J. Reichardt
Fachbereich Informatik
Hochschule Darmstadt

ANLEITUNG
ZUM 2. PRAKTIKUMSVERSUCH VERTEILTE SYSTEME

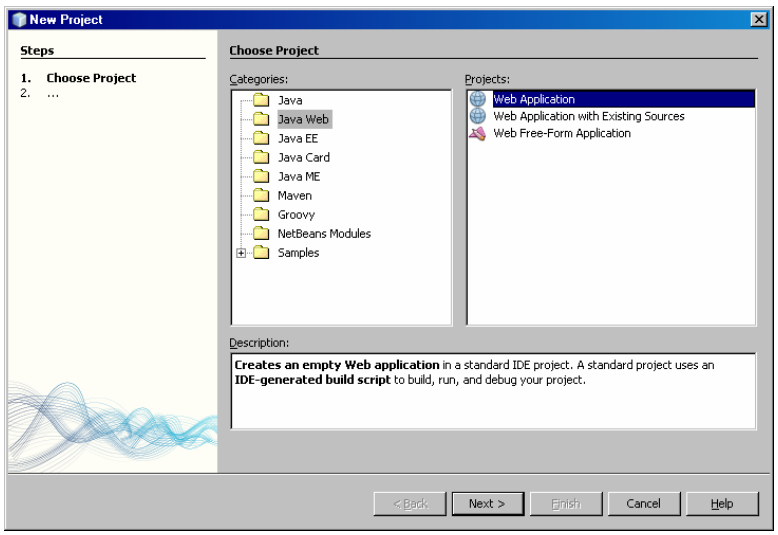
TaskBag-Webserver

mit JAX-WS und NetBeans

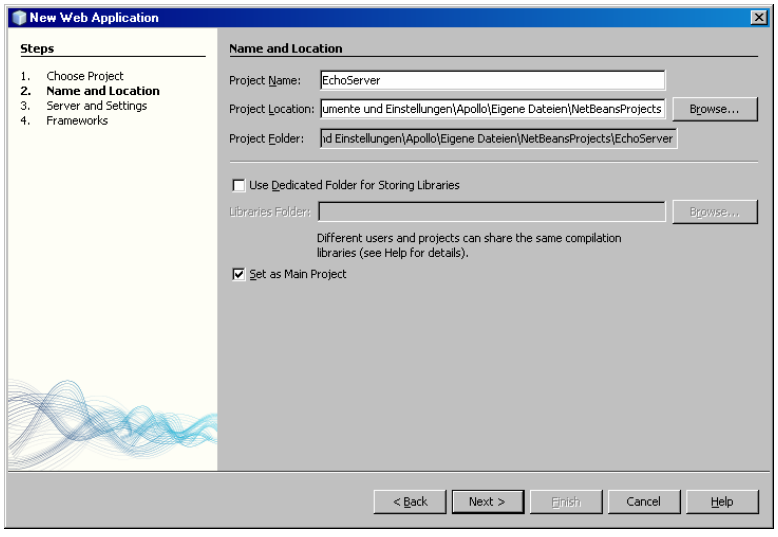
TEIL A: Zunächst als einfaches Beispiel einen Echo-Webservice realisieren.

A.1 Echo-Webserver erstellen

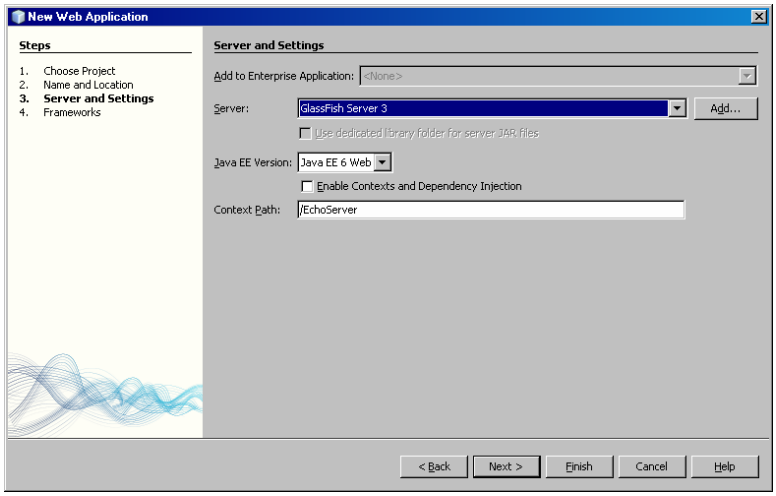
- **Projektdateien und –verzeichnisse erzeugen**
  - File -> New Project (*beachten: Java Web und Web Application auswählen*)



