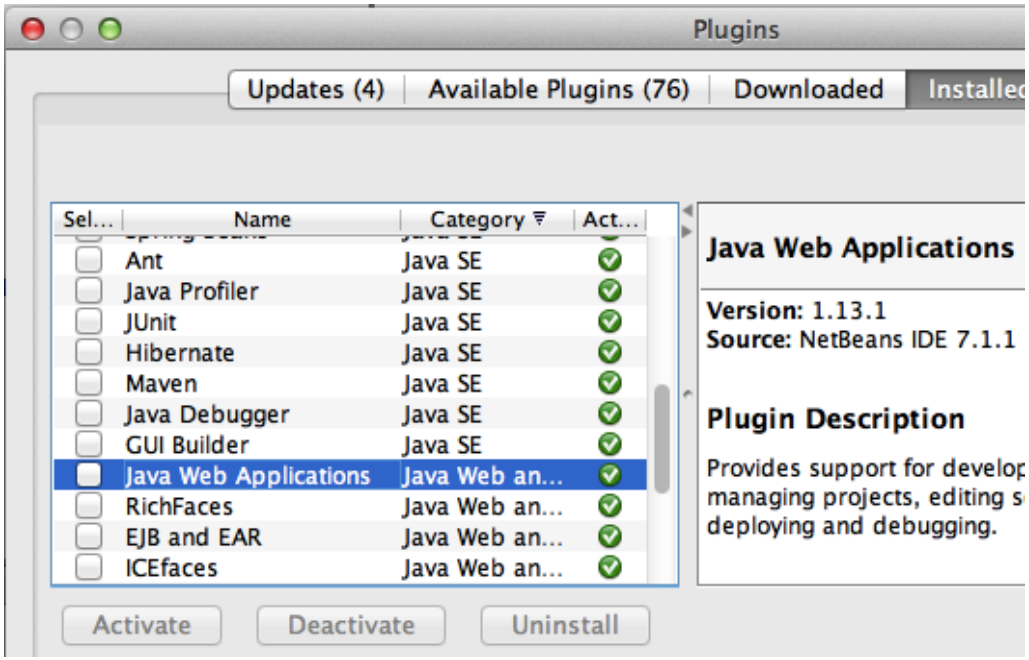
Pratica con i Web Services

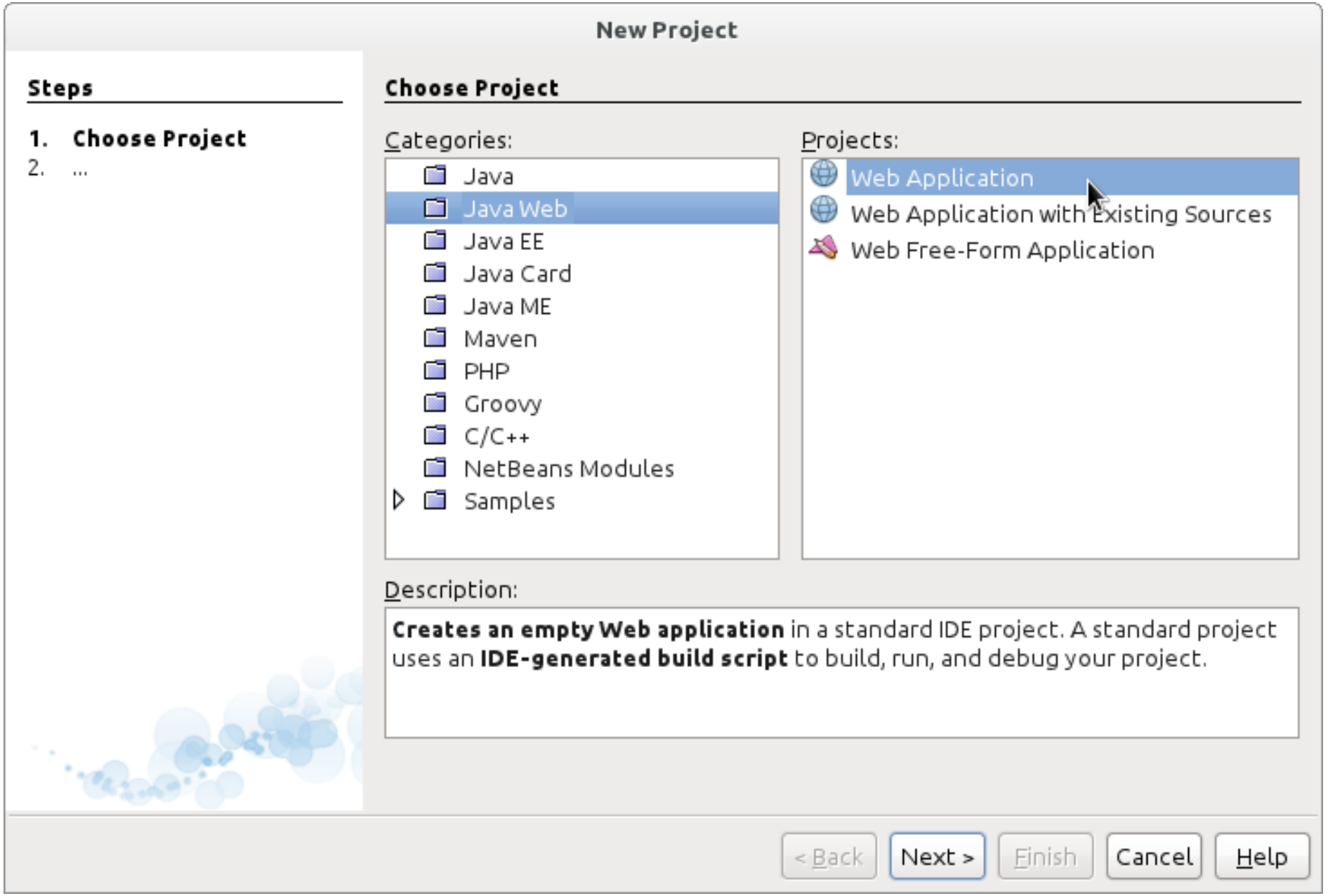
Netbeans: funzionalità richieste

- Se necessario, attivare plugin *Java Web App*, via menu *Tools-Plugins*
- Se necessario, installare un server (per i WS meglio *GlassFish*), via menu *Tools-Servers*

