэш-таблицу (пример метода, принимающего массив объектов пользовательского класса);
* **addMoney** - метод, принимающий код карты и начисление и изменяющий баланс карты (пример метода, принимающего значения базовых типов данных);
* **processOperation** - метод, принимающий массив объектов **CardOperation** и проводящий соответствующие операциям изменения баланса карт (пример метода, принимающего массив объектов пользовательского класса);
* **getCard** - метод, возвращающий карту со всеми атрибутами по ее коду (пример метода, возвращающего объект пользовательского класса).

Следует отметить, что поскольку класс **Billing** должен удовлетворять требованиям, предъявляемым к Web -сервису, у него определен конструктор по умолчанию (без параметров).

Ниже приведены соответствующие транспортные классы ([пример 12.2](https://www.intuit.ru/studies/courses/633/489/lecture/11090?page=1#example.12.2) и [пример 12.3](https://www.intuit.ru/studies/courses/633/489/lecture/11090?page=1#example.12.3)).

**Транспортный класс Card**

1 package com.asw.ws.ex1.endpoint;

2

3 import javax.jws.WebMethod;

4 import javax.jws.WebService;

5 import java.util.\*;

6 public class Card {

7 public Card(){}

8 public Card(String person, Date createDate, String cardNumber,double balance){

9 this.person = person;

10 this.createDate = createDate;

11 this.cardNumber = cardNumber;

12 this.balance = balance;

13 }

14 public String person;

15 public Date createDate;

16 public String cardNumber;

17 public double balance;

18 public String toString(){

19 return "Card: cardNumber="+cardNumber+"\tBalance="+balance+"\tPerson="+person+"\tCreateDate="+createDate+"";

20 }

21 }

Листинг 12.2. Транспортный класс Card

**Транспортный класс CardOperation**

1 package com.asw.ws.ex1.endpoint;

2 import java.util.\*;

3 import java.io.\*;

4

5 public class CardOperation {

6 public CardOperation(){}

7 public CardOperation(String card,double amount,Date operationDate){

8 this.card = card;

9 this.amount = amount;

10 this.operationDate = operationDate;

11 }

12 public String card;

13 public double amount;

14 public Date operationDate;

15 }

Листинг 12.3. Транспортный класс CardOperation

Следует обратить внимание на то, что у обоих транспортных классов добавлен конструктор по умолчанию - это одно из требований, накладываемых используемой технологией.

**Компиляция и инсталляция в сервере приложений**

Компиляция и развертывание приложения были подробно рассмотрены ранее, в предыдущем примере. Для компиляции этого примера использовалось инструментальное средство **ant** с разработанной системой скриптов.

Выполнив команду

asant build create-war deploy

получим следующий вывод:

Buildfile: build.xml

javaee-home-test:

init:

prepare:

[echo] Creating the required directories ...

compile-service:

[echo] Compiling the server-side source code ...

[javac] Compiling 3 source files to

H:\Java\jwstutorial20\_new\examples\jaxws\helloservice\build

[javac] Note:

H:\Java\jwstutorial20\_new\examples\jaxws\helloservice\src\Billing.java

uses unchecked or unsafe operations.

[javac] Note: Recompile with -Xlint:unchecked for details.

[wsgen] command line: wsimport -classpath

H:\Java\AppServer\lib\activation.jar;

H:\Java\AppServer\lib\admin-cli.jar;

H:\Java\AppServer\lib\appserv-admin.jar;

H:\Java\AppServer\lib\appserv-cmp.jar;

H:\Java\AppServer\lib\appserv-deployment-client.jar;

H:\Java\AppServer\lib\appserv-ext.jar;

H:\Java\AppServer\lib\appserv-jstl.jar;

H:\Java\AppServer\lib\appserv-jwsacc.jar;

H:\Java\AppServer\lib\appserv-launch.jar;

H:\Java\AppServer\lib\appserv-rt.jar;

H:\Java\AppServer\lib\appserv-tags.jar;

H:\Java\AppServer\lib\appserv-upgrade.jar;

H:\Java\AppServer\lib\appserv-ws.jar;