  - Next

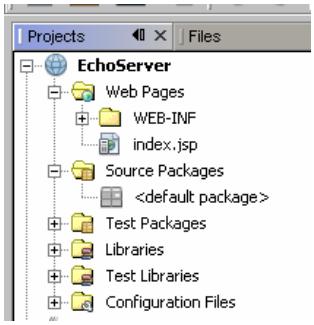


- Next

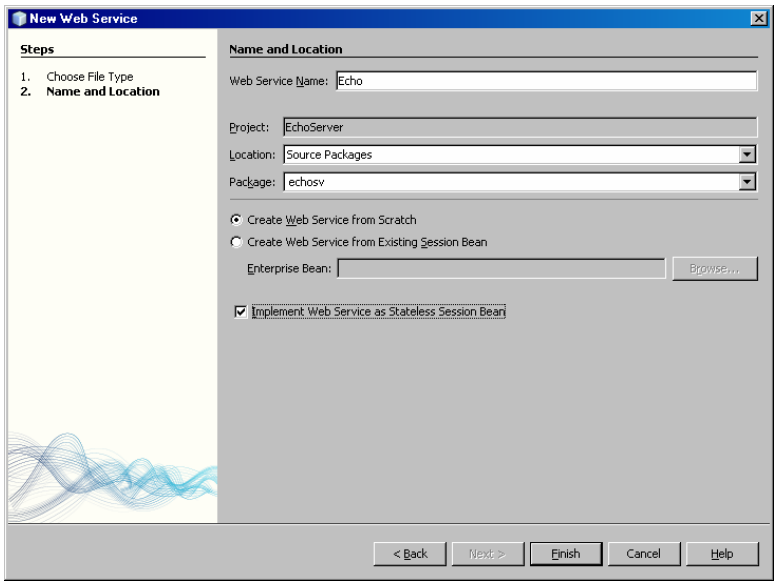


- Finish

- Folgende Projektdateien und –verzeichnisse wurden erzeugt:

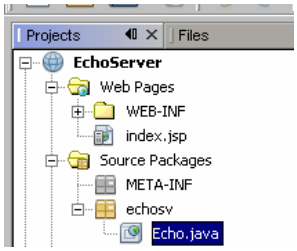


- **Echo-Klasse erzeugen**
  - Rechte Maustaste auf Ordner ‚Source Packages' -> New -> **Web Service**

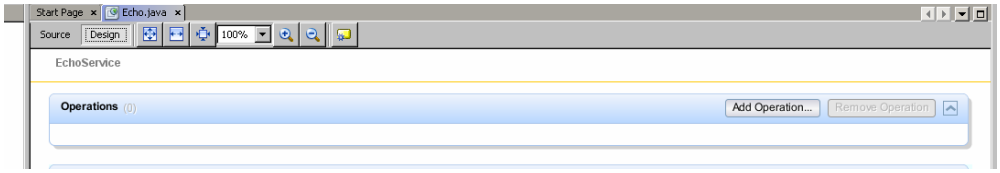


- Der Web Service Name ‚Echo' wird von NetBeans intern noch durch das Wort ‚Service' ergänzt.
- Die Implementierung als Stateless Session Bean ist für Teil B (Dependency Injection und Namensgebung der Methoden) wesentlich.

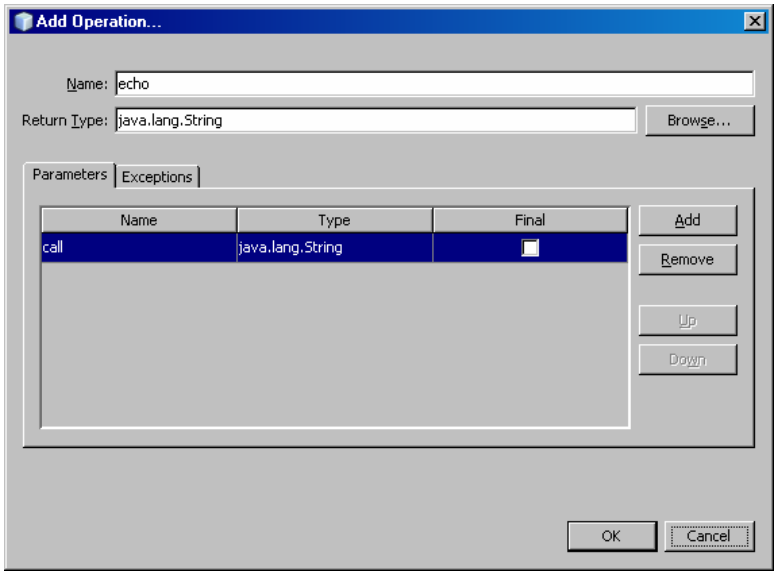
- Finish
- Im Verzeichnis ‚Source Packages' wurde die Datei ‚Echo.java' mit der Klasse ‚Echo' erzeugt:



- **Echo-Methode erzeugen**
  - Datei ‚Echo.java' öffnen.
  - ‚Design' wählen:



  - ‚Add Operation' wählen:



  - In Datei ‚Echo.java' wähle ‚Source'.
  - In Webmethode ‚echo' ersetze ‚return null' durch ‚return call' ('call' ist der Eingabeparameter).