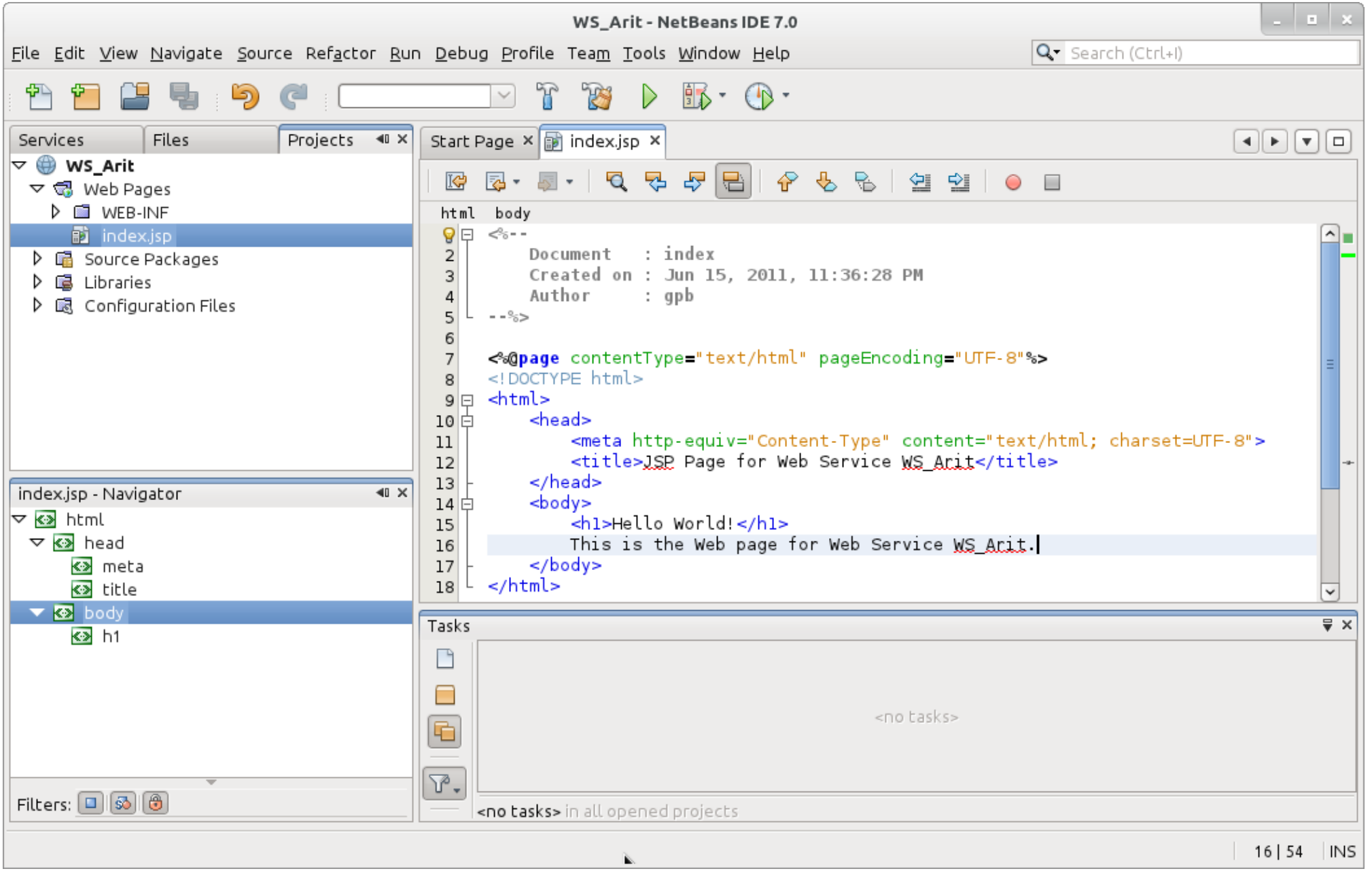


- Si può usare anche Tomcat, ma è meno integrato, p.es. non consente il test da Netbeans dei WS
- N.B.: in un progetto con WS già esistente, cambiare server “in corsa” (p.es. da GlassFish a Tomcat) tende a dare problemi