H:\Java\AppServer\lib\com-sun-commons-launcher.jar;

H:\Java\AppServer\lib\com-sun-commons-logging.jar;

H:\Java\AppServer\lib\dbschema.jar;

H:\Java\AppServer\lib\j2ee-svc.jar;

H:\Java\AppServer\lib\j2ee.jar;

H:\Java\AppServer\lib\javaee.jar;

H:\Java\AppServer\lib\jhall.jar;

H:\Java\AppServer\lib\jmxremote\_optional.jar;

H:\Java\AppServer\lib\jsf-impl.jar;

H:\Java\AppServer\lib\mail.jar;

H:\Java\AppServer\lib\sun-appserv-ant.jar;

H:\Java\AppServer\lib\toplink-essentials-agent.jar;

H:\Java\AppServer\lib\toplink-essentials.jar;

H:\Java\AppServer\jdk\lib\tools.jar;

H:\Java\jwstutorial20\_new\examples\jaxws\helloservice\build -d

H:\Java\jwstutorial20\_new\examples\jaxws\helloservice\build -keep -s

H:\Java\ jwstutorial20\_new\examples\jaxws\helloservice\build -verbose

com.asw.ws.ex1.endpoint.Billing

[wsgen] Note: ap round: 1

[wsgen] [ProcessedMethods Class: com.asw.ws.ex1.

endpoint.Billing]

[wsgen] [should process method: addNewCard hasWebMethods: true ]

[wsgen] [endpointReferencesInterface: false]

[wsgen] [declaring class has WebSevice: true]

[wsgen] [returning: true]

[wsgen] [WrapperGen - method: addNewCard(com.asw.ws.ex1. endpoint.Card[])]

[wsgen] [method.getDeclaringType(): com.asw.ws.ex1. endpoint.Billing]

[wsgen] [requestWrapper: com.asw.ws.ex1.endpo-int.jaxws.AddNewCard]

[wsgen] [should process method: addMoney hasWebMethods: true ]

[wsgen] [endpointReferencesInterface: false]

[wsgen] [declaring class has WebSevice: true]

[wsgen] [returning: true]

[wsgen] [WrapperGen - method: addMoney(java.lang.String,double)]

[wsgen] [method.getDeclaringType(): com.asw.ws.ex1. endpoint.Billing]

[wsgen] [requestWrapper: com.asw.ws.ex1.endpoint.jaxws. AddMoney]

[wsgen] [should process method: processOperation hasWebMethods: true ]

[wsgen] [endpointReferencesInterface: false]

[wsgen] [declaring class has WebSevice: true]

[wsgen] [returning: true]

[wsgen] [WrapperGen - method: processOperation

 (com.asw.ws.ex1.endpoint.CardOperation[])]

[wsgen] [method.getDeclaringType(): com.asw.ws.ex1.endpoint.

 Billing]

[wsgen] [requestWrapper: com.asw.ws.ex1.endpoint. jaxws.ProcessOperation]

[wsgen] [should process method: getCard hasWebMethods: true ]

[wsgen] [endpointReferencesInterface: false]

[wsgen] [declaring class has WebSevice: true] [wsgen] [returning: true]

[wsgen] [WrapperGen - method: getCard(java.lang.String)]

[wsgen] [method.getDeclaringType(): com.asw.ws.ex1.endpoint.

 Billing]

[wsgen] [requestWrapper: com.asw.ws.ex1.endpoint. jaxws.GetCard]

[wsgen] [ProcessedMethods Class: java.lang.Object]

[wsgen] com\asw\ws\ex1\endpoint\jaxws\AddMoney.java

[wsgen] com\asw\ws\ex1\endpoint\jaxws\AddMoneyResponse.java

[wsgen] com\asw\ws\ex1\endpoint\jaxws\AddNewCard.java

[wsgen] com\asw\ws\ex1\endpoint\jaxws\AddNewCardResponse.java

[wsgen] com\asw\ws\ex1\endpoint\jaxws\GetCard.java

[wsgen] com\asw\ws\ex1\endpoint\jaxws\GetCardResponse.java

[wsgen] com\asw\ws\ex1\endpoint\jaxws\ProcessOperation.java

[wsgen] com\asw\ws\ex1\endpoint\jaxws\ProcessOperationResponse.java

[wsgen] Note: ap round: 2

[wsgen] [completing model for endpoint: com.asw.ws.ex1.

 endpoint.Billing]

[wsgen] [ProcessedMethods Class: com.asw.ws.ex1.endpoint.