- **Echo-Server bereitstellen**
  - Rechte Maustaste auf Ordner 'EchoServer' -> Deploy. Der Output-Log von GlassFish Server 3 ist:

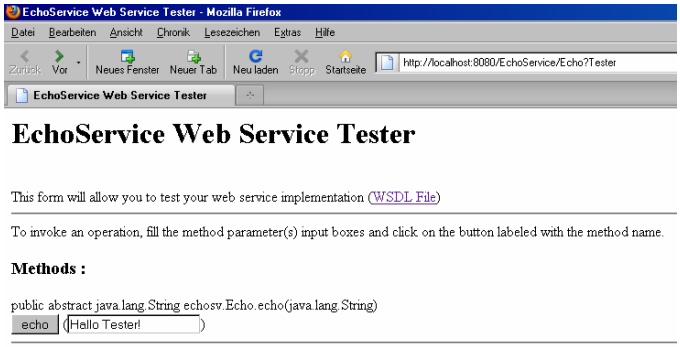
```
INFO: WS00019: EJB Endpoint deployed
 EchoServer  listening at address at http://localhost:8080/EchoService/Echo
INFO: Loading application EchoServer at /EchoServer
INFO: EchoServer was successfully deployed in 687 milliseconds.
```

- Die URL in der Ausgabe des GlassFish Server 3 wird in Teil A.2 für den Zugriff auf die WSDL-Datei des Servers benötigt; deshalb bitte diese URL vormerken!

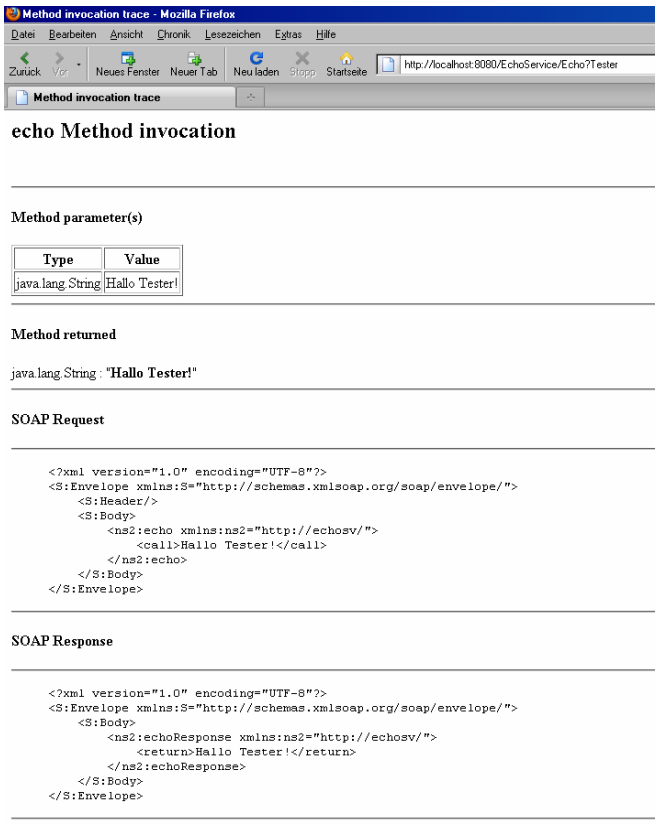
- Diese URL wird ebenso für den folgenden Aufruf des generierten Webservice-Testers benötigt.

- **Echo-Server testen**
  - Bevor der Echo-Client erstellt wird, kann der Server bereits durch ein von NetBeans generiertes Testtool (`Tester`) getestet werden:

  - Browser starten. Obige URL um ‚`?Tester`' ergänzen: `http://localhost:8080/Echo-Service/Echo?Tester` (beim Start des Testers von einer entfernten Plattform aus ist in der URL die Angabe ‚localhost' durch die IP-Adresse des Servers zu ersetzen):