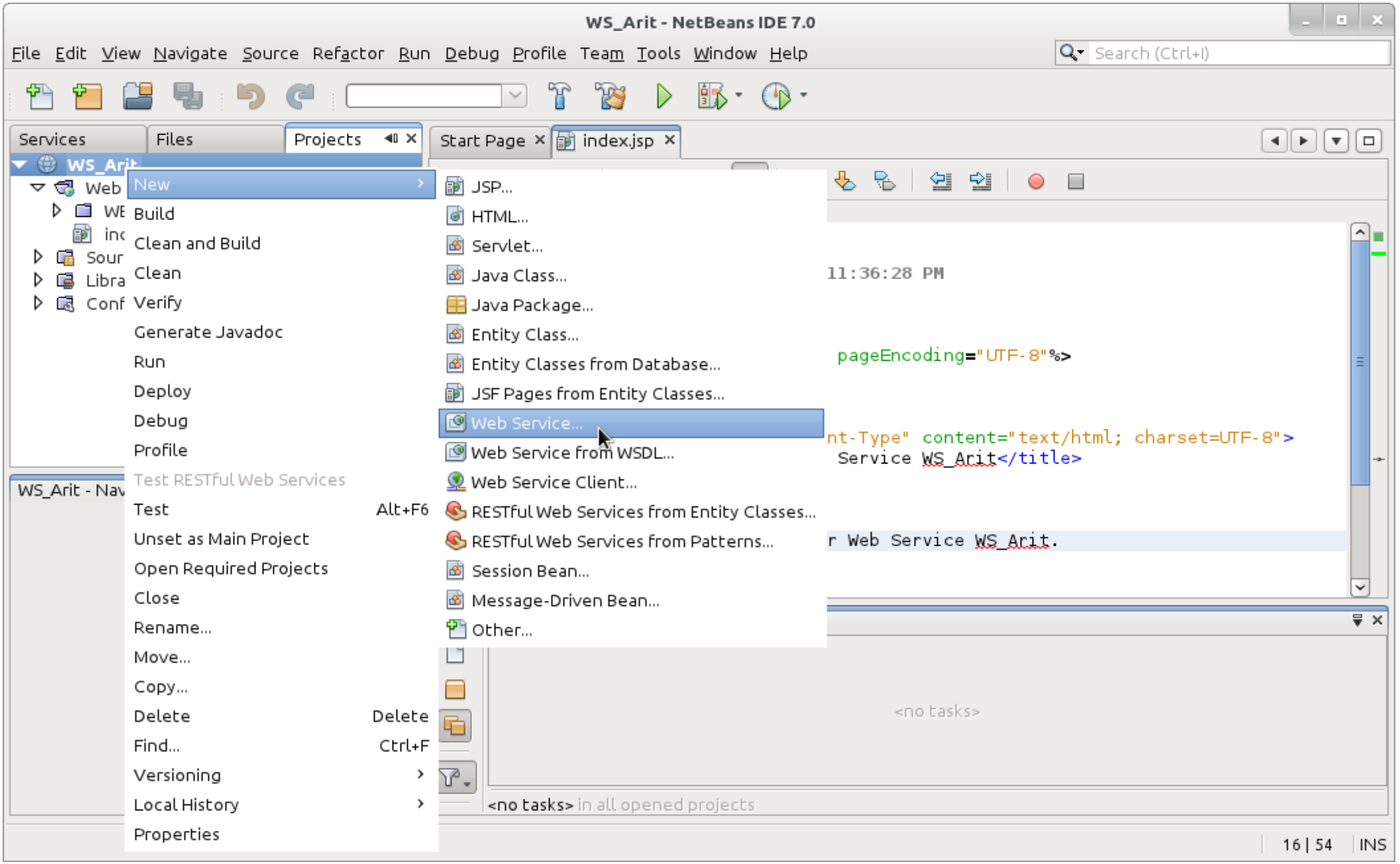
Progetto WS con Netbeans



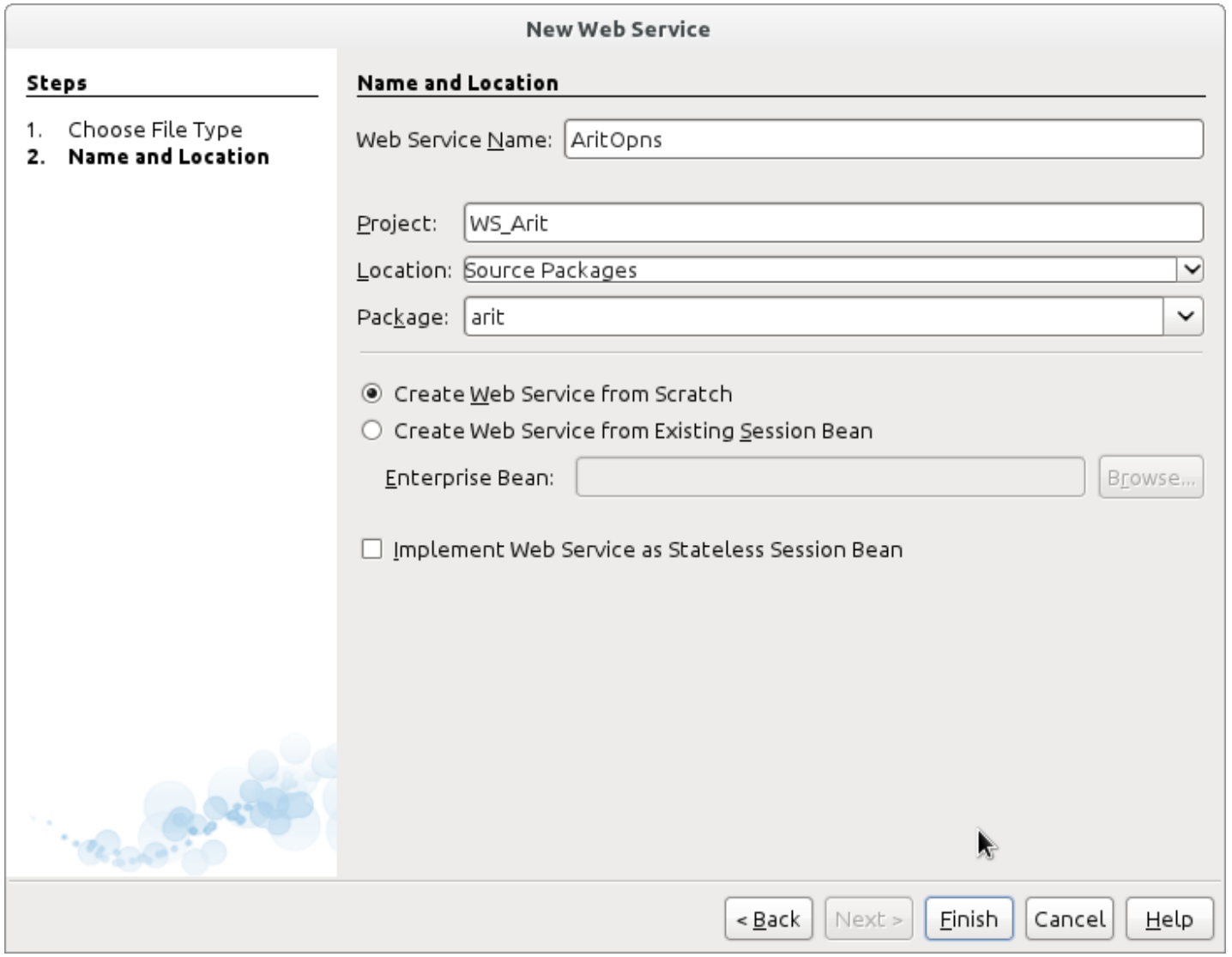
Progetto WS con Netbeans / 2



Definizione nuovo WS

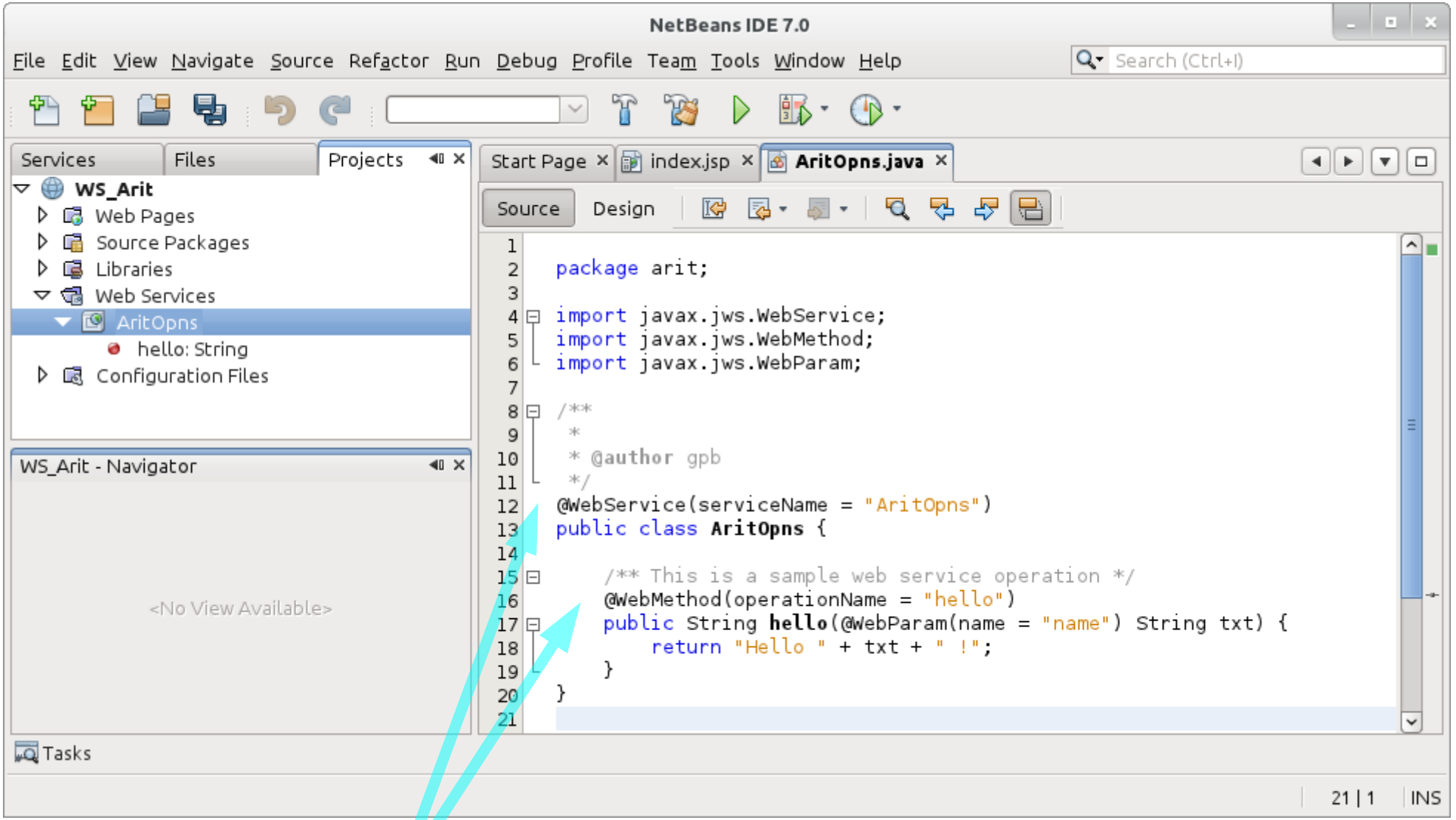


Definizione nuovo WS / 2



- crea template della classe *AritOpns* che implementerà WS *AritOpns*
- alla classe va attribuito un package, p.es. *dmi.lm.ds.arit* o (qui) *arit*