Billing]

[wsgen] [should process method: addNewCard hasWebMethods: true ]

[wsgen] [endpointReferencesInterface: false]

[wsgen] [declaring class has WebSevice: true]

[wsgen] [returning: true]

[wsgen] [WebServiceReferenceCollector - method:

addNewCard(com.asw.ws.ex1.endpoint.Card[])]

[wsgen] [should process method: addMoney hasWebMethods: true ]

[wsgen] [endpointReferencesInterface: false]

[wsgen] [declaring class has WebSevice: true]

[wsgen] [returning: true]

[wsgen] [WebServiceReferenceCollector - method: addMoney

 (java.lang.String,double)]

[wsgen] [should process method: processOperation hasWeb-Methods: true ]

[wsgen] [endpointReferencesInterface: false]

[wsgen] [declaring class has WebSevice: true]

[wsgen] [returning: true]

[wsgen] [WebServiceReferenceCollector - method:

 processOperation(com.asw.ws.ex1.endpoint.CardOperation[])]

[wsgen] [should process method: getCard hasWebMethods: true ]

[wsgen] [endpointReferencesInterface: false]

[wsgen] [declaring class has WebSevice: true]

[wsgen] [returning: true]

[wsgen] [WebServiceReferenceCollector - method: get-Card(java.lang.String)]

[wsgen] [ProcessedMethods Class: java.lang.Object] build-service:

build:

prepare-assemble:

[echo] Creating the assemble directory ...

[mkdir] Created dir: H:\Java\jwstutorial20\_new\examples\

 jaxws\helloservice\assemble

[mkdir] Created dir: H:\Java\jwstutorial20\_new\examples\

 jaxws\helloservice\assemble\war

create-war:

[echo] Creating the WAR ...

[war] Building war: H:\Java\jwstutorial20\_new\examples\

 jaxws\helloservice\assemble\war\billing-jaxws.war

deploy:

admin\_command\_common:

[echo] Doing admin task deploy assemble/war/

 billing-jaxws.war

[sun-appserv-admin] Executing: deploy --port 4848 --host

localhost --passwordfile "H:\Java\jwstutorial20\_new\examples\

common\admin-password.txt" --user admin assemble/war/

billing-jaxws.war

[sun-appserv-admin] Command deploy executed successfully.

BUILD SUCCESSFUL

Total time: 22 seconds

Одной командой мы сразу откомпилировали, упаковали в необходимую для Web -приложения структуру и разместили на сервере приложений наш пример. Следует обратить внимание на количество автоматически сгенерированных утилитой **wsgen** классов - на каждый метод Web -сервиса их сгенерировано по 2.

Теперь можно воспользоваться административной консолью, чтобы убедиться, что наш Web -сервис успешно инсталлирован ([рис. 12.1](https://www.intuit.ru/studies/courses/633/489/lecture/11090?page=1#image.12.1)).



**Рис. 12.1.** Проверка инсталляции серверного класса



**Рис. 12.2.** Тестовая страница

Воспользоваться автоматически построенной сервером тестовой страницей ([рис. 12.2](https://www.intuit.ru/studies/courses/633/489/lecture/11090?page=1#image.12.2)) нам, к сожалению, не удастся, так как наши операции требуют передачи серверу объектов.

**Клиентский класс BillingClient**

При реализации клиента первая проблема состоит в том, что вызываемые методы Web -сервиса принимают аргументы - объекты. Вторая проблема состоит в том, что некоторые из этих методов принимают массивы объектов.