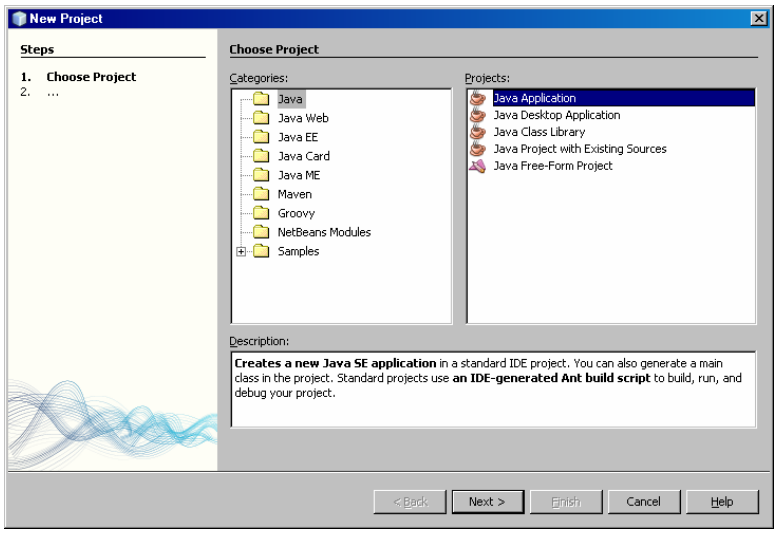
  - Echo-Parameter eingeben und Echo-Funktion starten:



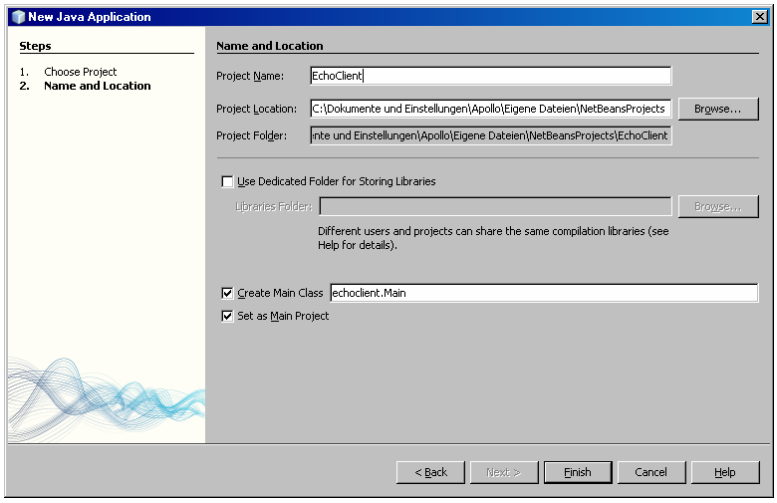
  - Inspizieren Sie das WSDL-File zum weiteren Kennenlernen des SOAP-Protokolls:
    - Überzeugen Sie sich, dass die <`soap:address`> des Echo-Service mit der vom GlassFish Server ausgegebenen URL übereinstimmt.
    - Öffnen Sie im Browser mit der unter <`schemaLocation`> angegebenen URL (`http://localhost:8080/EchoService/Echo?xsd=1`) die XMLSchema-Datei, welche die Syntaxregeln für die SOAP-Kodierung der Echo-Methode enthält.
    - Überzeugen Sie sich, dass der obige SOAP-Request und die SOAP-Response diese Syntax-regeln erfüllen.

A.2 Echo-Java-Client erstellen

- **Projektdateien und –verzeichnisse erzeugen**
  - File -> New Project (*beachten: Java und Java Application auswählen*)