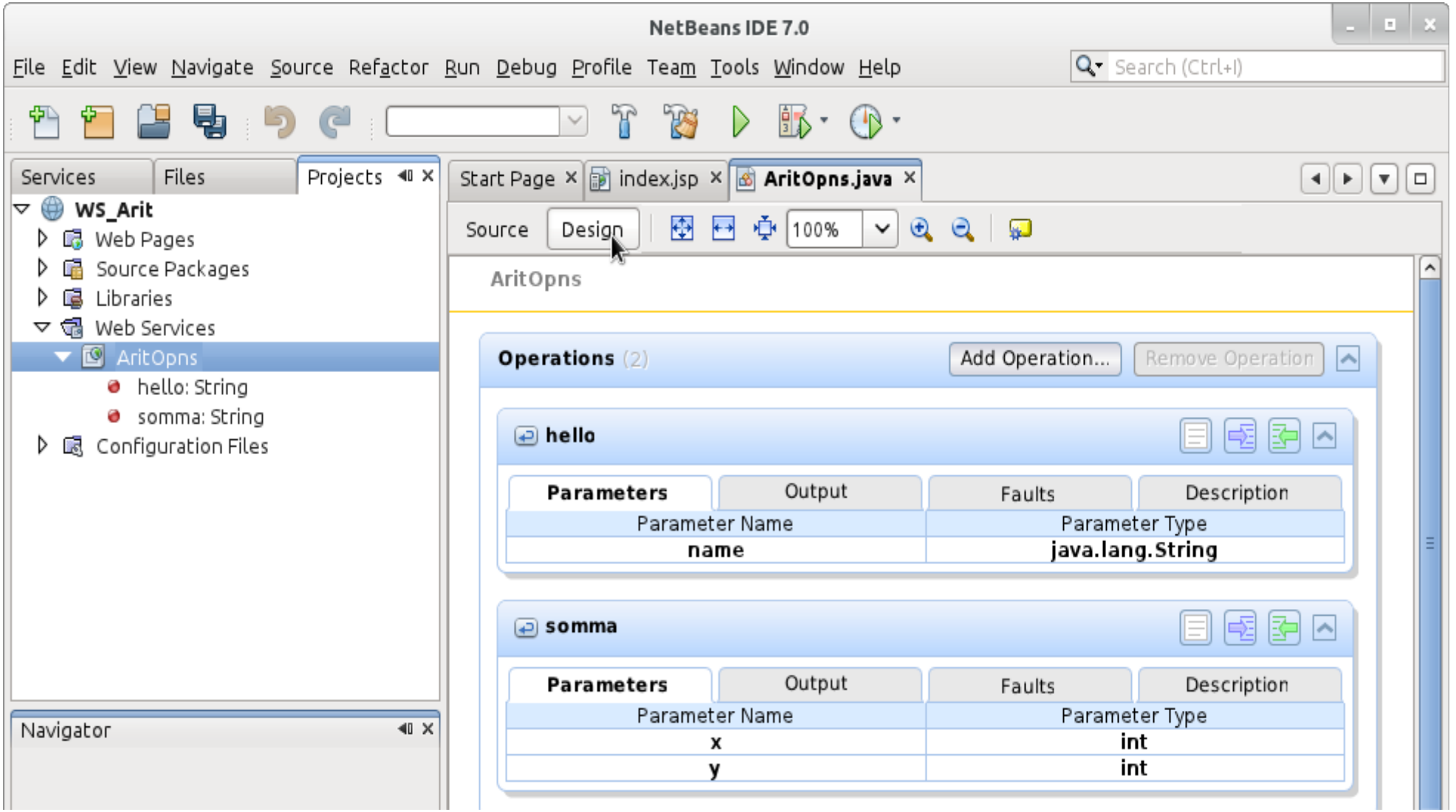
Nuovo WS: template classe Java



- N.B.: annotazioni , marcano il WS e i suoi metodi
- serviranno per generazione automatica di XML WSDL e SOAP

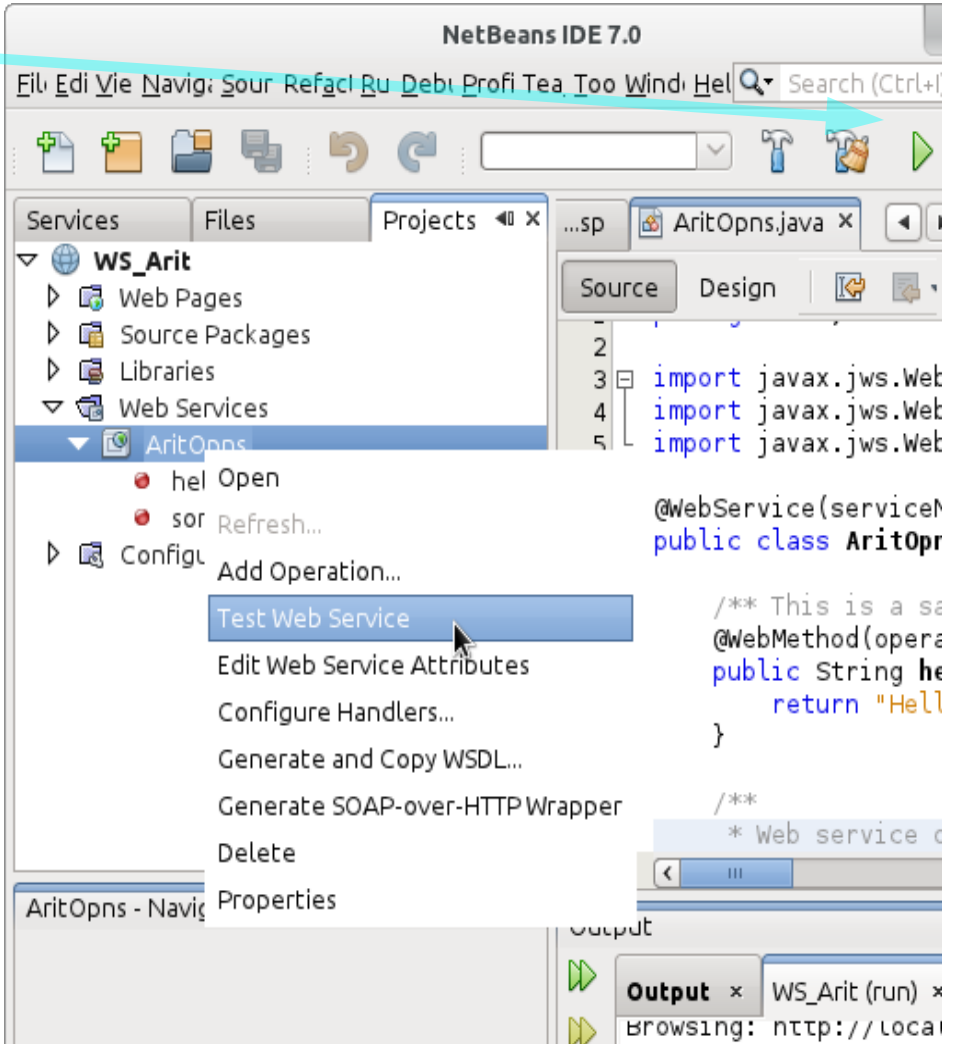
Operazioni WS nella classe Java

- Clic su *Design*...
- *Add Operation* disponibile anche via bottone destro, scheda *Projects*



Avvio WS per test

- Avviare Run (triangolo verde)
  - parte il server Web locale (http://localhost:8080)
  - sul server, avviene il deployment dell'applicazione
  - parte il browser sull'indice index.jsp dell'applicazione
- Avviare Test Web Service
  - parte il browser su una pagina locale di test
  - (funziona solo con GlassFish!)



Test WS (locale)

- NB: verranno mostrati, col valore restituito dal metodo del WS, il WSDL del WS e messaggi SOAP di richiesta e risposta