Первая проблема возникает из-за того, что на самом деле клиентское приложение не получает доступа к "настоящим" классам сервера - т.е. клиент не имеет доступа к классам **com.asw.ws.ex1.endpoint.Card** или **com.asw.ws.ex1.endpoint.CardOperation**,рассмотренным в предыдущем разделе и реализованным на сервере. Вместо этого утилитой wsimport на основании анализа WSDL файла генерируются другие классы. Они тоже располагаются в пакете **com.asw.ws.ex1.endpoint** (но клиента) и называются так же - **Card** и **CardOperation**.Но предназначены они только для того, чтобы служить оберткой для передаваемых данных. В этих сгенерированных классах будут содержаться все поля классов исходных, но в них не будет содержаться методов, которые определены в "настоящих" классах, располагающихся на сервере. Все, что они будут содержать, - методы установки и получения соответствующих полей. По большому счету, это даже не классы, а структуры, предназначенные для передачи параметров. Ниже для иллюстрации приведен один такой сгенерированный класс для **Card**.

Вторая проблема возникает из-за того, что при генерации структура вида "массив объектов" (Т[]) преобразуется в "список объектов" **(List<T>)**.Поэтому в соответствующие методы нужно передавать не массивы, а классы, производные от **List**,- например, **Vector**.

1 package com.asw.ws.ex1.client;

2

3 import javax.xml.ws.WebServiceRef;

4 import com.asw.ws.ex1.endpoint.BillingService;

5 import com.asw.ws.ex1.endpoint.Billing;

6 import com.asw.ws.ex1.endpoint.Card;

7 import com.asw.ws.ex1.endpoint.CardOperation;

8 import javax.xml.datatype.XMLGregorianCalendar;

9 import java.util.\*;

10

11 public class BillingClient {

12 @WebServiceRef(wsdlLocation="http://localhost:8080/billingservice/billing?wsdl")

13 static BillingService service;

14

15 public static void main(String[] args) {

16 try {

17 BillingClient client = new BillingClient();

18 client.startTest();

19 } catch(Exception e) {

20 e.printStackTrace();

21 }

22 }

23

24 public void startTest() {

25 try {

26 Billing port = service.getBillingPort();

27

28 Vector<Card> vc = new Vector<Card>();

29 Card c = new Card();

30 c.setPerson("Piter");

31 XMLGregorianCalendar dat = javax.xml.datatype.DatatypeFactory.newInstance().newXMLGregorianCalendar();

32 dat.setYear(2006);

33 dat.setMonth(11);

34 dat.setDay(5);

35 c.setCreateDate( dat );

36 c.setCardNumber("1");

37 c.setBalance(0.0);

38 vc.add(c);

39

40 c = new Card();

41 c.setPerson("Stefan");

42 c.setCreateDate( dat );

43 c.setCardNumber("2");

44 c.setBalance(0.0);

45 vc.add(c);

46

47 c = new Card();

48 c.setPerson("Nataly");

49 c.setCreateDate( dat );

50 c.setCardNumber("3");

51 c.setBalance(0.0);

52 vc.add(c);

53

54

55 port.addNewCard(vc);

56

57

58 int cnt = 30000;

59 Vector<CardOperation> vco = new Vector<CardOperation>();

60 CardOperation co;

61 for (int i = 0; i < cnt; i++) {

62 co = new CardOperation();

63 switch (i%3){

64 case 0: co.setCard("1"); co.setAmount(1); co.setOperationDate(dat);vco.add(co);break;

65 case 1: co.setCard("2"); co.setAmount(2); co.setOperationDate(dat);vco.add(co);break;

66 case 2: co.setCard("3"); co.setAmount(3); co.setOperationDate(dat);vco.add(co);break;

67 }

68 }

69

70 port.processOperation(vco);

71

72 printCard(port.getCard("1"));

73 printCard(port.getCard("2"));

74 printCard(port.getCard("3"));

75

76

77

78 } catch(Exception e) {

79 e.printStackTrace();

80 }

81 }

82 public void printCard(Card card){

83 System.out.println(card.getCardNumber()+"\t"+card.getPerson()+"\t"+

84 card.getCreateDate()+"\t"+card.getBalance());