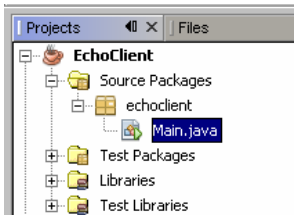


  - Next

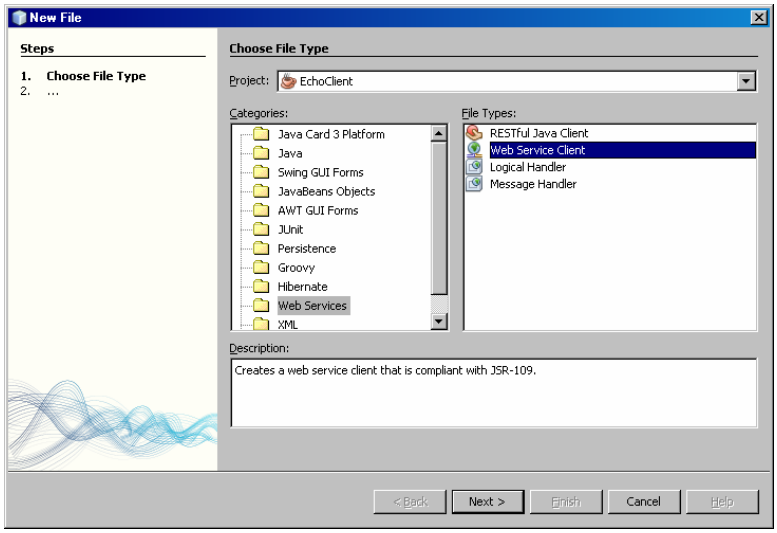


  - Finish

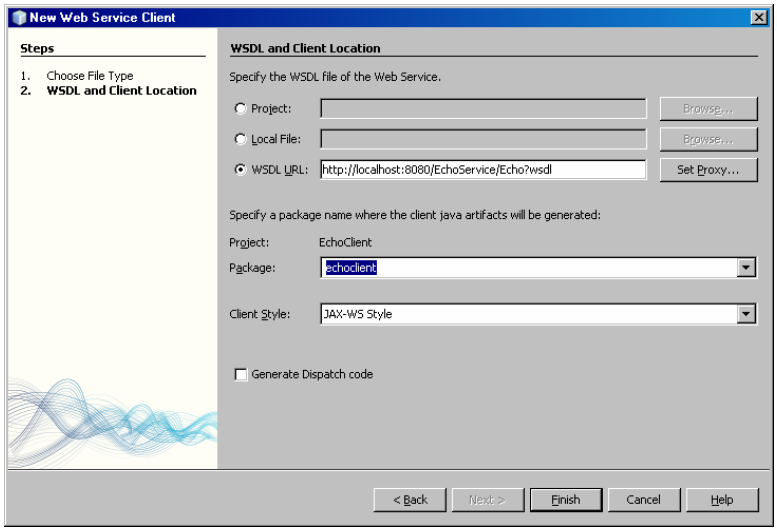
  - Folgende Projektdateien und –verzeichnisse wurden erzeugt:



- **WSDL-File des Echo-Servers importieren**
  - Rechte Maustaste auf Ordner ‚EchoClient' -> New -> **Other**
    (*beachten: Web Services und Web Service Client auswählen*)

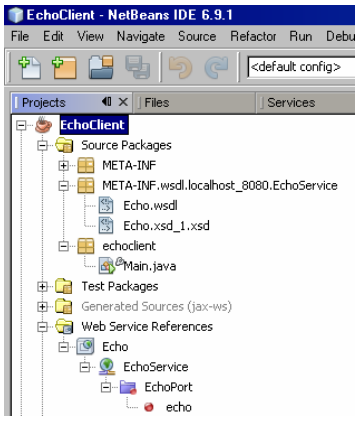


  - Next



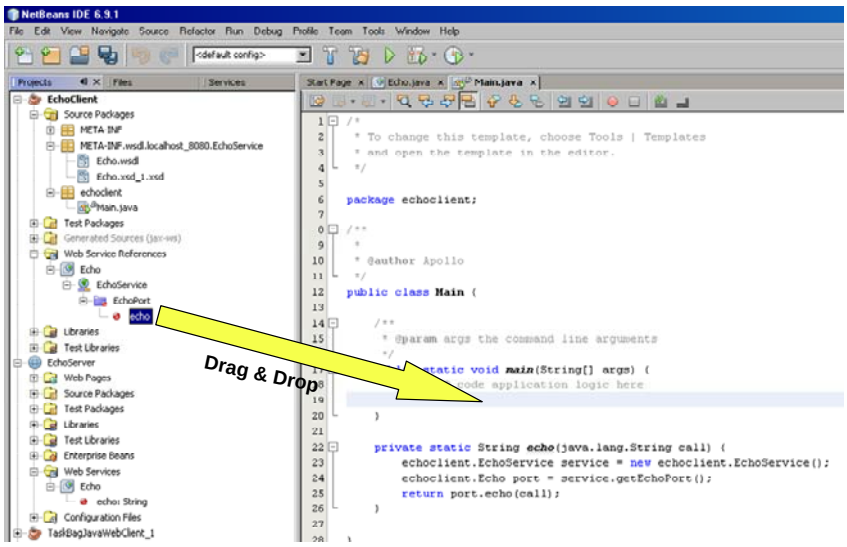
  - WSDL URL: Eingabe der URL, die beim Deployen des Echo-Servers im GlassFish Server-Logfile ausgegeben wurde (siehe oben). -> Finish

  - Das WSDL-File wurde importiert und eine Service-Referenz erzeugt:



- **Aufruf der Service-Methode *echo()***
  - Öffnen der EchoClient-Datei ‚Main.java' und Expandieren der ‚Web Service References' bis zum Methoden-Port ‚echo'.

  - Leerzeile in ‚main()' einfügen, Klick auf den Methoden-Port (rote Kugel) und Drag&Drop des Ports in die Leerzeile von ‚main()'.