Caveat

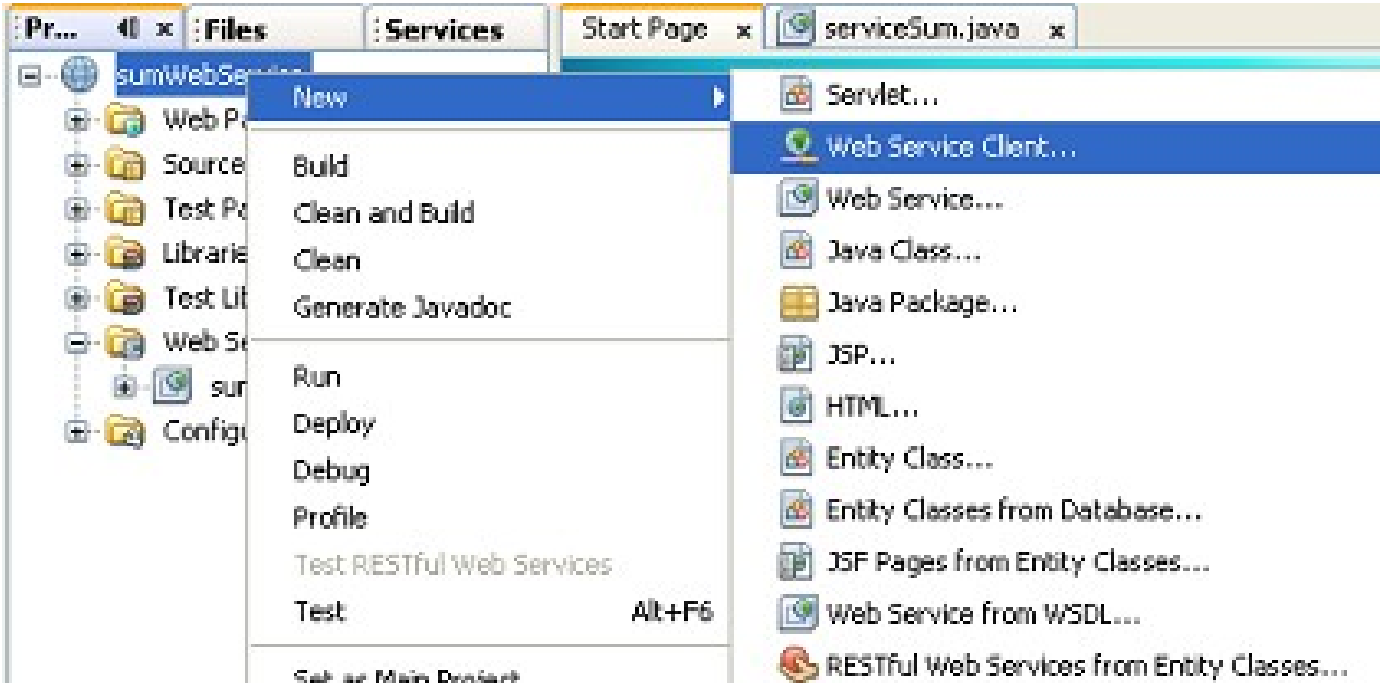
- Se si interviene “a mano” sulla classe del Web Service, quella marcata da Netbeans con annotazione @WebService, si potrebbe voler introdurre un costruttore per la classe.

```
@WebService(serviceName = "AritOpns")
public class AritOpns {
...
}
```

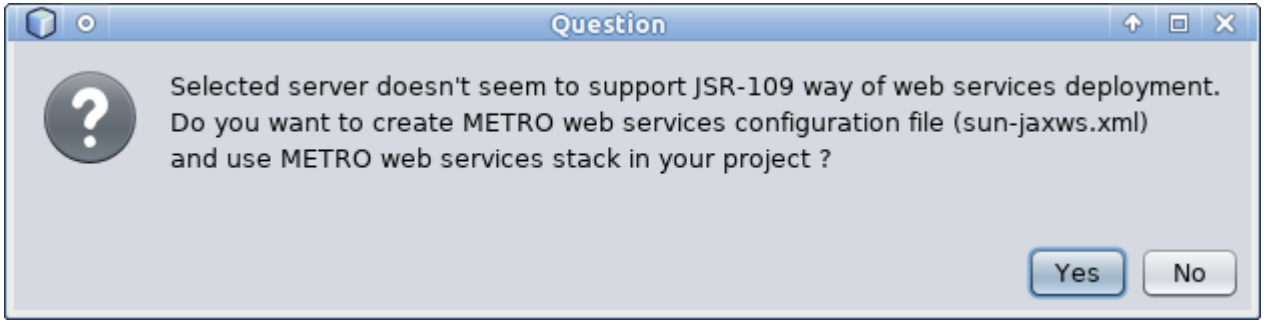
- Nel caso il costruttore deve essere `public` pena possibili crash a runtime!
  - (evidentemente il supporto a runtime del WS potrebbe cercare di instanziare la classe web service dentro il codice di una classe diversa)