85 }

86 }

Листинг 12.4. Клиентский класс BillingClient

Сначала подключается соответствующая аннотация ([пример 12.4](https://www.intuit.ru/studies/courses/633/489/lecture/11090?page=2#example.12.4)). Именно с ее помощью (строка 12) мы указываем месторасположения WSDL -файла интересующего нас Web -сервиса, на основании анализа которого генерируются все вспомогательные классы. Затем подключаются классы пакета **com.asw.ws.ex1.endpoint** - все эти классы сгенерированы автоматически утилитой **wsimport** (еще раз повторимся: это не те классы, что мы написали при реализации Web -сервиса).

В **main** создается экземпляр нашего класса **BillingClient** и запускается его метод **startTest** (строка 18). Метод **startTest** первым делом создает клиентскую заглушку (строка 26), или proxy -класс, - класс, который реализует такие же методы, как и нужный нам Web -сервис, и который все их вызовы пересылает Web -серверу. И этот класс, и класс **BillingService**,который, собственно и создает клиентский proxy, созданы автоматически.

После того как клиентская заглушка создана, мы можем вызывать ее методы. Поскольку первый метод, который нам нужно вызвать - метод **addNewCard**,а он принимает в качестве параметра массив, для его вызова предстоит выполнить некоторые подготовительные действия. Сначала создается вектор, который мы будем передавать в качестве параметра (строка 28 )1. Затем последовательно создаются три экземпляра **Card**,поля которых заполняются нужными значениями. Эти карты добавляются в вектор. Вызов addNewCard (строка 55) передает созданные карты Web -сервису и заставляет его создать эти карты.

Похожим образом создаются 30000 операций с картами - по 10000 на каждую из созданных нами трех карт. Вызов processOperation (строка 70) передает созданный массив Web -серверу, а тот, в свою очередь, обработав их, изменяет балансы карт.

И наконец, вызовы метода getCard (строки 72-74) возвращают на клиента карты, которые распечатываются.

**Компиляция и выполнение клиента**

Так же как и для компиляции сервера, для компиляции клиента воспользуемся ant'ом. Соответствующая строка компиляции выглядит следующим образом (компиляция является целью по умолчанию):

asant

Вывод получим следующий

Buildfile: build.xml clean:

javaee-home-test:

init:

prepare:

[echo] Creating the required directories....

[mkdir] Created dir: H:\Java\jwstutorial20\_new\examples\

jaxws\simpleclient\build

generate-client: [wsimport] command line: wsimport

H:\Java\AppServer\jdk\jre\bin\java.exe -classpath

H:\Java\AppServer\lib\activation.jar;

H:\Java\AppServer\lib\admin-cli.jar;

H:\Java\AppServer\lib\appserv-admin.jar;

H:\Java\AppServer\lib\appserv-ext.jar;

H:\Java\AppServer\lib\jmxremote\_optional.jar;

H:\Java\AppServer\lib\appserv-cmp.jar;

H:\Java\AppServer\lib\appserv-deployment-client.jar;

H:\Java\AppServer\lib\appserv-jstl.jar;

H:\Java\AppServer\lib\appserv-jwsacc.jar;

H:\Java\AppServer\lib\appserv-launch.jar;

H:\Java\AppServer\lib\appserv-rt.jar;

H:\Java\AppServer\lib\appserv-tags.jar;

H:\Java\AppServer\lib\appserv-upgrade.jar;

H:\Java\AppServer\lib\appserv-ws.jar;

H:\Java\AppServer\lib\com-sun-commons-launcher.jar;

H:\Java\AppServer\lib\com-sun-commons-logging.jar;

H:\Java\AppServer\lib\dbschema.jar;

H:\Java\AppServer\lib\j2ee-svc.jar;

H:\Java\AppServer\lib\j2ee.jar;

H:\Java\AppServer\lib\javaee.jar;

H:\Java\AppServer\lib\jhall.jar;

H:\Java\AppServer\lib\jsf-impl.jar;

H:\Java\AppServer\lib\mail.jar;

H:\Java\AppServer\lib\sun-appserv-ant.jar;