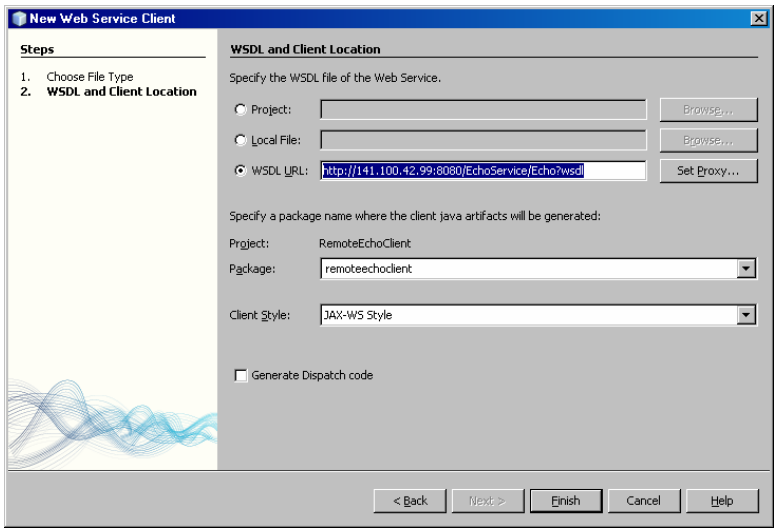
  - In der Klasse ‚Main' wurde außerhalb der Methode ‚main()' eine Definition für die Methode ‚echo()' eingefügt. Die Leerzeile in ‚main()' ist weiterhin leer. In diese Leerzeile kann nun ein Aufruf der Methode ‚echo()' eingefügt werden, z.B. innerhalb einer Ausgabeanweisung mit ‚System.out.println()'.

- **Starten des EchoClients**
  - Rechte Maustaste auf Ordner ‚EchoClient' -> Run.

A.3 Remote Echo-Java-Client erstellen

- **Projektdateien und –verzeichnisse erzeugen** (wie für lokalen Client)

- **WSDL-File des entfernten Echo-Servers importieren**
  - Anstatt ‚localhost' wird in der WSDL-URL die IP-Adresse des entfernten Servers angegeben. NetBeans setzt voraus, dass der Server unter derselben IP-Adresse wie die WSDL-Datei zu erreichen ist (in der WSDL-Datei unter `<soap:address>` nachzuprüfen). Alles weitere wie in A.2.

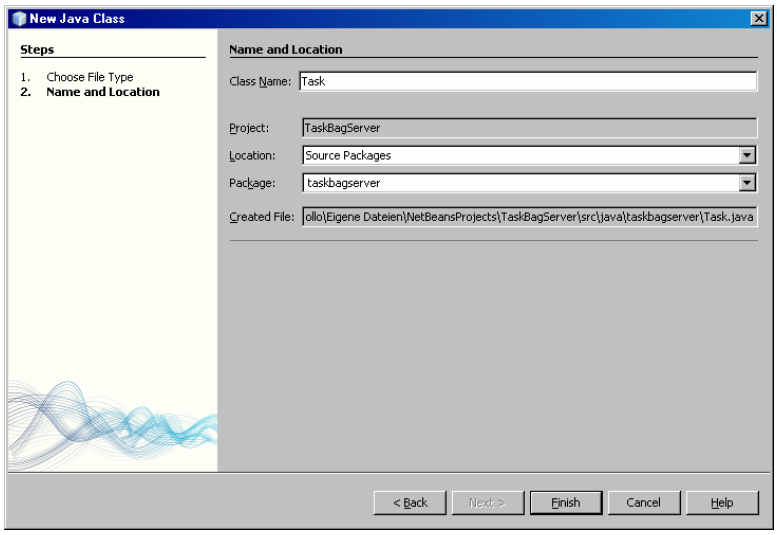


TEIL B: TaskBag-Webservice realisieren

B.1 TaskBag-Webserver erstellen

- **Projektdateien und –verzeichnisse erzeugen** (wie für Echo-Service)
- **TaskBag-Klasse erzeugen** (wie Echo-Klasse)
- **TaskBag-Methoden putTask(), getTask() und readTask() erzeugen** (wie *echo()*-Methode)

- **Task als Tabellenstruktur erstellen**
  - Rechter Mausklick auf ‚Source Packages/taskbagserver' -> New -> **Java Class**