Costruire una servlet cliente di WS

- Il nuovo progetto è ancora una Web app
- vi inseriamo un Web Service Client

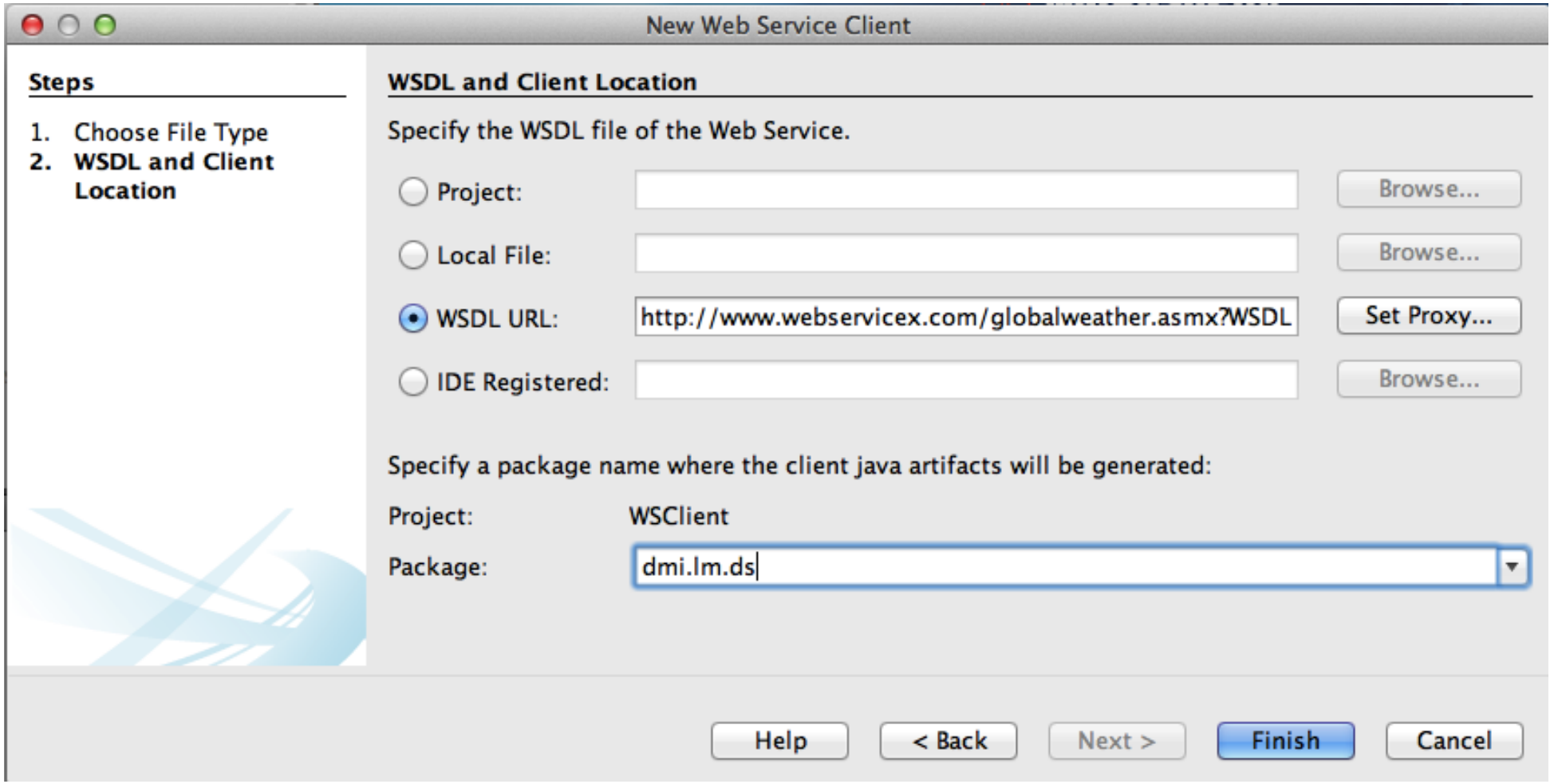


WS con Tomcat



- WS

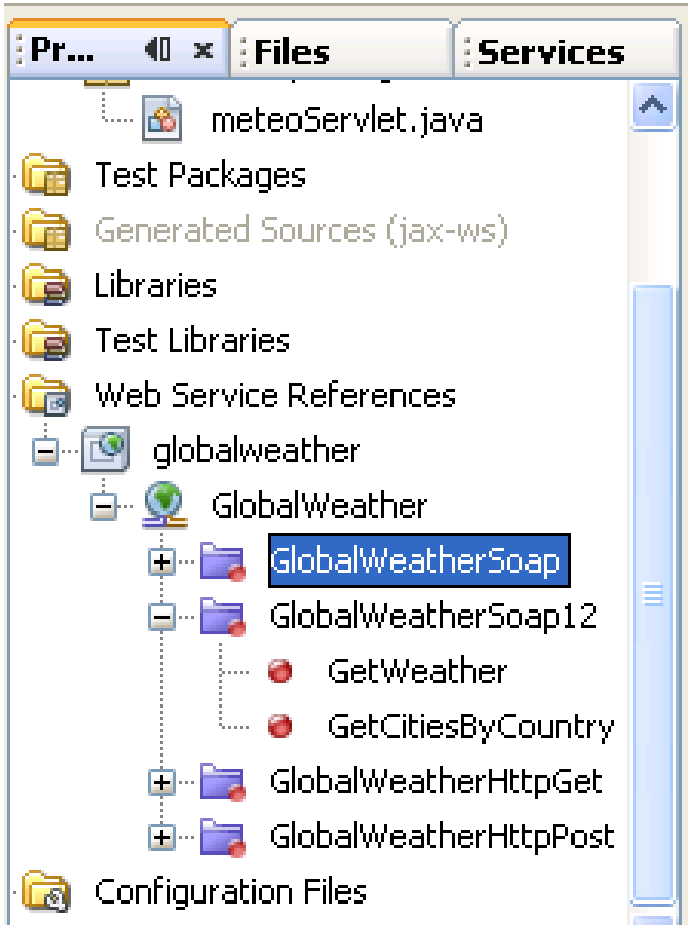
WS client da WSDL remoto



- WSDL URL: http://www.webservicex.com/globalweather.asmx?WSDL

Web Service Reference

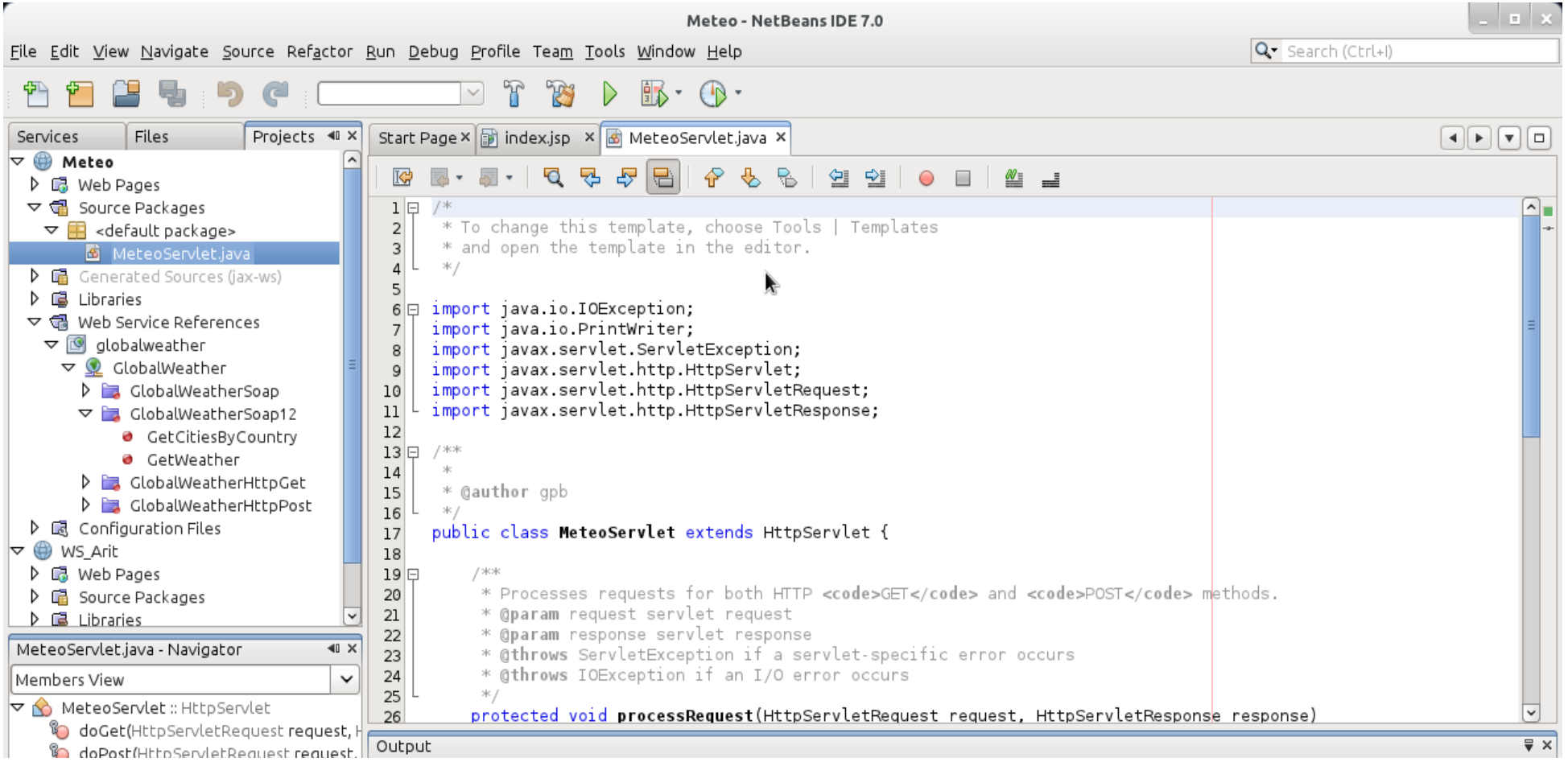
- Viene creata importando un file WSDL



- WSDL file: *globalweather*
- Web service: *GlobalWeather*
- Port: *GlobalWeatherSoap*
- Operazioni: *GetWeather(), GetCitiesByCountry*

Servlet che richiama WS remoto

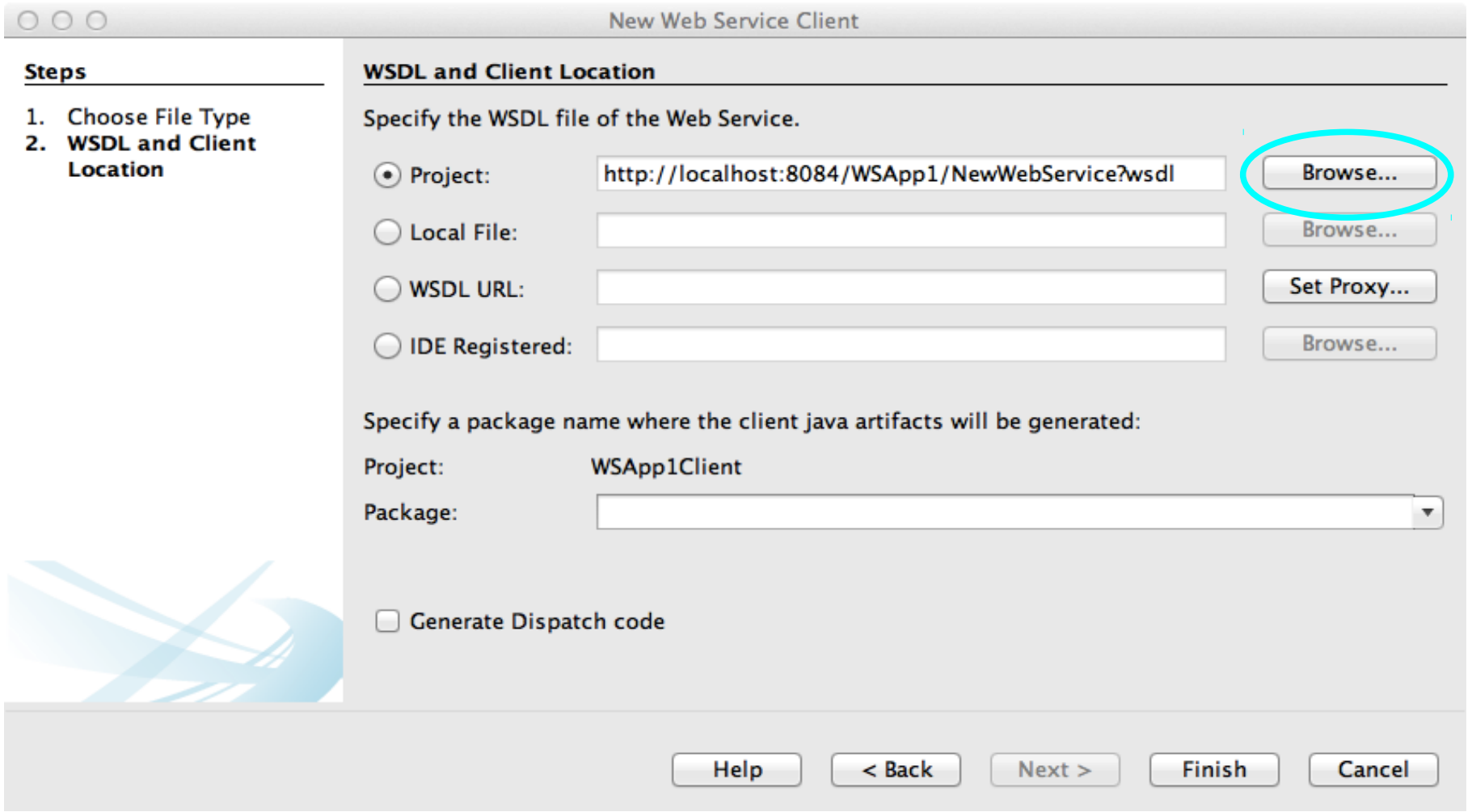
- *New–Servlet*, p.es. (vedi sotto) *MeteoServlet.java*
- Drag-and-drop dell'operazione remota, p.es. *GetCitiesByCountry()*: incorpora il codice Java necessario



Codice da drag-and-drop

```
try { // Call Web Service Operation
    net.webservicex.GlobalWeatherSoap port = service.getGlobalWeatherSoap();
    // TODO initialize WS operation arguments here
    java.lang.String countryName = "";
    // TODO process result here
    java.lang.String result = port.getCitiesByCountry(countryName);
    out.println("Result = "+result);
} catch (Exception ex) {
    // TODO handle custom exceptions here
}
```

Importare WS da propri progetti



- *Project* che implementa i WS da importare deve essere “deployed”, affinché il server locale renda disponibile il WSDL a una URL