H:\Java\AppServer\lib\toplink-essentials-agent.jar;

H:\Java\AppServer\lib\toplink-essentials.jar;

H:\Java\AppServer\jdk\lib\tools.jar com.sun.tools.ws.WsImport -d

H:\Java\jwstutorial20\_new\examples\

 jaxws\simpleclient\build -g -keep -s

H:\Java\jwstutorial20\_new\examples\jaxws\simpleclient\src -verbose

http://localhost:8080/billingservice/billing?wsdl -p com.asw.ws.ex1.endpoint

[wsimport] com\asw\ws\ex1\endpoint\AddMoney.java

[wsimport] com\asw\ws\ex1\endpoint\AddMoneyResponse.java

[wsimport] com\asw\ws\ex1\endpoint\AddNewCard.java

[wsimport] com\asw\ws\ex1\endpoint\AddNewCardResponse.java

[wsimport] com\asw\ws\ex1\endpoint\Billing.java

[wsimport] com\asw\ws\ex1\endpoint\BillingService.java

[wsimport] com\asw\ws\ex1\endpoint\Card.java

[wsimport] com\asw\ws\ex1\endpoint\CardOperation.java

[wsimport] com\asw\ws\ex1\endpoint\GetCard.java

[wsimport] com\asw\ws\ex1\endpoint\GetCardResponse.java

[wsimport] com\asw\ws\ex1\endpoint\ObjectFactory.java

[wsimport] com\asw\ws\ex1\endpoint\ProcessOperation.java

[wsimport] com\asw\ws\ex1\endpoint\ProcessOperationResponse.java

[wsimport] com\asw\ws\ex1\endpoint\package-info.java

[wsimport] com\asw\ws\ex1\endpoint\AddMoney.java

[wsimport] com\asw\ws\ex1\endpoint\AddMoneyResponse.java

[wsimport] com\asw\ws\ex1\endpoint\AddNewCard.java

[wsimport] com\asw\ws\ex1\endpoint\AddNewCardResponse.java

[wsimport] com\asw\ws\ex1\endpoint\Billing.java

[wsimport] com\asw\ws\ex1\endpoint\BillingService.java

[wsimport] com\asw\ws\ex1\endpoint\Card.java

[wsimport] com\asw\ws\ex1\endpoint\CardOperation.java

[wsimport] com\asw\ws\ex1\endpoint\GetCard.java

[wsimport] com\asw\ws\ex1\endpoint\GetCardResponse.java

[wsimport] com\asw\ws\ex1\endpoint\ObjectFactory.java

[wsimport] com\asw\ws\ex1\endpoint\ProcessOperation.java

[wsimport] com\asw\ws\ex1\endpoint\ProcessOperationResponse.java

[wsimport] com\asw\ws\ex1\endpoint\package-info.java

compile-client:

[echo] Compiling the client source code ...

[javac] Compiling 1 source file to

H:\Java\ jwstutorial20\_new\examples\jaxws\simpleclient\build

prepare-dist:

[echo] Creating the required directories ...

[mkdir] Created dir:

H:\Java\jwstutorial20\_new\ examples\jaxws\simpleclient\dist

package-client:

[echo] Building the client JAR file ...

[jar] Building jar:

H:\Java\jwstutorial20\_new\examples\jaxws\simpleclient\dist\client.jar

build-client:

build:

BUILD SUCCESSFUL

Total time: 16 seconds

В приведенном выводе можно увидеть работу утилиты wsimport по созданию вспомогательных файлов по WSDL Web -сервиса. Запустив клиента командой asant run

видим, что все необходимые операции на Web -сервисе - создание карт, изменение балансов и запрос информации по картам - успешно выполнены:

Buildfile: build.xml

run:

appclient-command-common:

[exec] 1 Piter 2006-11-05T00:00:00+03:00 10000.0

[exec] 2 Stefan 2006-11-05T00:00:00+03:00 20000.0

[exec] 3 Nataly 2006-11-05T00:00:00+03:00 30000.0

BUILD SUCCESSFUL

Total time: 23 seconds