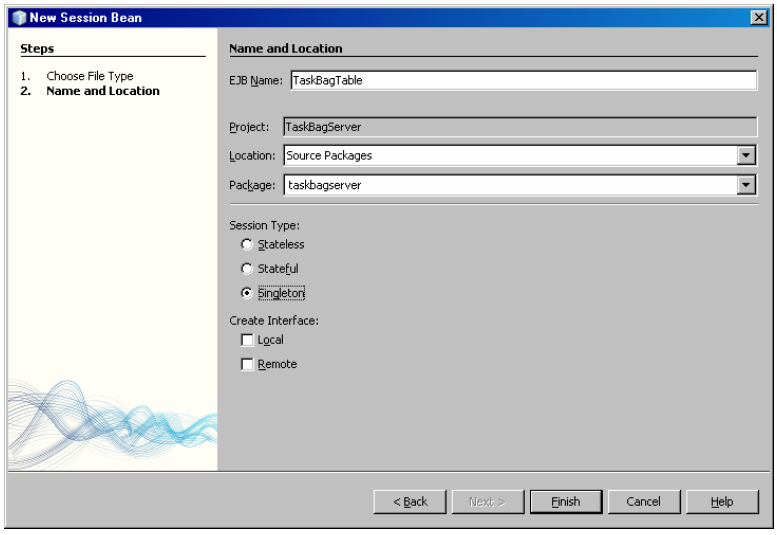


  - Finish

  - Datei 'Task.java' öffnen und Strukturelemente `Id`, `Type`, `Descr` und `Done` (siehe Aufgabenstellung) als Attribute der Klasse `Task` einfügen. Danach eine Leerzeile einfügen.

  - Rechter Mausklick auf die Leerzeile -> Insert Code -> **Getter and Setter**. Für alle Attribute Get- und Set-Methoden generieren lassen. Damit ist die Task-Struktur vollständig definiert.

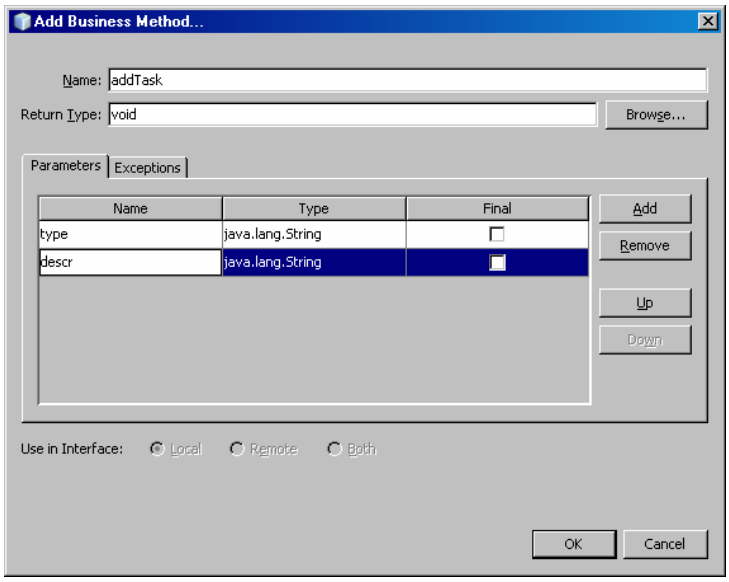
- **TaskBagTable als Singleton-Speichermedium** (an Stelle einer Datei) **erstellen**
  - Rechter Mausklick auf ‚Source Packages/taskbagserver' -> New -> **SessionBean**



  - Finish

- In der Datei ‚TaskBagTable.java' die Schnittstelle für die ‚TaskBagTable' definieren. Vorschlag:
  - `void addTask(String type, String descr);`
  - `String getNextTaskByType(String type);`
  - `String readNextTaskByType(String type);`

- Dazu rechter Mausklick innerhalb der Klasse 'TaskBagTable' -> Insert Code -> **Add Business Method**



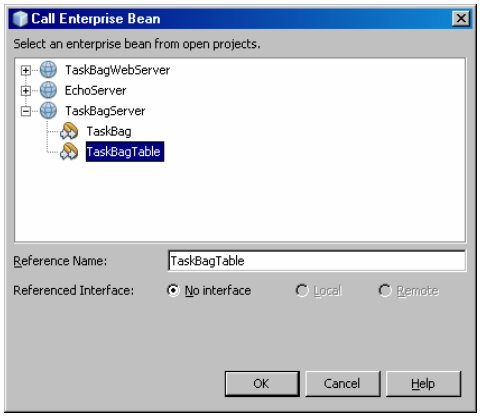
- OK (zwei mal), und für jede weitere Methode ‚Add Business Method' wiederholen.

- In der Datei ‚TaskBagTable.java' als Attribut der Klasse ‚TaskBagTable' die physikalische TaskBag-Tabelle definieren. Vorschlag:

```
private Map<Integer, Task> taskbag = new HashMap<Integer, Task>();
private int maxkey = 0; // zeigt stets auf den ersten freien Eintrag
```

- Die Methoden der TaskBagTable-Schnittstelle sind auszuprogrammieren.

- **Singleton ‚TaskBagTable' benutzen**
  - In Datei ‚TaskBag.java' rechte Maustaste auf eine Leerzeile am Anfang der TaskBag-Klasse -> Insert Code -> **Call Enterprise Bean**



- OK. Dadurch wird in der Webservice-Klasse ‚TaskBag' in Form einer EJB-Annotation eine Referenz auf die ‚TaskBagTable' hergestellt, die beim Deployen per Dependency-Injection initialisiert wird.

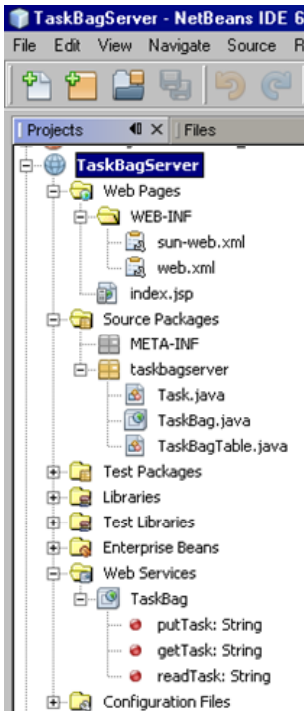
- Die Webservice-Klasse ‚TaskBag' sollte nun folgende Form haben:

```
@WebService()
@Stateless()
public class TaskBag {

@EJB
private TaskBagTable taskBagTable;

/**
* Web service operation
*/
@WebMethod(operationName = "putTask")
public String putTask(@WebParam(name = "type")
String type, @WebParam(name = "descr")
String descr) {
      //TODO write your implementation code here:
      return "ok";
}
…
}
```

- Insgesamt wurden bisher folgende Projektverzeichnisse und –dateien erzeugt:



- Rechte Maustaste auf ‚TaskBagServer' -> Clean and Build.

- Damit steht die Architektur des TaskbagServers bereit, mit allen Komponenten und ihren Kommunikations- und Benutzungsbeziehungen. Jetzt sind nur noch die Methoden der TaskBag-Schnittstelle auszuprogrammieren.

- **TaskBag-Methoden ausprogrammieren**

  - Die TaskBag-Methoden getTask() und readTask() sind **server-seitig blockierend** zu implementieren, das heißt, dass die Implementierung von getTask() innerhalb einer Schleife solange die TaskBagTable-Methode getNextTaskByType() aufruft bis diese einen Returnwert ungleich `null` zurückgibt. Erst dann erhält ein Client den Returnwert von getTask() und wird damit entblockiert.

Entsprechendes gilt für die Methode readTask().

- Die TaskBagTable-Methoden getNextTaskByType(), usw. sind dagegen nicht-blockierend zu implementieren, d.h. ihr Aufruf liefert immer sofort einen Returnwert (`null` oder ungleich `null`).

- **TaskBagServer bereitstellen und testen**
  - Rechte Maustaste auf TaskBagServer -> Deploy
  - Die URL des TaskBagServers in der Ausgabe von GlassFishServer3 für den späteren WSDL-Import auf Client-Seite vormerken (wie für Echo-Server und –Client)
  - Tester-Aufruf (wie für Echo-Server)
  - Blockieren und Entblockieren der Serveraufrufe testen:
    - putTask(garten, rasenmaehen) // Trägt einen Task ein
    - getTask(garten) // Liest und löscht den Task
    - getTask(garten) // 2. Aufruf blockiert, da kein passender Task vorhanden
    - putTask(garten, rasenmaehen) // Entblockiert den 2. Aufruf durch Eintrag eines passen-
      // den Task

B.2 TaskBag-Java-Clients erstellen und testen

Es sind zwei Clients zu erstellen, die später separat in unterschiedlichen Reihenfolgen gestartet werden, um das Blockieren bzw. Entblockieren der Aufrufe zu testen. Der erste Client soll zunächst nur getTask() aufrufen, der zweite Client putTask().

- **Projektdateien und –verzeichnisse für Client 1 erzeugen** (wie für Echo-Client)
- **In Client 1 das WSDL-File des Echo-Servers importieren** (wie für Echo-Client)
- **Aufruf der Service-Methode *getTask()*** (wie *echo()*-Methode)

- **Projektdateien und –verzeichnisse für Client 2 erzeugen** (wie oben)
- **In Client 2 das WSDL-File des Echo-Servers importieren** (wie oben)
- **Aufruf der Service-Methode *putTask()*** (wie oben)

- **Testen des TaskBag-Webservice mit den beiden Java-Clients gemäß Aufgabenstellung.**
  Zur Info: Für jeden Aufruf einer Web-Methode wird vom GlassFish-Application-Server aus einem Pool eine neue TaskBagBean geholt (vergleichbar mit dem Start eines neuen Prozesses oder Threads). Alle diese Komponenten greifen stets auf dieselbe ‚TaskBagTable' (Singleton) zu. Diese Zugriffe sind, im Gegensatz zu den Dateizugriffen in Aufgabe 1, default-mäßig voll synchronisiert (container-managed concurrency).

B.3 Remote TaskBag-Java-Clients erstellen und testen

Wie in A.3.